



PERFORCE

(<https://www.perforce.com>)

[Blog \(/blog\)](#)

[Request a Quote](#) ▼

[Downloads](#) ▼

[Company](#) ▼

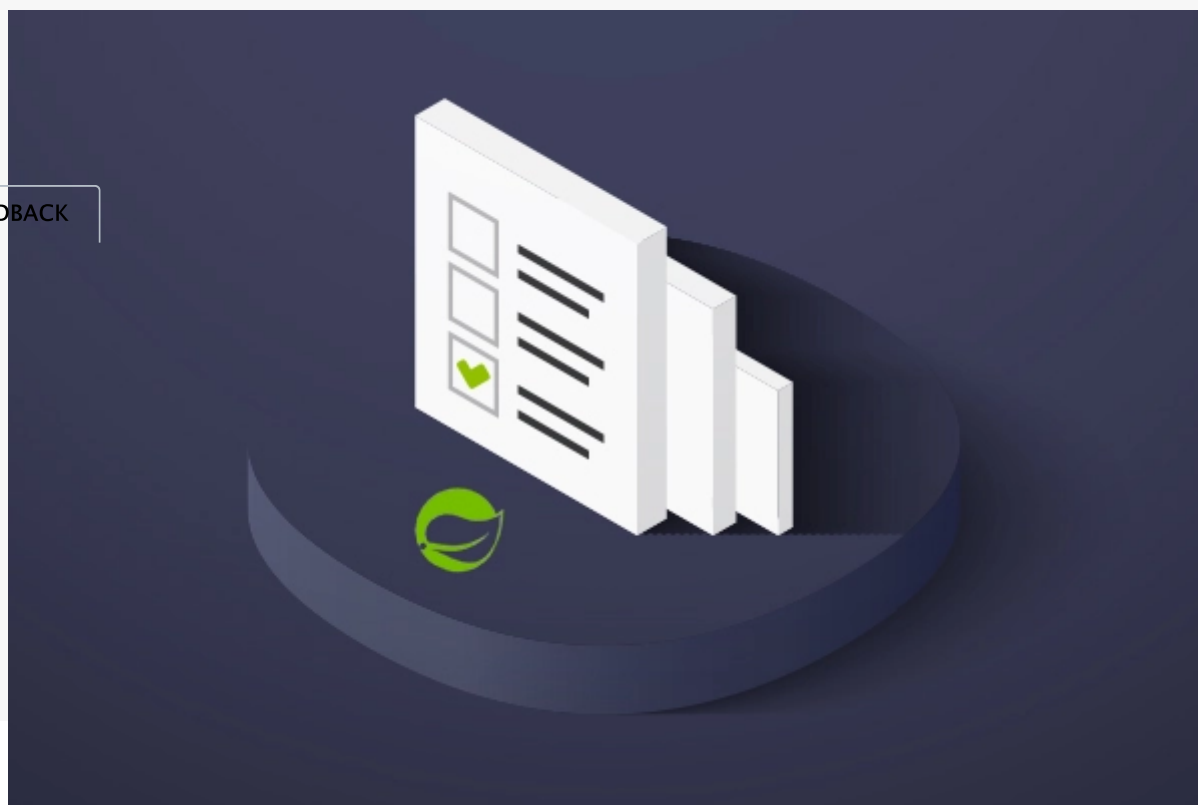
[Contact](#) ▼

Don't Let Remote Work Slow You Down — [Skip Redeploys With JRebel >> \(/products/jrebel/free-trial\)](#)

**JRebel**

(/).

SEND FEEDBACK



October 26, 2016

# Spring Annotations Cheat Sheet

JAVA FRAMEWORKS | DEVELOPER PRODUCTIVITY

In our post today, we're talking about Spring Framework Annotations. We've gathered up all the commonly-used and useful annotations developers will use and packed them into a one-page Spring annotations cheat sheet. It covers everything from the basic annotations you need to get your project started, to service discovery annotations.

You can grab a copy of the cheat sheet at the bottom of the page, or you can skip to a relevant section to grab an annotation to go.

- [Spring Annotations](#)
- [Spring Boot Annotations](#)
- [Spring Boot Web Annotations](#)
- [Spring Boot Cloud Annotations](#)
- [Spring Discovery Service Annotations](#)
- [Spring Annotations Cheat Sheet](#)

## Important Spring Annotations

Spring uses dependency injection to configure and to bring your application together. It means that you just declare the components of your system and how they interact. Then Spring wires it all together at runtime. Here are the most important annotations you should know of.

- **@Configuration** - annotation is used to mark a class as a source of the bean definitions. Beans are the components of the system that you want to wire together. A method marked with the **@Bean** annotation is a bean producer. Spring will handle the life cycle of the beans for you, and it will use these methods to create the beans.
- **@ComponentScan** - use to make sure that Spring knows about your configuration classes and can initialize the beans correctly. It makes Spring scan the packages configured with it for the **@Configuration** classes.
- **@Import** - If you need even more precise control of the configuration classes, you can always use **@import** to load additional configuration. This one works even when you specify the beans in an XML file like it's 1999.
- **@Component** - Another way to declare a bean is to mark a class with a **@Component** annotation. Doing this turns the class into a Spring bean at the auto-scan time.
- **@Service** - Mark a specialization of a **@Component**. It tells Spring that it's safe to manage them with more freedom than regular components. Remember, services have no encapsulated state.
- **@Autowired** - To wire the application parts together, use the **@Autowired** on the fields, constructors, or methods in a component. Spring's dependency injection mechanism wires appropriate beans into the class members marked with **@Autowired**.

## More Complex Spring Annotations

Now it's time to go a bit deeper. The bean initialization is eager by default. Spring will try to initialize all the beans and wire them all together. You can mark a `@Bean` or `@Component` with `@Lazy` to have them be initialized on demand. It can save some startup time!

If you have multiple beans that can be wired into the field marked with `@Autowired`, use `@Qualifier` to filter which beans should be used there.

Another useful annotation is `@Value`. This one indicates a default value expression for the field or parameter to initialize the property with. Typically something like the following expression, referencing a configuration property will be used there: `"#{systemProperties.myProp}"`.

And to make the system more robust, you can enable the verification to make Spring fail when a bean to autowire is not found. Use `@Required` to fail the wiring if the dependency cannot be injected.

- `@Lazy` - Initialize bean on demand
- `@Value` - indicate default value expression for field or parameter
- `@Required` - Fail wiring if dependency can't be injected

Armed with these annotations you can make the application come together with a very little effort. Naturally, there are more Spring annotations that you might want to use, but these here are the core of the framework that enables the flexibility Spring is known for!

## Important Spring Boot Annotations

Let's look at some of the most frequently used annotations in the context of web apps. Most of our readers are either backend engineers or are doing full stack developer jobs. So it makes sense to popularize the Spring Framework annotations that make web development easier.

### `@SpringBootApplication`

First one of the most basic, super helpful annotations, our all time favorite `@SpringBootApplication`. There's nothing magical about it. It's syntactic sugar for combining other annotations that we'll look at in just a moment. `@SpringBootApplication`

is **@Configuration**, **@EnableAutoConfiguration** and **@ComponentScan** annotations combined, configured with their default attributes.

## **@Configuration and @ComponentScan**

The **@Configuration** and **@ComponentScan** annotations that we described above make Spring create and configure the beans and components of your application. It's a great way to decouple the actual business logic code from wiring the app together.

## **@EnableAutoConfiguration**

Now the **@EnableAutoConfiguration** annotation is even better. It makes Spring guess the configuration based on the JAR files available on the classpath. It can figure out what libraries you use and preconfigure their components without you lifting a finger. It is how all the spring-boot-starter libraries work. Meaning it's a major lifesaver both when you're just starting to work with a library as well as when you know and trust the default config to be reasonable.

# **Important Spring Boot Web Annotations**

The following annotations make Spring configure your app to be a web application, capable of serving the HTTP response.

## **@Controller**

**@Controller** marks the class as a web controller, capable of handling the HTTP requests. Spring will look at the methods of the class marked with the **@Controller** annotation and establish the routing table to know which methods serve which endpoints.

## **@ResponseBody**

The **@ResponseBody** is a utility annotation that makes Spring bind a method's return value to the HTTP response body. When building a JSON endpoint, this is an amazing way to magically convert your objects into JSON for easier consumption.

## **@RestController**

Then there's the **@RestController** annotation -- a convenience syntax for **@Controller** and **@ResponseBody** together. This means that all the action methods in the marked class will return the JSON response.

## **@RequestMapping(method = RequestMethod.GET, value = "/path")**

The **@RequestMapping(method = RequestMethod.GET, value = "/path")** annotation specifies a method in the controller that should be responsible for serving the HTTP request to the given path. Spring will work the implementation details of how it's done. You simply specify the path value on the annotation and Spring will route the requests into the correct action methods.

## **@RequestParam(value="name", defaultValue="World")**

Naturally, the methods handling the requests might take parameters. To help you with binding the HTTP parameters into the action method arguments, you can use the **@RequestParam(value="name", defaultValue="World")** annotation. Spring will parse the request parameters and put the appropriate ones into your method arguments.

## **@PathVariable("placeholderName")**

Another common way to provide information to the backend is to encode it in the URL. Then you can use the **@PathVariable("placeholderName")** annotation to bring the values from the URL to the method arguments.

# **Most Useful Spring Cloud Annotations**

Now you have a fully functioning web app that is both neat on the code level as well as separated into beans and service classes. The app is declaratively configuring background jobs and gets constructed and configured at runtime without extra uproar.

If you're in the mood to build a larger system of such web apps, you'll need to work more on the organizational problems. These include issues like how to configure these multiple apps, how to make them communicate with each other, and how to make sure that network failures will not crash the whole system.

The Spring Cloud project (<http://projects.spring.io/spring-cloud/>) tries to precisely answer these questions and help you take your apps to the cloud.

## **@EnableConfigServer**

**@EnableConfigServer** turns your application into a server other apps can get their configuration from. You need a configuration server to have a central repository of all the appropriate configurations. The good thing is that creating one is just an annotation away.

Now when you have the central configuration server available, you can use the `spring.application.cloud.config.uri` property in the client Spring apps to point them to the configuration server. When these apps start, they will reach out to the configuration server and obtain the necessary properties. You don't have to package your configuration together with the app code. This separation of concerns works wonders to simplify your deployments.

---

## Spring Discovery Service Annotations

The next bit we want to cover is even cooler. Imagine a situation when you have many small services that are context bounded, excel at one thing and put together, form a complex system that brings your real business value. How do you wire the services together? You cannot go with hardcoding the URLs in code or even the configuration -- it just won't scale. What you need is a discovery service.

You probably know where I'm going. You can get yourself a fully functioning Eureka service discovery server (<https://github.com/Netflix/eureka>) just by using a single annotation.

### @EnableEurekaServer

Just by adding the **@EnableEurekaServer** annotation to your brand new Spring Boot app, you make it an Eureka discovery service. Now other apps can locate services through it. To take advantage of this wonderful topology, you need to instruct the apps that they should become clients for this Eureka server.

### @EnableDiscoveryClient

Luckily for you, there is a **@EnableDiscoveryClient** annotation. This makes your app register in the service discovery server and then consult with it to discover the other services you need. When all the apps use the discovery client, you don't need to hardcode any IP addresses or URLs. You can always say: please Eureka, give me a location of the service by a given name.

### @EnableCircuitBreaker

Now with all this fancy distributed system going around, you cannot expect things never to break down. Network can and will fail on you. You need a way to mitigate the damage, and still serve something meaningful back to your clients. So, how can developers ensure

resiliency in applications? [Resilience patterns](https://www.jrebel.com/blog/microservices-resilience-patterns)  
(<https://www.jrebel.com/blog/microservices-resilience-patterns>).

In this instance, you need a resilience pattern called the circuit breaker pattern. When things go south, you'll have a way to specify the fallback methods for retrieving data from another service or to respond with reasonable defaults. To make your app aware of the circuit breaker patterns, add the **@EnableCircuitBreaker** annotation. This will configure Hystrix circuit breaker protocols for your application.

And then mark the important methods with the **@HystrixCommand(fallbackMethod = "fallbackMethodName")** annotations. This says that in the case of an emergency when the method throws exceptions or times out, your app should make use of the fallback method. When things are going well, and everything works, this approach will help keep your code neat and maintainable.

## Final Thoughts

In this post, we've looked at many annotations that a Java developer should know if they want to use the Spring Framework. We've covered the most frequently used and perhaps the most important annotations -- those that enable dependency injection for your components, the ways to bind your code to respond to HTTP requests, and how to enable the correct patterns for cloud-based applications and [microservices](https://jrebel.com/rebellabs/why-spring-is-winning-the-microservices-game/)  
(<https://jrebel.com/rebellabs/why-spring-is-winning-the-microservices-game/>). architectures.

## Download the Spring Annotations Cheat Sheet

Ready to download our one-page Spring Annotations cheat sheet pdf? Click the button below to get started!

## Spring Boot and Web annotations

Use annotations to configure your web application.

**@SpringBootApplication** - uses `@Configuration`, `@EnableAutoConfiguration` and `@ComponentScan`.

**@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

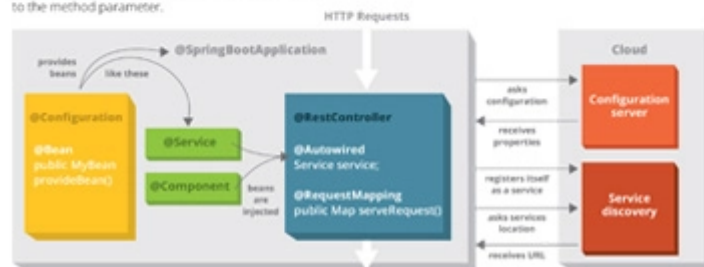
**@Controller** - marks the class as web controller, capable of handling the requests. **@RestController** - a convenience annotation of a **@Controller** and **@ResponseBody**.

 **@ResponseBody** - makes Spring bind method's return value to the web response body.

**@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

**@RequestParam** - bind HTTP parameters into method arguments.

**@PathVariable** - binds placeholder from the URL to the method parameter.



## Spring Cloud annotations

*Make your application work well in the cloud.*

**@EnableConfigServer** - turns your application into a server other apps can get their configuration from.

Use `spring.application.config.uri` in the client `@SpringBootApplication` to point to the config server.

**@EnableEurekaServer** - makes your app an Eureka discovery service, other apps can locate services through it.

**@enableDiscoveryClient** - makes your app register in the service discovery server and discover other services through it.

**@enableCircuitBreaker** - configures Hystrix circuit breaker protocols.


 `@HystrixCommand(fallbackMethod = "fallbackMethodName")` - marks methods to fall back to another method if they cannot succeed normally.

## Spring Framework annotations

Spring uses dependency injection to configure and bind your application together.

**@ComponentScan** - make Spring scan the package for the @Configuration classes.

**@Configuration** - mark a class as a source of bean definitions.

 **@Bean** - indicates that a method produces a bean to be managed by the Spring container.

**@Component** - turns the class into a Spring bean at the auto-scan time. **@Service** - specialization of the @Component, has no encapsulated state.

**CFM @Autowired** - Spring's dependency injection wires an appropriate bean into the marked class member.

**Lazy** - makes @Bean or @Component be initialized on demand rather than eagerly.

**CFM @Qualifier** - filters what beans should be used to @Autowired a field or parameter.

**CM @Value** - indicates a default value expression for the field or parameter, typically something like `"${systemProperties.myProp}"`

**CFM @Required** - fail the configuration, if the dependency cannot be injected.

### Legend

- T - class
- F - field annotation
- C - constructor annotation
- M - method

BRUNNEN

(<https://www.irebel.com/system/files/spring-annotations-cheat-sheet.pdf>)

**GET THE CHEAT SHEET ([HTTPS://WWW.JREBEL.COM/SYSTEM/FILES/SPRING-ANNOTATIONS-CHEAT-SHEET.PDF](https://www.jrebel.com/system/files/spring-annotations-cheat-sheet.pdf))**

## Additional Resources

If you're looking for additional Java cheat sheets, be sure to check out our [Java cheat sheet \(https://www.jrebel.com/resources/java-resources\)](https://www.jrebel.com/resources/java-resources) collection. It's packed full of cheat sheets and shortcuts for popular Java technologies.

Want to see how many developers are using Spring framework in 2020? Check out our latest Java Productivity Report! It shows the most-used frameworks, IDEs, and application servers -- all via our 2020 Java developer survey. You can watch a breakdown of the results via the video below, or download the full, free report by clicking the button below the video.




og/i

**GET THE REPORT (HTTPS://WWW.JREBEL.COM/RESOURCES/JAVA-DEVELOPER-PRODUCTIVITY-REPORT-2020)**

51:33

## Recommended Posts

decorative image for  
spring boot blog  
([/blog/what-is-spring-  
boot](/blog/what-is-spring-boot)).

August 5, 2020

**Java Basics:**  
**What Is**  
**Spring**  
**Boot?**  
**(/blog/what-**  
**is-spring-**  
**boot)**

JAVA  
FRAMEWORKS,  
JAVA APPLICATION  
DEVELOPMENT

decorative image for  
best java frameworks blog  
([/blog/best-java-  
frameworks](/blog/best-java-frameworks)).

May 28, 2020

**Best Java**  
**Frameworks**  
**(/blog/best-**  
**java-**  
**frameworks)**

JAVA  
FRAMEWORKS,  
JAVA APPLICATION  
DEVELOPMENT

([/blog/work-home-tips-  
java-development](/blog/work-home-tips-java-development)).

May 13, 2020

**Work From**  
**Home Tips**  
**for Java**  
**Development**  
**Teams**  
**(/blog/work-**  
**home-tips-**  
**java-**  
**development)**

## PRODUCTS >

JRebel  
(/Products/Jrebel)  
XRebel  
(/Products/Xrebel)  
Java Productivity  
Toolkit  
(/Products/Java-  
Productivity-  
Toolkit)  
Rebel Licenses  
(/Products/Licenses)

## CUSTOMERS >

## RESOURCES >

Papers & Videos  
(/Resources/Papers-  
And-Videos)  
Events &  
Webinars  
(/Resources/Events)  
Recorded  
Webinars  
(/Resources/Recorded-  
Webinars)  
Blog (/Blog)  
ROI Calculator  
(/Calculate-Your-  
Roi-With-Jrebel)

## SUPPORT >

JRebel  
Documentation  
(/Products/Jrebel/Learn)  
XRebel  
Documentation  
(/Products/Xrebel/Learn)  
FAQs  
(/Jrebel/Learn/Faq)

## DOWNLOADS

JRebel Download  
(/Products/Jrebel/Download)  
XRebel Download  
(/Products/Xrebel/Download)  
Rebel Licenses  
On-Premise  
Download  
(/Products/Licenses/On-  
Premise)  
Eclipse Plugin  
(/Jrebel-And-  
Xrebel-Eclipse-  
Plugins)  
IntelliJ Plugin  
(/Jrebel-And-  
Xrebel-IntelliJ-  
Idea-Plugins)

## HUBS >

New Features In  
Java  
(/Resources/New-  
Features-Java)  
Exploring Java  
Microservices  
(/Resources/Exploring-  
Java-  
Microservices)

## COMPANY >

About JRebel By  
Perforce (/About)  
Careers At  
Perforce  
(Https://Www.Perforce.Cc  
Press (/Press)

## CONTACT >

Contact Us  
(/Contact-Us)  
Request Support  
(/Support)  
Subscribe  
(/Subscription-  
Management-  
Center)

JRebel by Perforce  
(<https://www.perforce.com>)  
© 2020 Perforce Software,  
Inc.

[Terms of Use \(/legal/terms-of-use\)](/legal/terms-of-use) | [Privacy Policy \(/legal/privacy-policy\)](/legal/privacy-policy) | [Data Processing Policy \(/legal/data-processing-policy\)](/legal/data-processing-policy) | [Sitemap \(/sitemap\)](/sitemap)

