org.springframework.context.annotation

Annotation Type Profile

```
@Target(value={TYPE,METHOD})
 @Retention(value=RUNTIME)
 @Documented
 @Conditional(value=org.springframework.context.annotation.ProfileCondition.class)
public @interface Profile
```

Indicates that a component is eligible for registration when one or more specified profiles are active.

A *profile* is a named logical grouping that may be activated programmatically via `ConfigurableEnvironment.setActiveProfiles(java.lang.String...)` or declaratively by setting the `spring.profiles.active` property as a JVM system property, as an environment variable, or as a Servlet context parameter in `web.xml` for web applications. Profiles may also be activated declaratively in integration tests via the `@ActiveProfiles` annotation.

The `@Profile` annotation may be used in any of the following ways:

- as a type-level annotation on any class directly or indirectly annotated with `@Component`, including `@Configuration` classes
- as a meta-annotation, for the purpose of composing custom stereotype annotations
- as a method-level annotation on any `@Bean` method

If a `@Configuration` class is marked with `@Profile`, all of the `@Bean` methods and `@Import` annotations associated with that class will be bypassed unless one or more of the specified profiles are active. This is analogous to the behavior in Spring XML: if the `profile` attribute of the `beans` element is supplied e.g., `<beans profile="p1,p2">`, the `beans` element will not be parsed unless at least profile 'p1' or 'p2' has been activated. Likewise, if a `@Component` or `@Configuration` class is marked with `@Profile({"p1", "p2"})`, that class will not be registered or processed unless at least profile 'p1' or 'p2' has been activated.

If a given profile is prefixed with the NOT operator (`!`), the annotated component will be registered if the profile is *not* active — for example, given `@Profile({"p1", "!p2"})`, registration will occur if profile 'p1' is active or if profile 'p2' is *not* active.

If the `@Profile` annotation is omitted, registration will occur regardless of which (if any) profiles are active.

NOTE: With `@Profile` on `@Bean` methods, a special scenario may apply: In the case of overloaded `@Bean` methods of the same Java method name (analogous to constructor overloading), an `@Profile` condition needs to be consistently declared on all overloaded methods. If the conditions are inconsistent, only the condition on the first declaration among the overloaded methods will matter. `@Profile` can therefore not be used to select an overloaded method with a particular argument signature over another; resolution between all factory methods for the same bean follows Spring's constructor resolution algorithm at creation time. **Use distinct Java method names pointing to the same `bean name` if you'd**

like to define alternative beans with different profile conditions; see `ProfileDatabaseConfig` in `@Configuration`'s javadoc.

When defining Spring beans via XML, the `"profile"` attribute of the `<beans>` element may be used. See the documentation in the `spring-beans` XSD (version 3.1 or greater) for details.

Since:

3.1

Author:

Chris Beams, Phillip Webb, Sam Brannen

See Also:

`ConfigurableEnvironment.setActiveProfiles(java.lang.String...)`, `ConfigurableEnvironment.setDefaultProfiles(java.lang.String...)`, `AbstractEnvironment.ACTIVE_PROFILES_PROPERTY_NAME`, `AbstractEnvironment.DEFAULT_PROFILES_PROPERTY_NAME`, `Conditional`, `ActiveProfiles`

Required Element Summary

Required Elements

Modifier and Type	Required Element and Description
`String[]`	**`value`** The set of profiles for which the annotated component should be registered.

Element Detail

value

```
public abstract String[] value
```

The set of profiles for which the annotated component should be registered.