Fun Annotation Notes

@ComponentScan + @Component

During Spring initialization the class that is marked with @ComponentScan will have it's package and it's subpackage scanned for any class that's marked with @Component. If any @Component class is found then it will add them to the application context (Basically contains the beans that will be managed by Spring).

Spring resolves dependencies between Spring beans and injects beans into each other's fields or constructor in order to satisfy the dependency.

@Component and @Bean

@Component is applied to an entire class marking it as an bean for the Spring to manage. i.e. it will do dependency injection on all of the dependency, and then where-ever this class marked as @Component is needed it will be injected.

@Bean is applied to a method. The method will return the dependency class object you specified, with the specified constructor configuration. When the corresponding bean is used, by the type or the name of the @Bean method explicitly, it will retrieve the dependency and use it. The calling of the method and storing it as a bean is done at the initialization step.

If you use @Bean, the name of the method will become the name of the bean.

You should use @Bean when you don't have access to 3rd-party library source code, but you would like to autowire component from it.

You can inject @Bean into @Component variables, and you can also inject @Component into @Bean in the parameter. Inter-injection is also allowed (which should be obvious, you have @Component that uses other @Component dependencies).

> If you have multiple bean either from Component or from @Bean methods, you might run into errors where Spring doesn't know which Bean to use, you can distinguish between each of them using the @Qualifier annotation to mark the bean that you would like to name and then use in @Autowired

@Autowired / @Inject

@Autowired and @Inject can be used interchangeably. Spring created @Autowired, @Inject is part of Java but Spring supports it fully.

This is the annotation that is used to actually inject the dependency into the class. There are many styles of doing the injection, you can do:

- Field injection: Putting @Autowired on the private field of the class will inject that field for you
- Setter injection: Putting @Autowired on the setter for that particular field that you are injecting of the class
- Constructor injection: Putting @Autowired on the class's constructor for dependency injection

> For Spring 4.3+, if a class that is configured as a Spring bean or component, only have one constructor and constructor have the wanted auto injected field in the parameter, then you can skip out on the @Autowired annotation.

Before Spring 4.3:

```
@Component
public class ExampleDB {
□
□private ExampleRecord record;

    @Autowired
    public ExampleDB(ExampleRecord record) {
    □this.record = record;
    }
}
```

After:

```
@Component
public class ExampleDB {
□
□private ExampleRecord record;

    public ExampleDB(ExampleRecord record) {
    □this.record = record;
    }
}
```

As you can see this is more leegant, BUT, at the cost of context. If you don't specify it explicitly the programmer will not know at immediately that record is an injected dependency.

See all Spring Managed Beans

```
public void run(String... args) throws Exception {
    String[] beans = ctx.getBeanDefinitionNames();
        Arrays.sort(beans);
        for (String bean : beans) {
            System.out.println(bean);
        }
}
```

This will print out all of the registered beans.

For @Components without name arguments, the bean name will be the class name but camelcased. ExampleDB will be exampleDB for bean name.

For @Beans it is the same case as well, the return type will be the bean name but camelcased. Unless you specify the name argument for the @Bean annotation.

When you @Autowired, if there are multiple @Bean of the same type, you can disambiguate by using the @Qualifier annotation to specify exactly which bean you would like to use according to the name.

If you have multiple bean with the same bean name, it will be an error and you cannot run your application.

@Autowired, @Resource, @Inject

They all do the same stuff, @Autowired is by Spring. Other two is by Java.

https://www.baeldung.com/spring-annotations-resource-inject-autowire

@Autowiring an interface

If you have multiple implementation of an interface and are all marked as beans. To specify which implementation to @Autowire you would need to add the @Qualifier to specify exactly which bean it is to use.

Funny enough in the latest Spring framework, I observe that if the variable name matches one of the bean name then it will @Autowire to that bean without the need to specify the bean name.

Revision #9
Created 2023-05-16 23:25:10 UTC by Tamarine
Updated 2023-06-03 22:18:43 UTC by Tamarine