

GOOD DEFINITION OF SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
SPRING BOOT IS AN OPINIONATED FRAMEWORK BUILT ON TOP OF
SPRINGFRAMEWORK, IT SIMPLIFIES THE DEVELOPEMENT AS IT PROVIDE
DEFAULT AUTOMATIC CONFIGURATIONS DONE BASED ON THE DEPENDENCIES
SPECIFIED ON USING MAVEN POM.XML THAT ARE ON CLASSPATH .

FOR GRADLE IT BUILD.GRADLE TO SPECIFY DEPENDENCIES .

AND ALSO IT PROVIDES EMBEDDED SERVER LIKE TOMCAT, AND PRODUCTION
READY FEATURES LIKE ACTUATOR FOR MONITORING AND MANAGING THE
APPLICATIONS,

FOR RAPID APPLICATION DEVELOPEMENT OF APPLICATIONS LIKE FOR
MICROSERVICES BASED APPLICATION , IT IS TO DEVELOP QUICKLY.

IN WORLD OF MICROSERVICES , IT HAS TO BE DEVELOP THE MICROSERVICE
APPLICATIONS VERY QUICKLY, SO BY USING SPRINGBOOT IT SOLVE THE
PROBLEM, AS BY AUTOCONFIGURATION FEATURE.

OPINIONATED FRAMEWORK MEANS IT IMPLIES HOW THE FRAMEWORK DESIGNED BY
CREATOR ON BEHALF OF DEVELOPERS MADE THEIR STRONG DECISIONS, HOW
TO REDUCE THE BURDEN OF DEVELOPERS IN CONFIGURATIONS TO THE
APPLICATION , SO IT COMES WITH VARIOUS FEATURES LIKE DEFAULT
CONFIGURATIONS, EMBEDDED SERVER, AND PRODUCTION READY FEATURES, ALL
REDUCE THE CODE OF DEVELOPERS.

SO DEVELOPERS ADHERE TO THESE FEATURES IN USING THESE SPRING BOOT
FRAMEWORK.

EXAMPLES FOR CONFIGURATION
=====

SPRING MVC WITH ANNOTATIONS+XMLCONFIGURATION

XML Configuration (spring-config.xml):
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
Page 1

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
<http://www.springframework.org/schema/beans/spring-beans.xsd>
<http://www.springframework.org/schema/mvc>
<http://www.springframework.org/schema/mvc/spring-mvc.xsd>>

<mvc:annotation-driven />

<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolv
er">

<property name="prefix" value="/WEB-INF/views/" />

<property name="suffix" value=".jsp" />

</bean>

<!-- Other bean configurations -->

</beans>

=====

Controller (HelloController.java):

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;
```

@Controller

public class HelloController {

@GetMapping("/hello")

@ResponseBody

public String sayHello() {
 return "Hello, world!";

}

}

=====

=====

=====

SPRING WITH JAVA CONFIGURATION NO XML CONFIGURATION ONLY
ANNOTATIONS .. USING @CONFIGURATION AND @BEAN .

@COMPONENTSCAN(BASEPACKAGES="PACKAGES" /

@COMPONENTSCAN("PACKANGENAME") OR COMPONENTSCAN(BASEPACKAGGECLASSES=
"CLASSNAMES") FOR CLASSES SPECIFYTING @COMPONENT/@SERVICE @REPOSITORY
@CONTROLLER.

```

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.example")*****
public class AppConfig {
    // No additional configuration is needed

@Bean
    public MyBean myBean() {
        return new MyBean();
    }

    @Bean
    public AnotherBean anotherBean() {
        return new AnotherBean(myBean());
    }
}

import org.springframework.stereotype.Controller;

@Controller
public class MyController {
    public void doSomething() {
        System.out.println("Doing something in the controller");
    }
}

import org.springframework.stereotype.Service;

@Service
public class MyService {
    public void doSomething() {
        System.out.println("Doing something in the service");
    }
}

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationCont
ext;

public class MainApp {
    public static void main(String[] args) {
        // Create the application context using Java configuration
        ApplicationContext context = new

```

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
AnnotationConfigApplicationContext(AppConfig.class);

```
        // Retrieve the beans from the context
        MyController myController =
context.getBean(MyController.class);
        MyService myService = context.getBean(MyService.class);

        // Use the beans
        myController.doSomething();
        myService.doSomething();
    }
}
```

HERE IN APPCONFIG CLASS SPECIFY THE CONFIGURATIONS.

=====

spring with xml configurations , no annotations ...
=====

```
public class Student {
    private String name;
    private int age;
    private School school;

    // Getters and setters

    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("School: " + school.getName());
    }
}
```

```
public class School {
    private String name;

    // Getter and setter

    public void displayInfo() {
        System.out.println("School Name: " + name);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="school" class="com.example.School">
    <property name="name" value="ABC School" />
  </bean>

  <bean id="student" class="com.example.Student">
    <property name="name" value="John Doe" />
    <property name="age" value="20" />
    <property name="school" ref="school" />
  </bean>
</beans>
```

[illegible]

Spring Boot:
With Spring Boot, you can eliminate most of the configuration code.
The main class can be annotated with `@SpringBootApplication`, which
combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`:

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

```
@Controller
public class HomeController {

    @RequestMapping("/")
    public String home() {
        return "index";
    }
}
```

if you do not include the Thymeleaf dependency in your Spring Boot application, the default view resolver used is InternalResourceViewResolver.

FOR SPRINGBOOT

FOR INTERNALVIEW RESOLVER , YOU HAVE TO DO APPLICATION.PROPERTIES IN SPRINGBOOT.

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

VIMP POINTS , , TO NOTE HERE..

=====

DEFAULT SCOPE OF @COMPONENT @CONTROLLER @SERVICE IS SINGLETON ,
AND @BEAN SCOPE IS ALSO SINGLETON
YOU CAN SPECIFY SCOPE USING @SCOPE(PROTOTYPE/

```
<bean id="myBean" class="com.example.MyBean" scope="prototype"/>
```

In the Spring Framework, there are several bean scopes that define the

Page 6

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
lifecycle and visibility of beans. The commonly used bean scopes in Spring are as follows:

***Singleton: This is the default scope in Spring. It means that only one instance of the bean is created for the entire application context.

***Prototype: In this scope, a new instance of the bean is created every time it is requested from the container.

***Request: This scope is specific to web applications. A new instance of the bean is created for each HTTP request.

**Session: This scope is also specific to web applications. A new instance of the bean is created for each HTTP session.

***Global Session: This scope is used in a portlet context where multiple portlets share the same session. A new instance of the bean is created for each global session.

To specify a bean's scope in Spring, you can use the @Scope annotation along with the desired scope value. Here's an example:

java

Copy code

```
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;
```

```
@Component
```

```
@Scope("prototype")
```

```
public class MyPrototypeBean {
    // Bean implementation
```

```
}
```

In the above example, the MyPrototypeBean class is annotated with @Scope("prototype"), indicating that a new instance of the bean should be created whenever it is requested.

When configuring beans using XML in Spring, you can use the scope attribute to define the scope of a bean. Here's an example XML configuration:

xml

Copy code

```
<bean id="myPrototypeBean" class="com.example.MyPrototypeBean"
scope="prototype"/>
```

JUST TO KNOW THE CONCEPTS JUSTTOKNOW CONCEPTS JUST TO KNOW THE
CONCEPTS VERIPM VERYIMPORTNAT

[illegible]

```
@ComponentScan(basePackages = "com.baeldung.componentscan.springapp")
basepackageClasses = class,clas)
or
@ComponentScan("packagename")
```

AUTOMATICALLY DETECT DETECT DETECT --->AND REGISTER REGISTER THE SPING
CLASSES ANNOTATED WITH @COMPONENT @CONTROLLER@SERVICE@REPOSITORY
REGISTER IT IN THE SPRING CONTAINER,

SPRING CREATES AN INSTANCE OF THE CLASS AND INJECT DEPENDENCIES AND MANAGE LIFECYCLE , AND SCOPE PROVIDED.

The `@ComponentScan` annotation in Spring is used to automatically detect and register Spring components within a specified package or packages. It enables component scanning, which is the process of scanning the specified package(s) to find classes annotated with stereotype annotations such as `@Component`, `@Controller`, `@Service`, `@Repository`, etc.

When we say "registering Spring components," it refers to the process of making these components known and managed by the Spring framework. In other words, it allows Spring to recognize and handle these components as part of the application's configuration and runtime environment.

Components in the context of Spring typically refer to classes that fulfill specific roles within the application, such as controllers, services, repositories, or any other class annotated with Spring stereotype annotations (`@Component`, `@Controller`, `@Service`, `@Repository`, etc.).

When a component is registered, Spring performs several tasks:

Instantiation: Spring creates an instance of the component class, typically by using its default constructor. The instantiated object becomes a managed bean within the Spring application context.

Dependency Injection: Spring resolves and injects any dependencies that the component requires. This is typically achieved through constructor injection, setter injection, or field injection. BY USING

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
@AUTOWIRED.*****

Lifecycle Management: Spring manages the lifecycle of the registered components. It initializes the beans by calling any necessary initialization methods, and it can perform additional actions before and after the bean is created and destroyed.*****

Scoping: Spring provides various bean scopes (e.g., singleton, prototype, request, session, etc.) that define the lifecycle and visibility of the beans. By registering components, you can specify the desired scope for each bean.*****DEFAULT SCOPE IS SINGLETON

AOP (Aspect-Oriented Programming): Spring allows you to apply aspects (cross-cutting concerns) to registered components using AOP. Aspects provide additional functionality, such as logging, security, or transactions, that can be applied to multiple components.

By registering components with Spring, you delegate the responsibility of managing their lifecycle, dependencies, and configuration to the Spring framework. This allows for loose coupling between components, better separation of concerns, and easier integration with other Spring features and modules.

The @ComponentScan annotation, along with other stereotype annotations (@Component, @Controller, @Service, @Repository), facilitates the automatic detection and registration of these components, reducing the need for explicit configuration and enabling a more modular and flexible development approach.

VERY IMPORTANT.

=====

BY DEFAULT ALL BEAN CLASS ARE INSTANTIATED LAZILY ONLY... WHEN
U CALL BEAN WITH BEANFACTORY.GET(BEAN ID/BEANNAME) IT IS
INSTANTIATED.
OR APPLICATIONCONTEXT APTX.GET(BEANID/BEANNAME) ONLY BEAN CLASS IS
INSTANTIATED , OTHERWISE NOT.

When you use Spring's XML-based configuration with <bean> tags and property tags, the beans are defined in the configuration file, but they are not instantiated immediately when the Spring container starts.

The Spring container creates and manages beans on-demand, based on the application's need for them. This is known as lazy initialization or lazy loading. It means that beans are instantiated only when they are requested by the application.

When you call BeanFactory.getBean(beanId) or
ApplicationContext.getBean(beanId), the container checks if the bean

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
with the specified ID (or name) has already been instantiated. If the bean exists and has been created, the existing instance is returned. If the bean does not exist, the container creates a new instance of the bean, registers it, and returns it.

So, in your main class, when you call `BeanFactory.getBean(beanId)` or `ApplicationContext.getBean(beanId)`, only the requested bean is created and returned. The Spring container does not instantiate all the beans defined in the XML configuration at once. It follows a lazy initialization strategy to create beans only when needed, conserving resources and improving performance.

This lazy initialization behavior allows for better memory management, especially in large-scale applications with many beans. It ensures that resources are allocated only to the beans that are actually required at runtime, rather than instantiating all beans upfront.

It's worth noting that there are ways to eagerly instantiate beans during the startup of the Spring container if needed. For example, you can use the `lazy-init="false"` attribute in the `<bean>` tag or use the `@Lazy(false)` annotation on individual bean definitions to specify eager initialization.

In summary, with XML-based configuration and `<bean>` tags, beans are lazily instantiated by the Spring container when requested. The container creates beans on-demand, ensuring efficient resource utilization and improving application performance.

User

WHEN U SPECIFY WITH BEAN TAG WITH LAZ-INIT=FALSE AND FOR
@COMPONENT CLASS USING @LAZY(FALSE) THEN ALL BEAN CLASS
DEFINITIONS TO EAGAER INSTANTIATION

ChatGPT

Yes, when you specify `lazy-init="false"` in the `<bean>` tag or use `@Lazy(false)` on a class annotated with `@Component`, it instructs the Spring container to eagerly instantiate the bean during the startup process.

Here's how it works:

XML Configuration with `lazy-init="false"`:

xml

Copy code

```
<bean id="myBean" class="com.example.MyBean" lazy-init="false" />
```

In this example, the `<bean>` tag explicitly sets `lazy-init="false"`, indicating that the bean should be eagerly instantiated during the container startup.

Component Class with @Lazy(false):

```
java
Copy code
@Component
@Lazy(false)
public class MyComponent {
    // Class implementation
}
```

In this example, the @Lazy(false) annotation is used on the MyComponent class. It tells the Spring container that the bean should be eagerly initialized during the container startup.

When you use these configurations, all bean instances defined with lazy-init="false" or @Lazy(false) will be eagerly instantiated when the Spring container starts. The container will create instances of these beans even if they are not immediately needed by the application.

Eager initialization can be useful in certain scenarios, such as initializing beans that perform costly operations during their instantiation or beans that need to be available immediately for other beans' dependencies.

It's important to note that the default behavior of Spring is to lazily initialize beans. So, if you don't specify lazy-init="false" or @Lazy(false), the beans will be lazily instantiated when they are first requested by the application.

In summary, setting lazy-init="false" in XML configuration or using @Lazy(false) on a @Component class allows for eager initialization of the corresponding beans during the container startup. This ensures that the beans are instantiated immediately, regardless of whether they are immediately needed or not.

MKYONG EXAMPLE

=====

WITHOUT USING @COMPONENT, @SERVICE ,@CONTROLELR.

Declares Components Manually

See a normal way to declare a bean in Spring.

Normal bean.

```
package com.mkyong.customer.dao;
```

```
public class CustomerDAO
{
    @Override
    public String toString() {
        return "Hello , This is CustomerDAO";
    }
}
DAO layer.
```

```
package com.mkyong.customer.services;
```

```
import com.mkyong.customer.dao.CustomerDAO;
```

```
public class CustomerService
{
    CustomerDAO customerDAO;

    public void setCustomerDAO(CustomerDAO customerDAO) {
        this.customerDAO = customerDAO;
    }

    @Override
    public String toString() {
        return "CustomerService [customerDAO=" + customerDAO +
    "];
    }
}
```

Bean configuration file (Spring-Customer.xml), a normal bean configuration in Spring.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
    <bean id="customerService"
class="com.mkyong.customer.services.CustomerService">
        <property name="customerDAO" ref="customerDAO" />
    </bean>
    <bean id="customerDAO"
class="com.mkyong.customer.dao.CustomerDAO" />
```

</beans>
Run it

```
package com.mkyong.common;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.mkyong.customer.services.CustomerService;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(new String[]
{"Spring-Customer.xml"});

        CustomerService cust =
(CustomerService)context.getBean("customerService");
        System.out.println(cust);

    }
}
output
```

CustomerService [customerDAO=Hello , This is CustomerDAO]

freestar

2. Auto Components Scanning

Now, enable Spring auto component scanning features.

Annotate with @Component to indicate this is class is an auto scan component.

```
package com.mkyong.customer.dao;

import org.springframework.stereotype.Component;

@Component
public class CustomerDAO
{
    @Override
    public String toString() {
        return "Hello , This is CustomerDAO";
    }
}
```

DAO layer, add @Component to indicate this is an auto scan component

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
also.

```
package com.mkyong.customer.services;

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

import com.mkyong.customer.dao.CustomerDAO;

@Component
public class CustomerService
{
    @Autowired
    CustomerDAO customerDAO;

    @Override
    public String toString() {
        return "CustomerService [customerDAO=" + customerDAO +
    "];
    }
}
```

Put this “context:component” in bean configuration file, it means, enable auto scanning feature in Spring. The base-package is indicate where are your components stored, Spring will scan this folder and find out the bean (annotated with @Component) and register it in Spring container.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd">
    <context:component-scan base-package="com.mkyong.customer" />
</beans>
Run it
```

```
package com.mkyong.common;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
```


GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT

***Bean Registration: You register a bean class with the Spring container either through annotations (@Component, @Service, etc.) or programmatically using configuration files or code.

***Bean Instantiation: When the container starts up or when the bean is first requested, the container creates an instance of the bean class using either a no-argument constructor or constructor injection, depending on the configuration.

***Dependency Injection: If the bean has dependencies (other beans that it depends on), the container automatically resolves and injects those dependencies into the bean. This can be done through constructor injection, setter injection, or field injection, depending on the configuration.

***Bean Lifecycle Management: The container manages the lifecycle of the bean, which includes initializing the bean (calling initialization methods if defined), allowing the bean to perform any necessary setup operations. During the lifecycle, the bean can participate in various container-managed callbacks, such as @PostConstruct and @PreDestroy methods.

****Bean Availability: Once the bean is instantiated and its dependencies are injected, it is available for use throughout the application. Other components or services can access the bean by requesting it from the container.

*****By having the Spring container manage the beans, you benefit from automatic dependency injection, lifecycle management, and easy access to the beans throughout your application. This simplifies the development process and promotes modular and loosely coupled application design.

FOR DYNAMIC REGISTER OF BEAN CLASS PROGRAMMATICALLY THE CODE IS ..
=====

YOU MUST WRITE A CLASS SDRSNAClass JUST A CLASS , YOU CAN WRITE WITH SETTER METHODS IF U WANT , DEPENDS UPON U

```
CLASS SDRSNA{}
```

OR

```
class SdrSnaClass {  
    private String property1;  
    private int property2;
```


GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT

```
public void setProperty1(String property1) {
    this.property1 = property1;
}

public void setProperty2(int property2) {
    this.property2 = property2;
}

public void doSomething() {
    System.out.println("SdrSnaClass: property1 = " + property1);
    System.out.println("SdrSnaClass: property2 = " + property2);
}
}
```

example of dynamically registering a bean class (SdrSnaClass) with property values using GenericBeanDefinition and BeanDefinitionRegistry:

```
import org.springframework.beans.MutablePropertyValues;
import
org.springframework.beans.factory.support.BeanDefinitionBuilder;
import
org.springframework.beans.factory.support.BeanDefinitionRegistry;
import
org.springframework.beans.factory.support.GenericBeanDefinition;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class DynamicBeanRegistrationExample {
    public static void main(String[] args) {
        // Create an application context
        AnnotationConfigApplicationContext context = new
        AnnotationConfigApplicationContext();

        //////////Application context = new
        ClassPathXmlApplicationContext("applicationContext.xml");*****
        *****ALSO IF SPECIFIED CONFGFIRUATION XML FILE .. OTHER
        WIRESE NOT.

        // Get the BeanDefinitionRegistry
        BeanDefinitionRegistry registry = (BeanDefinitionRegistry)
        context.getBeanFactory();
```

```

// Create a GenericBeanDefinition for the bean you want to
register
GenericBeanDefinition beanDefinition = new
GenericBeanDefinition();
    beanDefinition.setBeanClass(SdrSnaClass.class);

    // Customize the bean definition with property values
MutablePropertyValues propertyValues = new
MutablePropertyValues();
    propertyValues.add("property1", "value 1");
    propertyValues.add("property2", 42);
    beanDefinition.setPropertyValues(propertyValues);

// Register the bean definition with the
BeanDefinitionRegistry
    registry.registerBeanDefinition("sdrSnaBean", beanDefinition);

// Refresh the application context to apply the changes
    context.refresh();

// Retrieve the dynamically registered bean
SdrSnaClass sdrSnaBean = context.getBean(SdrSnaClass.class);
sdrSnaBean.doSomething();

// Close the application context
    context.close();
}
}

class SdrSnaClass {
    private String property1;
    private int property2;

    public void setProperty1(String property1) {
        this.property1 = property1;
    }

    public void setProperty2(int property2) {
        this.property2 = property2;
    }

    public void doSomething() {
        System.out.println("SdrSnaClass: property1 = " + property1);
        System.out.println("SdrSnaClass: property2 = " + property2);
    }
}

```

YOU CAN SPECIFY SPRING CONFIGURATION XML FILE .. DECLARE , NOT
NEEDED, IF U CAN PROVIDE DO IT ALSO , NO PROBLEM.

IN PLACE OF ANNOTATIONCONFIGAPPLICATIONCONTEXT U CAN USE DIRECTLY APPLICATIONCONTEXT WITH CLASSPATHXMLAPPLICATIONCONTEXT LIKE THESE , IS CORRECT

ChatGPT

Yes, that is correct. Instead of using AnnotationConfigApplicationContext, you can use ApplicationContext with ClassPathXmlApplicationContext to load the application context from XML configuration files.

The ApplicationContext interface is the central interface in Spring's application context hierarchy, and ClassPathXmlApplicationContext is an implementation of ApplicationContext that loads the context from XML files on the classpath.

Here's an example of using ClassPathXmlApplicationContext:

java

Copy code

```
ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");*****
*****
```

In this example, the applicationContext.xml file is located on the classpath and contains the bean definitions and other configuration for the application context.

Using ClassPathXmlApplicationContext is suitable when you want to configure your application context using XML-based configuration files instead of annotations or Java-based configuration classes.

SO BY TAKING A CLASS , REGISTERINGG DYNAMICALLY IN THE SPRING CONTAINER. WITH ITS PROPERTY VALUES , ..

=====

USING

CLASS GENERICBEANDEFINITION

AND INTERFACE BEANDEFINTIONREGISTRY ..

```
Package org.springframework.beans.factory.support
public class GenericBeanDefinition
extends AbstractBeanDefinition
```

GenericBeanDefinition is a one-stop shop for declarative bean definition purposes. Like all common bean definitions, it allows for specifying a class plus optionally constructor argument values and property values. Additionally, deriving from a parent bean definition can be flexibly configured through the "parentName" property. In general, use this GenericBeanDefinition class for the purpose of registering declarative bean definitions (e.g. XML definitions which a bean post-processor might operate on, potentially even reconfiguring the parent name). Use RootBeanDefinition/ChildBeanDefinition where parent/child relationships happen to be pre-determined, and prefer RootBeanDefinition specifically for programmatic definitions derived from factory methods/suppliers.

Since:
2.5

```
PACKAGE org.springframework.beans.factory.support
```

```
public interface BeanDefinitionRegistry  
extends AliasRegistry  
Interface for registries that hold bean definitions, for example  
RootBeanDefinition and ChildBeanDefinition instances. Typically  
implemented by BeanFactories that internally work with the  
AbstractBeanDefinition hierarchy.
```



```

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

import com.java2novice.dao.InventoryDao;

@Service("j2nInventoryService")
public class J2nInventoryService {

    @Autowired
    private InventoryDao inventoryDao;

    public void testInventoryService(){
        System.out.println("Inside service class,
testInventoryService()");
        inventoryDao.printInventoryDaoStatus();
    }
}

package com.java2novice.dao;

import org.springframework.stereotype.Component;

@Component
public class InventoryDao {

    public void printInventoryDaoStatus(){
        System.out.println("Inside InventoryDao & status method");
    }
}

```

Here is the xml based configuration file, to enable component auto scanning include component-scan tag, and specify base-package. The base-package value should be package name and will be scanned for the components with in the packages and its sub-packages.

```

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="com.java2novice" />
</beans>
Spring bean demo class:

```

```

package com.java2novice.test;

```

```

GOOD DEFINITION OS SPRING BOOT    BY SURESHKUMARSDRSNA    TKWIT
import org.springframework.context.ConfigurableApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.java2novice.service.J2nInventoryService;

public class SpringDemo {

    public static void main(String a[]){
        String confFile = "applicationContext.xml";
        ConfigurableApplicationContext context
                                = new
ClassPathXmlApplicationContext(confFile);
        J2nInventoryService inventoryService =
                                (J2nInventoryService)
context.getBean("j2nInventoryService");
        inventoryService.testInventoryService();
    }
}

```

he framework's creators have made decisions on behalf of the developers, based on their expertise and best practices. These decisions are embedded into the framework's design and architecture, and developers are expected to follow them.

The term "opinionated" implies that the framework has a strong opinion on how things should be done and encourages developers to adhere to those opinions. It often comes with predefined patterns, default configurations, and conventions to streamline development and ensure consistency across projects.

Rapid Development: An opinionated framework reduces the number of decisions developers need to make, allowing them to focus on the application's business logic rather than spending time on low-level implementation details.

Consistency: With predefined conventions and patterns, an opinionated framework ensures that the application follows a consistent structure and design. This makes the codebase easier to understand, maintain, and collaborate on, especially when multiple developers are working on the same project.

There are several reasons why microservice applications can be built quickly:

Decoupled Development: Microservice architecture allows teams to work on different services independently. Each service can be developed, tested, and deployed separately, without having to coordinate with other teams. This decoupled development approach enables parallel development and speeds up the overall development process.

Technology Flexibility: Microservices give developers the flexibility to choose different technologies, frameworks, and programming languages for each service. This means that teams can use the most suitable tools and technologies for their specific service requirements, which can result in increased productivity and faster development.

Scalability: Microservices are designed to be independently scalable. Each service can be scaled horizontally or vertically based on its specific needs. This scalability allows developers to quickly adapt to changing user demands and handle increased traffic or workload by scaling the relevant services without affecting others.

Faster Deployment: With microservices, it's easier to achieve Continuous Deployment and Continuous Integration practices. Each service can be deployed independently, allowing for faster deployment cycles. Developers can quickly roll out updates, bug fixes, and new features without impacting the entire application, leading to faster time-to-market.

Specialized Teams: Microservice architecture allows teams to specialize in specific services. This specialization enables developers to become experts in their respective areas, leading to faster development and more efficient problem-solving. Specialized teams can work more quickly and effectively on their specific services, reducing development time.

Improved Testing: Microservices are relatively smaller and focused, making them easier to test thoroughly. Each service can have its own dedicated test suite, and testing can be automated to a greater extent. This allows for faster feedback loops, quicker bug identification, and easier maintenance, resulting in faster development cycles.

Agile Development: Microservices align well with Agile development principles, such as iterative development, frequent releases, and close collaboration with stakeholders. The modular nature of microservices allows for incremental development, where new features can be added or modified without impacting the entire application. This agility promotes faster development and adaptability to changing requirements.

GOOD DEFINITION OF SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
on the classpath can be cumbersome and error-prone.

The main advantage of using a build tool like Maven or Gradle with Spring Boot is that it simplifies the dependency management process. These build tools handle downloading and managing the required libraries, along with their transitive dependencies. They also provide a standardized way to configure and build your project, making it easier to manage dependencies, handle version conflicts, and package your application for deployment.

@@

OPINIONATED MEANING IN TELUGU ABHIPRAYAPADADAM.

What does it mean when someone is strongly opinionated?

An opinionated person is certain about their beliefs, and expresses their ideas strongly and often:

SPRINGBOOT IS OPINIONATED FRAMEWORK BECAUSE

Spring Boot is opinionated because it follows the opinionated default configuration that reduces developer efforts to configure the application.

SPRING BOOT IS OPINIONATED FRAMEWORK AS IT PERFORMS DEFAULT AUTOMATIC CONFIGURATIONS DONE BASED ON DEPENDENCIES THAT ARE ON CLASSPATH, SO IT REDUCES THE DEVELOPER TIME TO CONFIGURE THE APPLICATION,

Spring Boot uses sensible defaults, "opinions", mostly based on the classpath contents.

- For example
 - Sets up a JPA Entity Manager Factory if a JPA implementation is on the classpath.
 - Creates a default Spring MVC setup, if Spring MVC is on the classpath.
- Everything can be overridden easily
 - But most of the time not needed

WHILE WORKING ON BIG WEB BASED APPLICATION, IT IS VERY COMPLEX TO CONFIGURE ALL THE DEPENDENCIES PRESENT, BY USING AUTOCONFIGURATION FEATURE IT AUTOMATICALLY DEFAULT CONFIGURATION DONE BY BASED ON DEPENDENCIES JARS ON CLASSPATH, IT REDUCES DEVELOPER TIME.

Spring Boot is lightweight and an amazing tool for Java developers seeking to develop and run enterprise-grade applications quickly. It hides a lot of complexity behind the scene, which allows you to avoid a lot of boilerplate code and go straight to coding

LOT OF COMPLEXITY IN CONFIGURAION IS HIDE , WHICH ALLOWS THE DEVELOPER TO AVOID A LOT OF BOILERPLATE CODE AND GO STRAIGHT WAY TO CODING.

""""""

Spring Boot is opinionated because it follows the opinionated default configuration that reduces developer efforts to configure the application.

""""""

It follows opinionated default configuration so it reduces the developer efforts. Spring boot always uses sensible opinions, mostly based on the class path contents. So it overrides the default configuration.

What is spring boot what do you mean by opinionated defaults how autoconfiguration works?

Spring Boot just decides on a set of default configured beans which you can override if you want. For example if you include the spring boot starter pom for jpa, you'll get autoconfigured for you an in memory database, a hibernate entity manager, and a simple datasource.

SPRING BOOT AUTOCONFIGURATION.

=====

SPRING MVC WE HAVE TO DO A LOT OF CONFIGURATION TO BE SET.

LIKE DISPATCHER-SERVVLET.XML AND WEB.XML, AND ALSO CONFIGURRE FOR VIEWRESOLVER AND ALSO FOR STATIC RESOURCES TO CONFIGURE MVC:RESOURCES TAG .FOR JPA/HIBERNATE YOU HAVE TO CONFIGURE THE DATASOURCE, ENTITYMANAGER /SESSIONFACTORY FOR HIBERNATE , AND ALSO TRANSACTION MANAGER .

ALSO FOR TRANSITIVE DEPENDENCIES YOU HAVE TO SEE , AND ALSO FOR VERSION COMPATIBILITY YOU HAVE TOLOOK FOR IT FOR SPECIFY IT.

DEPENDENCY MANAGEMENT IS MECHANISM FOR CENTRALIZING DEPENDENCY INFORMATION.

IN WORLD OF MICROSERVICES , IT HAS TO BE DEVELOP THE MICROSERVICE APPLICATIONS VERY QUICKLY, SO BY USING SPRINGBOOT IT SOLVE THE PROBLEM, AS BY AUTOCONFIGURATION FEATURE.

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT

FOR SPRING MVC, YOU HAVE TO SPECIFY ALL DEPENDENCIES LIKE SERVER, JSTL AND WHAT SPRINGWEB MVC, FOR VALIDATION LIKE HIBERNATE VALIDATOR AND IF ANY VIEW TECHNOLOGIES LIKE THYMELEAF JACKSON YOU HAVE ADD THAT DEPENDENCY FOR THAT .

AND ALSO FOR HIBERNATE , YOU HAVE ADD DEPENDENCIES FOR DATABASE, AND HIBERNATE

SO BY USING SPRING BOOT AUTOCOFNGIURATION FREATEURE SOLVE THESE BY USING SPRINGBOOT STARTERS .

NO NEED TO SPECIFY FOR EVERY DEPENDENCY IN POM.XML.

SPRNG-BOOT-STARTER

SPRING-BOOT-STARTER-WEB

SPRING-BOOT-STARTER-WEBFLUX

SPRING FRAMEWORK VS SPRINGBOOT DIFFERNCE , WHAT ADVANTAGES OF SPRINGBOOT GIVES
ChatGPT

2 / 2

The Spring framework and Spring Boot are related but serve different purposes. Here are the key differences between them and the advantages that Spring Boot offers:

Configuration: In the Spring framework, developers need to manually configure various components, such as data sources, security, and web configuration. Spring Boot, on the other hand, embraces convention over configuration and provides auto-configuration. It automatically

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT
configures many components based on the dependencies and sensible defaults, reducing the amount of manual configuration required.

Dependency Management: Spring Boot simplifies dependency management by providing a curated set of starter dependencies. These starters include all the necessary dependencies for common use cases, such as web applications, data access, security, and more. It also manages transitive dependencies and version compatibility, ensuring a smooth and hassle-free dependency management experience.

Embedded Server: Spring Boot includes an embedded server (Tomcat, Jetty, or Undertow) by default. This means you don't have to deploy your application to a separate server during development and testing. It simplifies the deployment process and makes it easier to create self-contained, executable JAR files that can be run independently.

Auto-configuration: Spring Boot's auto-configuration feature automatically configures various components based on the classpath and the dependencies used in the project. It analyzes the project's dependencies and environment to configure components like data sources, message brokers, security, and more. This reduces the need for manual configuration and helps in getting started quickly with minimal effort.

Production-Ready Features: Spring Boot provides a range of production-ready features out-of-the-box, such as metrics, health checks, centralized configuration, and logging. These features enable developers to build applications that are well-suited for production environments without having to spend time and effort on boilerplate configuration.

Actuator: Spring Boot Actuator is a module that provides endpoints and monitoring capabilities for managing and monitoring Spring Boot applications. It offers insights into application health, metrics, diagnostics, and more. Actuator endpoints can be easily exposed for monitoring and management purposes, making it convenient to monitor and manage your application in production.

Developer Productivity: Spring Boot aims to boost developer productivity by providing a streamlined development experience. With its opinionated approach, sensible defaults, and built-in features, developers can focus more on writing business logic rather than spending time on tedious configuration tasks. It enables rapid prototyping and faster development cycles.

In summary, Spring Boot builds on top of the Spring framework and simplifies the development of Spring applications by providing convention-over-configuration, automatic dependency management, embedded server, production-ready features, and a range of developer-friendly tools. It enhances developer productivity, reduces boilerplate code, and facilitates the rapid development of production-ready applications.

GOOD DEFINITION OS SPRING BOOT BY SURESHKUMARSDRSNA TKWIT

=====

=====

=====

=====

=====

=====
