Spring Annotation Programming Model

Table of Contents

Overview

Over the years, the Spring Framework has continually evolved its support for annotations, *meta-annotations*, and *composed annotations*. This document is intended to aid developers (both end users of Spring as well as developers of the Spring Framework and Spring portfolio projects) in the development and use of annotations with Spring.

See also: MergedAnnotation API internals

Goals of this Document

The primary goals of this document include explanations of the following:

- How to use annotations with Spring.
- How to develop annotations for use with Spring.
- How Spring *finds* annotations (i.e., how Spring's annotation search algorithms work).

Non-goals of this Document

This document does not aim to explain the semantics or configuration options for particular annotations in the Spring Framework. For details on a particular annotation, developers are encouraged to consult the corresponding Javadoc or applicable sections of the reference manual.

Terminology

Meta-Annotations

A ***meta-annotation*** is an annotation that is declared on another annotation. An annotation is therefore *meta-annotated* if it is annotated with another annotation. For example, any annotation that is declared to be *documented* is meta-annotated with `@Documented` from the `java.lang.annotation` package.

Stereotype Annotations

A ***stereotype annotation*** is an annotation that is used to declare the role that a component plays within the application. For example, the `@Repository` annotation in the Spring Framework is a marker for any class that fulfills the role or *stereotype* of a repository (also known as Data Access Object or DAO).

`@Component` is a generic stereotype for any Spring-managed component. Any component annotated with `@Component` is a candidate for component scanning. Similarly, any component annotated with an annotation that is itself meta-annotated with `@Component` is also a candidate for component scanning. For example, `@Service` is meta-annotated with `@Component` .

Core Spring provides several stereotype annotations out of the box, including but not limited to: `@Component`, `@Service`, `@Repository`, `@Controller`, `@RestController`, and `@Configuration`. `@Repository`, `@Service`, etc. are specializations of `@Component`.

Composed Annotations

A ***composed annotation*** is an annotation that is *meta-annotated* with one or more annotations with the intent of combining the behavior associated with those meta-annotations into a single custom annotation. For example, an annotation named `@TransactionalService` that is meta-annotated with Spring's `@Transactional` and `@Service` annotations is a composed annotation that combines the semantics of `@Transactional` and `@Service`. `@TransactionalService` is technically also a custom *stereotype annotation*.

Annotation Presence

The terms ***directly present***, ***indirectly present***, and ***present*** have the same meanings as defined in the class-level Javadoc for `java.lang.reflect.AnnotatedElement` in Java 8.

In Spring, an annotation is considered to be ***meta-present*** on an element if the annotation is declared as a meta-annotation on some other annotation which is *present* on the element. For example, given the aforementioned `@TransactionalService`, we would say that `@Transactional` is *meta-present* on any class that is directly annotated with `@TransactionalService`.

Attribute Aliases and Overrides

An ***attribute alias*** is an alias from one annotation attribute to another annotation attribute. Attributes within a set of aliases can be used interchangeably and are treated as equivalent. Attribute aliases can be categorized as follows.

1. **Explicit Aliases**: if two attributes in one annotation are declared as aliases for each other via `@AliasFor`, they are *explicit aliases*.
2. **Implicit Aliases**: if two or more attributes in one annotation are declared as explicit overrides for the same attribute in a meta-annotation via `@AliasFor`, they are *implicit aliases*.
3. **Transitive Implicit Aliases**: given two or more attributes in one annotation that are declared as explicit overrides for attributes in meta-annotations via `@AliasFor`, if the attributes *effectively* override the same attribute in a meta-annotation following the law of transitivity, they are *transitive implicit aliases*.

An ***attribute override*** is an annotation attribute that *overrides* (or *shadows*) an annotation attribute in a meta-annotation. Attribute overrides can be categorized as follows.

1. **Implicit Overrides**: given attribute `A` in annotation `@One` and attribute `A` in annotation `@Two`, if `@One` is meta-annotated with `@Two`, then attribute `A` in annotation `@One` is an *implicit override* for attribute `A` in annotation `@Two` based solely on a naming convention (i.e., both attributes are named `A`).
2. **Explicit Overrides**: if attribute `A` is declared as an alias for attribute `B` in a meta-annotation via `@AliasFor`, then `A` is an *explicit override* for `B`.
3. **Transitive Explicit Overrides**: if attribute `A` in annotation `@One` is an explicit override for attribute `B` in annotation `@Two` and `B` is an explicit override for attribute `C` in annotation `@Three`, then `A` is a *transitive explicit override* for `C` following the law of transitivity.

Examples

Many of the annotations within the Spring Framework and Spring portfolio projects make use of the `@AliasFor` annotation for declaring *attribute aliases* and *attribute overrides*. Common examples include `@RequestMapping`, `@GetMapping`, and `@PostMapping` from Spring MVC as well as annotations such as `@SpringBootApplication` and `@SpringBootTest` from Spring Boot.

The following sections provide code snippets to demonstrate these features.

Declaring attribute aliases with @AliasFor

Spring Framework 4.2 introduced first-class support for declaring and looking up aliases for annotation attributes. The `@AliasFor` annotation can be used to declare a pair of aliased attributes *within* a single annotation or to declare an alias from one attribute in a custom composed annotation to an attribute in a meta-annotation.

For example, `@ContextConfiguration` from the `spring-test` module is declared as follows.

```
public @interface ContextConfiguration {

    @AliasFor("locations")
    String[] value() default {};

    @AliasFor("value")
    String[] locations() default {};

    // ...
}
```

The `locations` attribute is declared as an alias for the `value` attribute, and vice versa. Consequently, the following declarations of `@ContextConfiguration` are equivalent.

```
@ContextConfiguration("/test-config.xml")
public class MyTests { /* ... */ }
```

```
@ContextConfiguration(value = "/test-config.xml")
public class MyTests { /* ... */ }
```

```
@ContextConfiguration(locations = "/test-config.xml")
public class MyTests { /* ... */ }
```

Similarly, *composed annotations* that override attributes from meta-annotations can use `@AliasFor` for fine-grained control over exactly which attributes are overridden within an annotation hierarchy. In fact, it is even possible to declare an alias for the `value` attribute of a meta-annotation.

For example, one can develop a composed annotation with a custom attribute override as follows.

```
@ContextConfiguration
public @interface MyTestConfig {

    @AliasFor(annotation = ContextConfiguration.class, attribute = "value")
    String[] xmlFiles();

    // ...
}
```

The above example demonstrates how developers can implement their own custom *composed annotations*; whereas, the following demonstrates that Spring itself makes use of this feature in many core Spring annotations.

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@RequestMapping(method = RequestMethod.GET)
public @interface GetMapping {

    /**
     * Alias for {@link RequestMapping#name}.
     */
    @AliasFor(annotation = RequestMapping.class)
    String name() default "";

    /**
     * Alias for {@link RequestMapping#value}.
     */
    @AliasFor(annotation = RequestMapping.class)
    String[] value() default {};

    /**
     * Alias for {@link RequestMapping#path}.
     */
    @AliasFor(annotation = RequestMapping.class)
    String[] path() default {};

    // ...
}
```

Spring Composed and Spring Polyglot

The Spring Composed project is a collection of *composed annotations* for use with the Spring Framework 4.2.1 and higher. There you will find annotations such as `@Get`, `@Post`, `@Put`, and `@Delete` that served as the inspiration for the `@GetMapping`, `@PostMapping`, `@PutMapping`, and `@DeleteMapping` annotations that are now part of Spring MVC and Spring WebFlux.

Feel free to check out `spring-composed` for further examples and inspiration for how you can implement your own custom *composed annotations*, and for a bit of geek humor and entertainment that further demonstrate the power of `@AliasFor`, take a look at Spring Polyglot.

FAQ

1) Can `@AliasFor` be used with the `value` attributes for `@Component` and `@Qualifier`?

The short answer is: no.

The `value` attributes in `@Qualifier` and in *stereotype* annotations (e.g., `@Component`, `@Repository`, `@Controller`, and any custom stereotype annotations) *cannot* be influenced by `@AliasFor`. The reason is that the special handling of these `value` attributes was in place years before `@AliasFor` was invented. Consequently, due to backward compatibility issues it is simply not possible to use `@AliasFor` with such `value` attributes.

Topics yet to be Covered

- Document the general search algorithm(s) for annotations and meta-annotations on classes, interfaces, methods, fields, parameters, and annotations.
  - What happens if an annotation is *present* on an element both locally and as a meta-annotation?
  - How does the presence of `@Inherited` on an annotation (including custom composed annotations) affect the search algorithm?
- Document support for annotation attribute aliases configured via `@AliasFor`.
  - What happens if an attribute *and* its alias are declared in an annotation instance (with the same value or with different values)?
    - Typically an `AnnotationConfigurationException` will be thrown.
- Document support for *composed annotations*.
- Document support for meta-annotation attribute overrides in composed annotations.
  - Document the algorithm used when looking up attributes, specifically explaining:
    - implicit mapping based on naming convention (i.e., composed annotation declares an attribute with the exact same name and type as declared in the *overridden* meta-annotation)
    - explicit mapping using `@AliasFor`
  - What happens if an attribute *and* one of its aliases are declared somewhere within the annotation *hierarchy*? Which one takes precedence?
  - In general, how are conflicts involving annotation attributes resolved?