

List of Annotations

=====

1. @Configuration
2. @ComponentScan
3. @Autowired
4. @Component
5. @Service
6. @Controller(will be used in MVC)
7. @Repository
8. @Import
9. @PropertySource
10. @Bean
11. @Qualifier
12. @Scope
13. @Lazy
14. @Primary
15. @Required
16. @Value
17. @ImportResource
18. @PostConstruct
19. @PreDestroy

100%Code driven SpringApp development/Java Config Approach of SpringApp development

=====

Advantages

- a. XMLBased cfg can be avoided in maximum cases
- b. Improves the readability
- c. Debugging becomes easy
- d. Foundation to learn SpringBoot

ThumbRule

=====

1. Configure userDefined classes as Springbean using Stereotype annotations(@Component) and link them with Configuration class alternative to SpringBean cfg file(xml file) using @ComponentScan

note: Java class that is annotated with @Configuration automatically becomes Configuration class

||

2. Configure PreDefined class as Spring beans using @Bean methods(method that is annotated with @Bean) of @Configuration class.

3. use AnnotationConfigApplicationContext class to create an IOC container having @Configuration class as the input classname

Note: @Configuration class is internally a Spring bean becoz @Configuration internally contains @Component.

ApplicationContext container

1. It is an extension of BeanFactory
2. Implementation classes of ApplicationContext(I)
 - a. FileSystemXmlApplicationContext(standalone)
 - b. ClassPathXmlApplicationContext(standalone)
 - c. XmlWebApplicationContext(SpringMVC apps)
 - d. AnnotationConfigApplicationContext(Standaloneapp's)
 - e. AnnotationConfigWebApplicationContext(SpringMVC apps)

Configuration of container in purejava style

```
=====
@Configuration
@ComponentScan(basePackages={"in.ineuron"})
@Import(value=PersistConfig.class)
public class AppConfig{

}

ClientApp
=====
public class ClientApp{
    public static void main(String[] args){
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
    }
}
```

SpringBoot ==> SpringFramework - NoXML +Autoconfiguration(purejavacode)
+EmbeddedDatabase+EmbeddedServer+.....

To work with different environments of spring application, spring boot has given few starters like

- a. spring-boot-starter-jdbc
- b. spring-boot-starter-mail
- c. spring-boot-starter-aop
- d. spring-boot-starter-datajpa
- e. spring-boot-starter-security
- f. spring-boot-starter.mvc

.....

SpringBoot

1. These application can be created using

- a. STS(SpringToolSuite)
- b. Eclipse using STS
- c. Spring.io(Intializer)
- d. using command line runners

Building application using SpringBoot[eclipse version(2021-03)]

Step1:: Keep the following software ready

=> Eclipse IDE with STS plugin(SpringToolSuite)

To install sts plugin :: Help menu-> Eclipse market place -> search for sts(3.9.14)

select all -> click on install->

accept terms and conditions -> restart IDE.

=>Plugin is a patch software that provides additional features/functionalities to existing software.

=>STS plugin makes eclipse to develop spring,spring boot apps very easily ... more over it brings STS IDE features to eclipse IDE.

When we create SpringBoot apps using Spring.io/STS then it would give 3 files

- a. MainClass/ConfigurationClass/Starter class.[class with @SpringBootApplication + main()]
- b. application.properties file[src/main/resources]

c. pom.xml / build.gradle(build file to build configurations)

`@SpringBootApplication`

=>`@EnableAutoConfiguration`(It enables AutoConfiguration)

=>`@ComponentScan`(Scan for the stereo type annotations in the given package and subpackage)

=>`@Configuration`(Marking the class as Configuration class)

=>`@PropertySource`(location="application.properties")

Need of application.properties

Using application.properties with predefined keys we do many configurations like

a.DataSourceConfiguration

b.SpringBoot banner configuration

c.SpringSecurity configuration

d.SpringBatch configuration

e.SpringMail configuration

f.InMemoryDB configuration

g.

PreDefined keys are available in

<https://docs.spring.io/spring-boot/docs/current/reference/html/>

`SpringApplication.run()` internally uses `AnnotationConfigApplicationContext` class to create an IOC container by taking java class

as `@Configuration` class(in fact it takes current class nothing but `ClientApp` cum `ConfigurationClass`)

Note: By default all the components are of Singleton, we can explicitly make it as other scopes using the anotation called

`@Scope(value="")`