


@Bean vs @Component

Theory

Practice

 100% completed, 1 problem solved

▼

Theory

🕒 12 minutes reading

Unskip this topic

Start practicing

You've learned by now that Spring provides different ways to create container-managed beans. One of them is by using the `@Bean` annotation and another is applying the `@Component` annotation. Both of them provide special objects that can be initialized at startup and automatically wired across different parts of an application. However, it might sometimes be unclear how to use them together and which of them to choose in certain cases.

In this topic, you will learn about the key differences between the `@Bean` and `@Component` annotations and how to use them together.

§1. Using @Bean and @Component together

In practice, beans created manually by annotating a method with `@Bean` can be injected into components using the `@Autowired` annotation and it's quite a canonical way of using them. In this case, `@Bean` is used to create different application- or component-wide configs and autowire them when needed.

To see how it works, let's take a look at an application that generates random passwords. You've already seen it in previous topics. We are going to discuss three key parts of this application:



- a configuration bean that is used to define a set of characters for creating passwords;
- a component that requires this configuration bean and performs some calculations;
- a component that is used to interact with the IO system.

Here is `PasswordConfig` that produces the bean containing the information about the alphabet we'd like to use:

▼ Java


```
1  @Configuration
2  public class PasswordConfig {
3      private static final String ALPHA =
"abcdefghijklmnopqrstuvwxyz";
4      private static final String NUMERIC = "0123456789";
5      private static final String SPECIAL_CHARS = "!@#$%^&* _+=- /";
6
7      @Bean
8      public PasswordAlphabet allCharacters() {
9          return new PasswordAlphabet(ALPHA + NUMERIC +
SPECIAL_CHARS);
10     }
11
12     static class PasswordAlphabet {
13
14         private final String characters;
```

1 required topic



  [Spring components](#)

▼

4 dependent topics

  [ApplicationContext](#)

▼

  [Scopes of beans](#)

▼

 [CRUD repositories](#)

▼

[Async methods](#)

▼

```

1
5     public PasswordAlphabet(String characters) {
1
6         this.characters = characters;
1
7     }
1
8
1
9     public String getCharacters() {
2
10        return characters;
2
11    }
2
12    }
2
13    }
3

```

▼ Kotlin

```

1  @Configuration
2  class PasswordConfig {
3      companion object {
4          private const val ALPHA = "abcdefghijklmnopqrstuvwxyz"
5          private const val NUMERIC = "0123456789"
6          private const val SPECIAL_CHARS = "!@#$%^&* _+=-/"
7      }
8
9      @Bean
1
10     fun allCharacters(): PasswordAlphabet {
1
11         return PasswordAlphabet(ALPHA + NUMERIC + SPECIAL_CHARS)
1
12     }
1
13
14     class PasswordAlphabet(val characters: String)
1
15 }

```

Below is a new version of the `PasswordGenerator` component that applies this config to the password generation process.

▼ Java

```

1  @Component
2  public class PasswordGenerator {
3      private static final Random random = new Random();
4      private final PasswordAlphabet alphabet;
5
6      public PasswordGenerator(@Autowired PasswordAlphabet alphabet) {
7          this.alphabet = alphabet;
8      }
9
10
11     public String generate(int length) {
1
12         String allCharacters = alphabet.getCharacters(); // get the
characters from the bean
1
13         StringBuilder result = new StringBuilder();
1
14         for (int i = 0; i < length; i++) {
1
15             int index = random.nextInt(allCharacters.length());
1
16             result.append(allCharacters.charAt(index));
17         }
18     }
19 }

```

```
1
6      }
1
7      return result.toString();
1
8  }
1
9  }
```

▼ Kotlin

```
1  @Component
2  class PasswordGenerator(@Autowired private val alphabet:
PasswordAlphabet) {
3      companion object {
4          private val random = Random()
5      }
6
7      fun generate(length: Int): String {
8          val allCharacters = alphabet.characters // get the
characters from the bean
9          val result = StringBuilder()
1         for (i in 0 until length) {
1
1             val index: Int = random.nextInt(allCharacters.length)
1
1             result.append(allCharacters[index])
3         }
1
4         return result.toString()
1
5     }
1
6 }
1
7
```

And, last but not least, here is the component that is responsible for interacting with IO:

▼ Java

```
1  @Component
2  public class Runner implements CommandLineRunner {
3      private final PasswordGenerator generator;
4
5      public Runner(PasswordGenerator generator) {
6          this.generator = generator;
7      }
8
9      @Override
1     public void run(String... args) {
1
1         System.out.println("A short password: " +
generator.generate(5));
1
2         System.out.println("A long password: " +
generator.generate(10));
1
3     }
1
4 }
```

▼ Kotlin

```
1  @Component
2  class Runner(private val generator: PasswordGenerator) :
CommandLineRunner {
3      override fun run(vararg args: String) {
4          println("A short password: " + generator.generate(5))
5          println("A long password: " + generator.generate(10))
6      }
7  }
8
```

If you run this application again, you will most likely get passwords containing digits and special characters.

```
A short password: e&7sd
A long password: up_&g4xtj7
```

Besides, it is possible to declare several alphabet config beans and, depending on our intentions, inject them using the `@Qualifier` annotation. It will allow you to make this application much more customizable for different use cases.

Usually, components depend on other components or objects produced by `@Bean`-annotated methods. However, technically, you can also autowire a component into the parameters of a method.

§2. @Component vs @Bean

So far, we have used both `@Bean` and `@Component` annotations to create beans that can be injected into each other. Now let's look at the differences between these annotations.

- The `@Bean` annotation is a method-level annotation and `@Component` is a class level annotation.
- The `@Component` annotation doesn't need to be used with the `@Configuration` annotation whereas the `@Bean` annotation has to be used within the class which is annotated with `@Configuration`.
- If you want to create a bean for a class from an external library, you cannot just add the `@Component` annotation because you cannot edit the class. However, you can declare a method annotated with `@Bean` and return an object of this class from this method.
- There are several specializations of the `@Component` annotation, whereas `@Bean` doesn't have them.

In most cases, you can use both approaches but Spring developers typically prefer `@Component` whenever possible. The `@Bean` annotation is mostly used for producing beans of unmodifiable classes or creating some configs.

§3. Specializations of components

As we mentioned before, there are several specializations of components depending on their role in Spring applications:

- `@Component` indicates a generic Spring-managed component;
- `@Service` indicates a business logic component but doesn't provide any additional functions;
- `@Controller` / `@RestController` indicates a component that can work in a web environment;
- `@Repository` indicates a component that interacts with an external data storage (e.g. a database).

In the following topics, you will learn more about different types of Spring components and the way they are typically used in applications. For now, you

Table of contents:

- [1 @Bean vs @Component](#)
- [§1. Using @Bean and @Component together](#)
- [§2. @Component vs @Bean](#)
- [§3. Specializations of components](#)
- [§4. Conclusion](#)
- [Discussion](#)

can just remember: if your component doesn't need to communicate with a database or return an HTTP result, you can use just the `@Service` annotation whenever using the `@Component` annotation is possible. Overall, as long as you write simple programs, there should be no difference.

§4. Conclusion

Now you've learned about the differences between the `@Bean` and `@Component` annotations and when to use them for developing Spring applications. We discussed an example with two components and one `@Bean`-annotated method and saw how these two concepts are used together to make your application more customizable. We also introduced several component specializations, and you are going to further explore and gradually start using them in future topics.

 Report a typo

132 users liked this piece of theory. 4 didn't like it. **What about you?**



Start practicing

Unskip this topic

[Comments \(6\)](#)

[Useful links \(3\)](#)

[Show discussion](#)

 JetBrains Academy

Tracks

Pricing

For organizations

About

Contribute

Careers

 Become beta tester

Be the first to see what's new

Terms Support  

Made with  by Hyperskill and JetBrains