SPRING

@Configuration is placed on configuration classes, classes that declare beans

@Bean is used on methods to declare beans

In a Spring application an application context loads bean definitions and wires them together

@Aspect is used on a class to declare it as an aspect

@Pointcut is used on a method to specify the target method

@Before specifies what method gets invoked before the target method – before advice

@After specifies what method gets invoked after the target method – after advice

@EnableAspectJAutoProxy needs to be applied on the configuration class in order for aspects to work

Application context :

Spring comes with several flavors of application context. Here are a few that you’ll

most likely encounter:

AnnotationConfigApplicationContext—Loads a Spring application context

from one or more Java-based configuration classes

AnnotationConfigWebApplicationContext—Loads a Spring web application

context from one or more Java-based configuration classes

ClassPathXmlApplicationContext—Loads a context definition from one or

more XML files located in the classpath, treating context-definition files as classpath

resources

FileSystemXmlApplicationContext—Loads a context definition from one or

more XML files in the filesystem

XmlWebApplicationContext—Loads context definitions from one or more

XML files contained in a web application

http://projects.spring.io/spring-webflow/.

<http://docs.spring.io/spring-ws/site/>

projects.spring.io/spring-security/

<http://projects.spring.io/spring-integration>

[www.manning.com/fisher/](http://www.manning.com/fisher/)

[www.manning.com/templier/](http://www.manning.com/templier/)

<http://projects.spring.io/spring-batch/>

<https://spring.io/guides/gs/accessing-facebook/>

<https://spring.io/guides/gs/accessing-twitter/>

http://projects.spring.io/spring-android/

Page 46

Aspect oriented programming: *cross-cutting concerns*

Dhanji R. Prasanna’s *Dependency Injection*

Wiring :

Three ways of wiring :

* Explicit configuration in XML
* Explicit configuration in Java
* Implicit bean discovery and automatic wiring

Automatic wiring :

Component scanning – spring automatically discovers beans to be created in the application context

Autowiring – spring automatically satisfies bean dependencies

@Component – placed on a class, annotation that identifies the class as a component class and serves as a clue to Spring that a bean should be created for this class, thus eliminating the need to explicitly declare a Spring bean. All beans in a spring application context are given an id, if an id is not explicitly provided spring derives one from the class name

@Component(“somename”)

@Bean(“somename”) / @Bean(name = “somename”)

@Configuration – placed on class, designates a class as a configuration class, bean declarations can be made inside it

@ComponentScan –placed on class, on a configuration class, enables component scanning in Spring. By default it will only scan the same package as the configuration class. It scans the package for classes annotated with @Component and creates a bean. @ComponentScan(“packagename”) or @ComponentScan(basePackages=”packagename”) for explicitly declaring a package, or when you need multiple packages @ComponentScan(basePackages = {“packagename1”, “packagename2”}) , the more typesafeoption is specifying the packages by classes that they contain @ComponentScan(basePackageClasses={Class1.class, Class2.class})

@RunWith(SpringJUnit4ClassRunner.class) – placed on test class, used to have a Spring application context automatically created when the test starts and @ContextConfiguration tells it to load its configuration from a specific configuration class.

Autowiring is a means of letting spring automatically satisfy a bean’s dependencies by finding other beans in the application context that are a match to the bean’s needs.

@Autowired can be placed on constructor, setter methods– indicates that autowiring should be performed

If no match if found an error is thrown, to avoid this: @Autowired(required = false)

@Bean

public CDPlayer cdPlayer() {

return new CDPlayer(sgtPeppers());

}

The cdPlayer() method, like the sgtPeppers() method, is annotated with @Bean to

indicate that it will produce an instance of a bean to be registered in the Spring application

context. The ID of the bean will be cdPlayer, the same as the method’s name.

The body of the cdPlayer() method differs subtly from that of the sgtPeppers()

method. Rather than construct an instance via its default method, the CDPlayer

instance is created by calling its constructor that takes a CompactDisc.

It appears that the CompactDisc is provided by calling sgtPeppers, but that’s not

exactly true. Because the sgtPeppers() method is annotated with @Bean, Spring will

intercept any calls to it and ensure that the bean produced by that method is returned

rather than allowing it to be invoked again.

If the call to sgtPeppers() was treated like any other call to a Java method, then each

CDPlayer would be given its own instance of SgtPeppers. That would make sense if we

were talking about real CD players and compact discs. If you have two CD players,

there’s no physical way for a single compact disc to simultaneously be inserted into

two CD players.

In software, however, there’s no reason you couldn’t inject the same instance of

SgtPeppers into as many other beans as you want. By default, all beans in Spring are

singletons, and there’s no reason you need to create a duplicate instance for the second

CDPlayer bean. So Spring intercepts the call to sgtPeppers() and makes sure

that what is returned is the Spring bean that was created when Spring itself called

sgtPeppers() to create the CompactDisc bean. Therefore, both CDPlayer beans will

be given the same instance of SgtPeppers.

In any event, it’s important to recognize that although you’re performing DI via

the CDPlayer’s constructor, there’s no reason you couldn’t apply other styles of DI

here. For example, if you wanted to inject a CompactDisc via a setter method, it might

look like this:

@Bean

public CDPlayer cdPlayer(CompactDisc compactDisc) {

CDPlayer cdPlayer = new CDPlayer(compactDisc);

cdPlayer.setCompactDisc(compactDisc);

return cdPlayer;

}