Spring, Spring Boot, and Component Scan

Component scanning can make your life easier in Spring, and Spring Boot brings an additional level of power. In this post, we take a closer look at component scanning.

28 Minions by Ranga Karanam 具体的 CORE · Jan. 02, 18 · Java Zone · Tutorial

This guide will help you understand the most important concept in Spring — Component Scan. Spring Boot does some magic around Component Scan. Let's understand that in this article.

You Will Learn

- What is Component Scan?
- Why is Component Scan important?
- Which packages does Spring Boot scan automatically?
- How do you define Component Scan with Spring Boot?
- How do you resolve problems involving Component Scan?

@ComponentScan

If you understand Component Scan, you understand Spring.

Spring is a dependency injection framework. It is all about beans and wiring in dependencies.

The first step of defining Spring Beans is by adding the right annotation — @Component or @Service or @Repository.

However, Spring does not know about the bean unless it knows where to search for it.

> **This part of "telling Spring where to search" is called a Component Scan.**

You define the packages that have to be scanned.

Once you define a Component Scan for a package, Spring would search the package and all its sub packages for components/beans.

Defining a Component Scan

- If you are using Spring Boot, check the configuration in Approach 1.
- If you are doing a JSP/Servlet or a Spring MVC application without using Spring Boot, use Approach 2.

Approach 1: Component Scan in a Spring Boot Project

- If your other package hierarchies are below your main app with the @SpringBootApplication annotation, you're covered by the implicit Component Scan.
- If there are beans/components in other packages that are not sub-packages of the main package, you should manually add them as @ComponentScan

Consider the class below:

```
package com.in28minutes.springboot.basics.springbootin10steps;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication
public class SpringbootIn10StepsApplication {

	public static void main(String[] args) {
```

`@SpringBootApplication` is defined in the `SpringbootIn10StepsApplication class` which is in the package `com.in28minutes.springboot.basics.springbootin10steps`

`@SpringBootApplication` defines an automatic Component Scan on the package `com.in28minutes.springboot.basics.springbootin10steps`.

You are fine if all your components are defined in the above package or a sub-package of it.

However, let's say one of the components is defined in package `com.in28minutes.springboot.somethingelse`

In this case, you would need to add the new package into Component Scan.

You have two options:

- Define @ComponentScan("com.in28minutes.springboot")
  - This would scan the entire parent tree of com.in28minutes.springboot.
- Or define two specific Component Scans by using an array.
  - @ComponentScan({"com.in28minutes.springboot.basics.springbootin10steps","com.in28minutes.springboot.somethingelse"})

Option 1:

```
@ComponentScan("com.in28minutes.springboot")
@SpringBootApplication
public class SpringbootIn10StepsApplication {
```

Option 2:

```
@ComponentScan({"com.in28minutes.springboot.basics.springbootin10steps","com.in28minutes.springboot.somethingelse"})
@SpringBootApplication
public class SpringbootIn10StepsApplication {
```

Approach 2: Non-Spring Boot Project

In a non-Spring Boot Project, we would typically define the component scan explicitly in an XML application context or a Java Application Context.

Option 1:

```
@ComponentScan("com.in28minutes")
@Configuration
public class SpringConfiguration {
```

Option 2:

```
@ComponentScan({"com.in28minutes.package1","com.in28minutes.package2"})
@Configuration
public class SpringConfiguration {
```

XML application context:

```
<context:component-scan base-package="com.in28minutes" />
```

Specific multiple packages:

```
<context:component-scan base-package="com.in28minutes.package1, com.in28minutes.package2" />
```

URL Not Working

The server starts up fine, but:

- My URL is not working
- My login URL is not working
- My todo URL is not working ``` WARNING: No mapping found for HTTP request with URI [/spring-mvc/login] in DispatcherServlet with name 'dispatcher' WARNING: No mapping found for HTTP request with URI [/login] in DispatcherServlet with name 'dispatcher' WARNING: No mapping found for HTTP request with URI [/list-todos] in DispatcherServlet with name 'dispatcher'

```
### No qualifying bean of type found
```

No qualifying bean of type [com.in28minutes.springboot.jpa.UserRepository] found for dependency [com.in28minutes.springboot.jpa.UserRepository]: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}
```

> **Same root cause for both above problems — the component is not being picked up.**

There are three possible things you would need to look at:

- You have not added the right annotation — @Controller, @Repository, or @Controller
- You have not added a Component Scan.
- The package of your component is not defined in Component Scan.

You have two options:

1. Add the annotation or component scan
2. Move the component to a package already under Component Scan

## @Component vs. @ComponentScan

@Component and @ComponentScan are for different purposes.

- @Component indicates that a class might be a candidate for creating a bean. It's like putting a hand up.
- @ComponentScan is searching packages for Components. Trying to find out who all put their hands up.

Topics: JAVA, COMPONENT SCANNING, SPRING BOOT, TUTORIAL

Published at DZone with permission of Ranga Karanam, DZone MVB. See the original article here.
Opinions expressed by DZone contributors are their own.
```