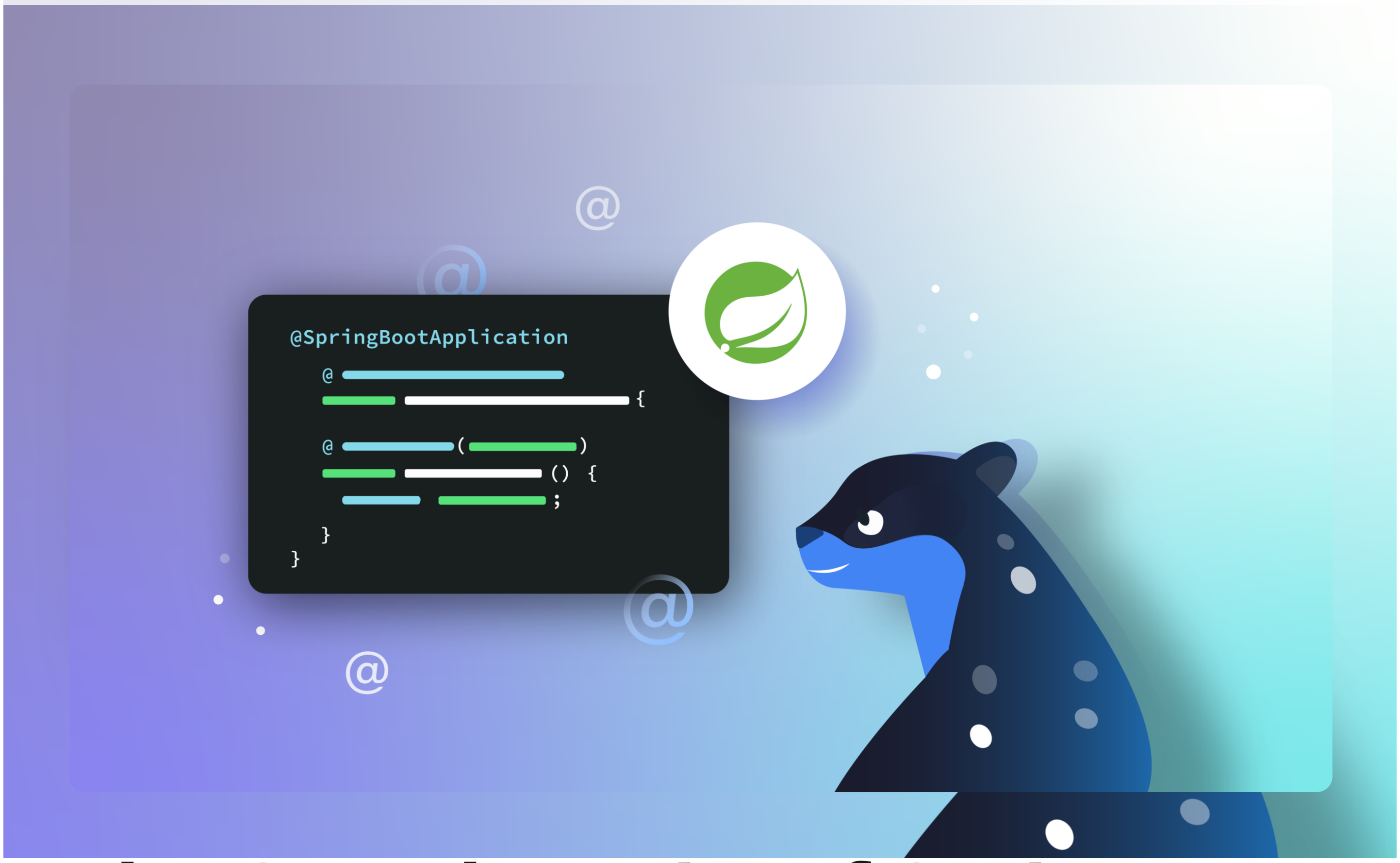
May we use cookies to track your activities? We take your privacy very seriously. Please see our privacy policy for details and any questions.Yes No



The Complete List of Spring Boot Annotations You Must Know

James Konik

21-Sep-2021

9 min read

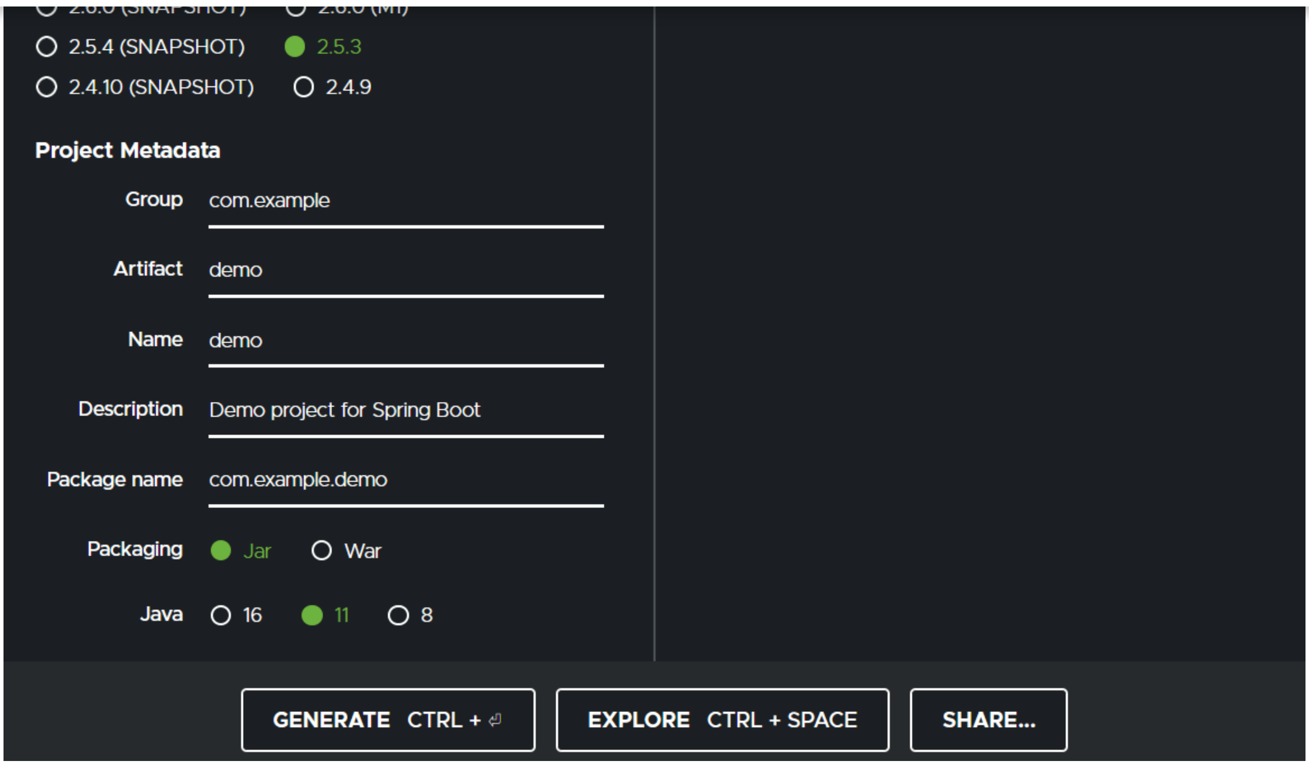
Java

Answer these 3 questions to know if Developer Observability is for you!

Java engineers working in the spring framework can deploy web applications quickly, without the need for a separate server. Spring focuses on speed, simplicity, and productivity. It uses Inversion of Control, or IoC, and dependency injection to increase modularity and enable loose coupling.

Spring Boot builds off of the Spring framework. It's **an *opinionated* framework, making decisions for you and minimizing the amount of configuration and boilerplate code you need to get started**.

Spring applications need a fair amount of configuration. Things like Maven dependencies, MVC configuration, and security all require a lot of code to get up and running. Spring reduces or eliminates these needs entirely, instead allowing you to override any of the defaults it selects for you.



Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	[✓ Jar] War
Java	16 [✓ 11] 8

Convinced? It's easy to try out. Spring's website has an easy quickstart guide, and lets you create a test application with a few clicks. You can use it with different languages and spring versions and add whatever dependencies you need.

Spring Boot Annotations

Spring Boot's annotations are key to getting up to speed with the framework. They allow you to direct the framework to do your bidding, taking control and overriding its defaults when needed. Annotations are quick, easy to use, and orders of

You can learn more in Spring's documentation. The PDF or web versions are essential reading if you plan on working with Spring Boot.

Knowing how to use annotations can level up your game and help you get the most out of the framework. Listed below are several common annotations, some of them with sample code. Spring Boot works with Java, Kotlin, and Groovy. I'll use Java for the examples in this article.

- **Basic Setup**

  - `@SpringBootApplication`
  - `@Configuration`
  - `@ComponentScan`
  - `@EnableAutoConfiguration`

- `@RequestMapping`
- `@RequestParam`

- **Component Types**
  - `@Component`
  - `@Service`
  - `@Repository`
  - `@Controller`
  - `@RestController`
- **Testing**

    - `@MockBean`
    - `@Validated`
- **Misc**
    - `@Bean`
    - `@ConditionalOnJava`

List of Essential Spring Boot Annotations

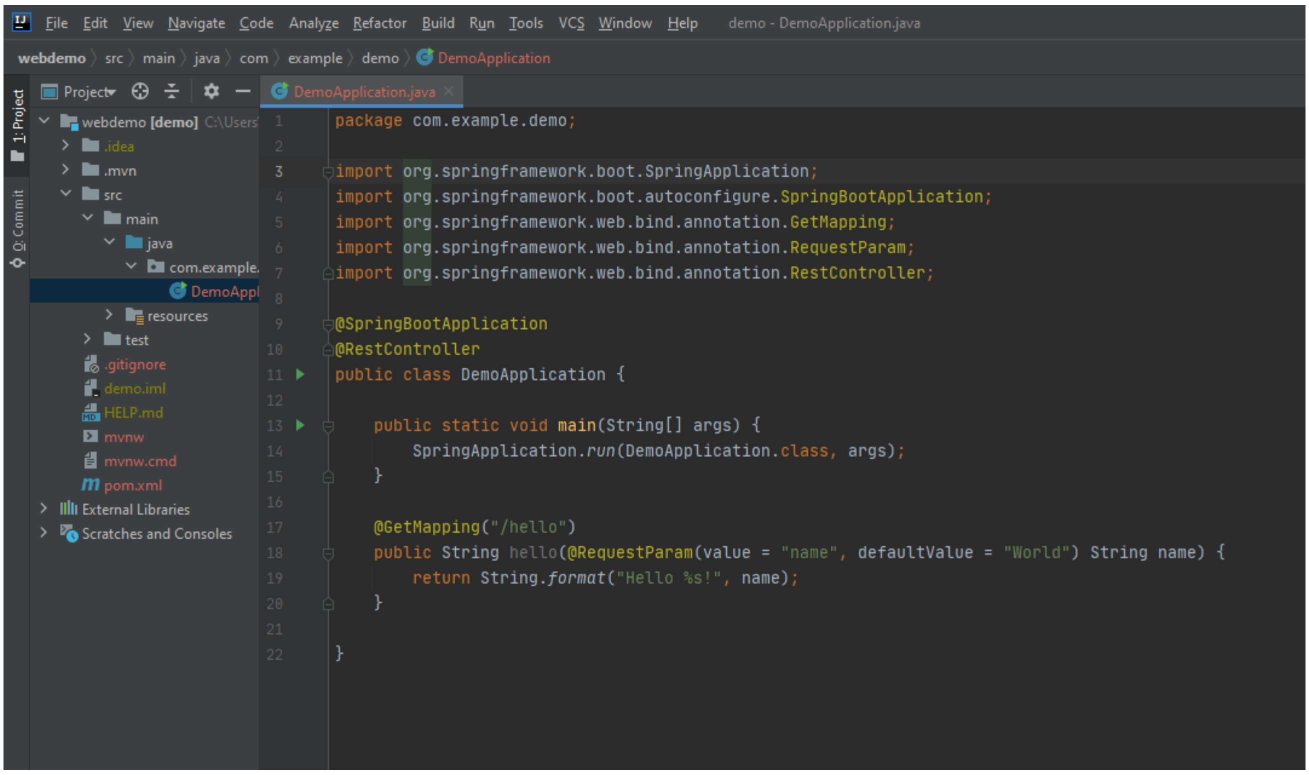
There are many annotations you can use to control and define your applications. Here are some of the most useful, sorted by category.

1. Basic Setup

`@SpringBootApplication`

Spring Boot applications.

The 1.2 version delivers the same functionality as the near ubiquitous @Configuration, @ComponentScan, and @EnableAutoConfiguration.



```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
        return String.format("Hello %s!", name);
    }

}
```

This is a good place for a simple `Hello World`, based on Spring Boot's demo application, so here goes:

```
package com.example.demo;
```

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args)
     SpringApplication.run(DemoApplication
    }

}
```

It's not much of a `Hello World` though, as it doesn't actually display anything. For that, let's introduce @RestController and @GetMapping. But, before that, let's talk about the annotations that @SpringBootApplication replaces.

```
@Configuration
```

Now superseded by @SpringBootApplication, @Configuration enables Java configuration and lets you use Spring Beans in the class.

Also superseded by @SpringBootApplication, @ComponentScan enables component scanning and means controller classes and components you create can be discovered by the framework. It marks classes to be discovered with @Controller.

If you include the @ComponentScan annotation, then all application components will be registered as Spring Beans automatically. That includes @Service, @Component, @Repository, @Controller, and others.

```
@EnableAutoConfiguration
```

The final annotation replaced by @SpringBootApplication, @EnableAutoConfiguration enables Spring Boot's autoconfiguration. Spring Boot makes over 200 decisions for you. These can be overridden if you want to make your own choices, but it will pick sensible defaults for you, saving you a lot of time at the beginning of projects.

2. Request Responses

The next few annotations show how easy it is to respond to HTTP requests using Spring. There are several ways to mark classes and functions to control how they behave and what they return.

```
@GetMapping
```

Take a look at your earlier `Hello World` and get it returning a response.

In this example, again from Spring's demo code, the @GetMapping class combines with the @RestController to deliver a response to calls like `http://yoururl/hello`.

Here it's a simple `Hello World` by default, though it also shows how to take a parameter, which can replace the default *World* in the response. `http://yoururl/?name=Dave` will return `Hello Dave`, for example.

```
import org.springframework.boot.SpringAppl
import org.springframework.boot.autoconfig
import org.springframework.web.bind.annota
import org.springframework.web.bind.annota
import org.springframework.web.bind.annota

@SpringBootApplication
@RestController
public class DemoApplication {

public static void main(String[] args) {
SpringApplication.run(DemoApplication.clas
}

@GetMapping("/hello")
public String hello(@RequestParam(value = 
return String.format("Hello %s!", name);
}
}
```

@RequestMapping

```
@Controller
@RequestMapping("users")
public class UserController {

    @GetMapping("/{id}", produces = "appli
    public @ResponseBody User getUser(@Path
        return findUsersById(id);
    }

    private User findUsersById(int id) {
            // return user specific data
    }
```

```
@RequestParam
```

As seen in the code above, @RequestParam allows you to send parameters in the get request and use them in Java. It also supplies a default value.

3. Component Types

but allow you to organize your application and mark classes for specific roles, helping to keep your application modular.

`@Component`

Application components and their variants are automatically registered as Spring Beans, providing dependency injection, provided you use either @SpringBootApplication or @ComponentScan.

@Repositry, @Controller, and @Service are move specific alternatives to @Componenet.

`@Service`

This is an alternative to @Component that specifies you intend to use the class as part of your service layer. However, it doesn't actually implement anything differently than @Component.

`@Repository`

tests and generating the right exceptions.

```
@Controller
```

@Controller is a specialized @Component marked as a controller in MVC architecture.

```
@RestController
```

@RestController combines the @Controller and @ResponseBody into a single annotation. @RestController classes return domains instead of views.

4. Testing

As it should, Spring Boot makes it very easy to write tests. Identifying and fixing errors is much easier when the framework helps you.

```
@SpringBootTest
```

using Spring Boot specific features.

@SpringBootTest works best when testing the whole application together. There are other options, such as @WebMvcTest and @DataJpaTest to use, if you're looking at those specific areas.

You can mark a test class as follows:

```
@SpringBootTest(properties = "spring.main.
class YourTests {
    // code here
}
```

```
@MockBean
```

The @MockBean annotation allows you to create a temporary version of a service for testing. It's useful if you have a web service you connect to that isn't suitable for testing, or if you want to test against specific results.

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.ar
import org.springframework.boot.test.conte
import org.springframework.boot.test.mock.r

import static org.assertj.core.api.Asserti
import static org.mockito.BDDMockito.*;

@SpringBootTest
class ServiceTests {

    @MockBean
    private RemoteService remoteService;

    @Autowired
    private Capitalizer capitalizer;

    @Test
    void exampleTest() {
        // RemoteService has been injected
        given(this.remoteService.testCall(
        String caps = capitalizer.capitali
}
```

```
@Validated
```

To validate input for methods, you apply the @Validated annotation to the class. For nested properties, apply the @Valid tag.

Here's an example of using the @Validated tag along with javax.validation constraints.

```
import javax.validation.constraints.Size;
import javax.validation.constraints.NotNul
import org.springframework.stereotype.Serv
import org.springframework.validation.anno

@Service
@Validated
public class TestBean {
public Archive findByCopiesAndTitle(@Size(
return …
```

Miscellaneous

```
@Bean
```

Dependency injection is central to Spring Boot, and the @Bean annotation allows you to mark tightly coupled classes. That tells the Spring container to handle their lifecycle.

It’s useful if you have a class that requires an instance of another class as a property, or in other similar scenarios. The Spring IoC container makes sure all dependencies are satisfied. When the bean is created, it makes sure everything is instantiated in the right order, and nothing is left out.

```
@ConditionalOnJava
```

If you want code to run only on a particular Java version, this annotation is very handy.

```
Class UserAlertService {
    // strongly suggest a Java update
    ...
}
```

There are several other conditional annotations, too.

@ConditionalOnProperty, @ConditionalOnWebApplication, @ConditionalOnClass, and many others can be used and combined as needed. Check them out if you want to offer variable functionality, but be wary of relying on this kind of thing too much. It’s all too easy to get tangled up.

Conclusion

Spring Boot makes life faster and easier. It provides automatic configuration and gives you a ready-made starting point for your application, with many built in features, saving you the need to write boilerplate code.

Knowing how to use its annotations helps you take full advantage of everything Spring Boot offers. Annotations provide complicated functionality for almost no coding time, so the impact on your development speed can be huge.

If you're new to Spring Boot, learning its annotations is a no-brainer. Don't stop here though. You can learn more about the annotations covered here, as well as others available in the Spring Boot documentation. Spring Boot also has

For more help building web applications, create a free Lightrun account. It can help Java, Python, and Node developers log and debug applications in real time, delivering metrics on running services. If you're working in Spring Boot, Lightrun can help you get feedback on how well your app runs, giving you another way to improve your code.

Share

5

Subscribe to our Newsletter

and get the latest news, articles and resources sent to your inbox.

E-mail*

Related Posts

Debugging Live Java Applications with Lightrun

How to Debug Race Conditions Between Threads in Java

It’s Really not that Complicated.

You **can** actually understand what’s going on inside your live applications. It’s a registration form away.

Get Lightrun

Subscribe to our Newsletter

E-mail*

Subscribe

Product

Open a Free Account

Request a Demo

Debug WORDLE with us

IDEs & Runtimes

Production Debugging

Kubernetes Debugging

Serverless Debugging

Feature Flag Debugging

Resources

Dev Observability

How it Works

Security & Compliance

Documentation

Blog

Case Studies

Webinars

White Papers

Company

Contact Us

Support

Press

Terms of Use

Services Terms & Conditions

Privacy Policy

© 2022, Lightrun, Inc. All Rights Reserved.