

Spring Annotation Cheat Sheet



Soufiane Mouhtaram

Spring Annotation Cheat Sheet

Spring includes a lot of annotations. Some are annotations created within Spring. Some are annotations called for by various Java specifications. The annotations fall into categories, as follows:

- [Spring Framework](#)
- [REST](#)
- [HATEOAS](#)
- [Session](#)
- [Boot](#)
- [Integration](#)
- [Cloud](#)
- [Data](#)
- [Batch](#)
- [Aspect-oriented Programming](#)
- [Integration Testing](#)
- [JMX](#)
- [Task Execution and Scheduling](#)
- [Cache Abstraction](#)
- [Other](#)

Spring Framework

The Spring Framework is the core project within Spring. The Spring Framework includes annotations in the following categories:

- [Dependency Injection](#)
- [Configuration](#)
- [JMS](#)
- [AMQP](#)

- [Bean Lifecycle](#)
- [MVC/Web](#)
- [CORS Support](#)

Dependency Injection

Spring's dependency injection capability includes the following annotations:

@Resource	Injects the requested resource.
@Autowired	Widely used dependency injection mechanism for constructors, methods, and interfaces.
@Inject	A dependency injection mechanism that can replace <code>@Autowired</code> . <code>@Named</code> may also be used in place of <code>@Autowired</code> and <code>@Inject</code> .
@Named	A dependency injection mechanism that can replace <code>@Autowired</code> and <code>@Inject</code> . <code>@Named</code> may also be used in place of <code>@Autowired</code> and <code>@Inject</code> . <code>@Named</code> is also equivalent to <code>@Component</code> and <code>@ManagedBean</code> .
@ManagedBean	A dependency injection mechanism that can replace <code>@Autowired</code> and <code>@Inject</code> . <code>@ManagedBean</code> is also equivalent to <code>@Component</code> and <code>@Named</code> . However, it is not composable.
@Value	Injection mechanism for fields and methods that indicates a default value. Often used to get values from property files.
@Required	Marks a method (typically a JavaBean setter method) as being 'required'. That is, the method must be configured to be dependency-injected with a value.
@Primary	Indicates that a particular bean should be given preference when multiple beans are candidates to be autowired to a single-valued dependency. If exactly one 'primary' bean exists among the candidates, it will be the autowired value.
@Component	Generic stereotype for any Spring-managed component.
@Service	Indicates that an annotated class is a service.
@Import	Allows for loading <code>@Bean</code> definitions from another configuration class.
@DependsOn	Indicates beans on which the current bean depends.
@ConstructorProperties	Used to explicitly name your constructor arguments.

<u>@Lookup</u>	Indicates 'lookup' methods, to be overridden by the container to redirect them back to the BeanFactory for a getBean call. This is essentially an annotation-based version of the XML lookup-method attribute.
<u>@AliasFor</u>	Used to declare aliases for annotation attributes.

Configuration

Spring's configuration capability includes the following annotations:

<u>@ComponentScan</u>	Configures component scanning directives for use with @Configuration classes.
<u>@Configuration</u>	Indicates that the primary purpose of the annotated class is to provide a source of bean definitions.
<u>@Order</u>	Defines the sort order for an annotated component. Lower values have higher priority. @Priority can replace @Order.
<u>@Priority</u>	Defines the sort order for an annotated component. Lower values have higher priority. @Order can replace @Priority.
<u>@Qualifier</u>	Associates a value with a particular argument. More finely tuned way than @Order and @Priority to control selection.
<u>@Target</u>	Indicates the contexts in which an annotation type is applicable.
<u>@Retention</u>	Indicates how long annotations with the annotated type are to be retained.
<u>@interface</u>	Lets you create custom annotations to use as qualifiers.
<u>@Repository</u>	Indicates that an annotated class is a repository.
<u>@Profile</u>	Indicates that a component is eligible for registration when one or more specified profiles are active.
<u>@Conditional</u>	Indicates that a component is only eligible for registration when all specified conditions match.
<u>@ImportResource</u>	Allows for loading @Bean definitions from another configuration class. Also allows for importing XML configuration files.
<u>@EnableLoadTimeWeaving</u>	Enables load-time weaving, which is used by Spring to dynamically transform classes as they are loaded into the JVM.

<code>@EventListener</code>	Registers an event listener on a public method of a bean.
<code>@Async</code>	Specifies that an event listener is asynchronous. See also: Async under Task Execution and Scheduling .
<code>@PropertySource</code>	Adds a PropertySource to Spring's Environment.

JMS

Spring's support for JMS includes the following annotations:

<code>@EnableJms</code>	Add to an <code>@configuration</code> class to enable support for <code>@JmsListener</code> annotations.
<code>@JmsListener</code>	Indicates that a method of a managed bean is a JMS listener endpoint.
<code>@JmsListeners</code>	Aggregates several <code>@JmsListener</code> annotations. On Java 8, it can be replaced by repeatable <code>@JmsListener</code> annotations.
<code>@Payload</code>	Indicates that a method provides the payload of a message.
<code>@SendTo</code>	Indicates the method (or sometimes class) that responds to a message.
<code>@SendToUser</code>	Indicates that the return value of a message-handling method should be sent as a Message to the specified destination(s) prepended with <code>/user/{username}</code> where the user name is extracted from the headers of the input message being handled.

AMQP

Spring AMQP provides the following annotations:

<code>@Queue</code>	A queue definition used within the bindings attribute of an <code>@QueueBinding</code> .
<code>@QueueBinding</code>	Defines a queue, the exchange it is to be bound to, and an optional binding key. Used with <code>@RabbitListener</code> .
<code>@Exchange</code>	Defines an exchange to which to bind a <code>RabbitListener</code> queue.
<code>@Argument</code>	Represents an argument used when declaring queues etc within a <code>QueueBinding</code> .
<code>@EnableRabbit</code>	Enable Rabbit listener annotated endpoints that are created behind the scenes by a <code>RabbitListenerContainerFactory</code> . To be used on <code>@Configuration</code> classes.
<code>@RabbitHandler</code>	marks a method to be the target of a Rabbit message listener within a class that is annotated with <code>@RabbitListener</code> .

<u>@RabbitListener</u>	Marks a method as the target of a Rabbit message listener on the specified queues (or bindings).
<u>@RabbitListeners</u>	Container annotation that aggregates several @RabbitListener annotations.
<u>@RabbitListenerTest</u>	Enables proxying of @RabbitListener beans to capture arguments and results (if any). Used on @Configuration classes.

Bean Lifecycle

Spring's bean lifecycle management capability includes the following annotations:

<u>@{BeanName}</u>	For example, @foo will find a bean named foo, provided the evaluation context has been configured with a bean resolver.
<u>@Bean</u>	Indicates that a method instantiates, configures and initializes a new object to be managed by the Spring IoC container.
<u>@Description</u>	Adds a description to a bean.
<u>@Component</u>	Generic stereotype for any Spring-managed component.
<u>@Lazy</u>	Causes lazy resolution of a bean in the IoC container.
<u>@Scope</u>	Specifies a non-default scope for a component.
<u>@PostConstruct</u>	Identifies a method to be called after an instance of a bean has been constructed. Used to populate caches and similar operations.
<u>@PreDestroy</u>	Identifies a method to be called before an instance of a bean is to be destroyed. Used to de-populate caches and similar operations.

MVC/Web

Spring provides the following annotations for web applications: Major source: <https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>

<u>@EnableWebMvc</u>	Added to a @configuration class to enable Web MVC.
<u>@Controller</u>	Indicates that a class is an MVC controller.
<u>@RestController</u>	Indicates that a class is a REST controller.
<u>@RequestMapping</u>	Associates a URI path with a method in a controller.
<u>@ModelAttribute</u>	Indicates that a method or argument contributes to a model.
<u>@SessionAttribute</u>	Provides access to pre-existing session attributes that are managed globally.

<u>@SessionAttributes</u>	Declares session attributes used by a specific handler.
<u>@ResponseBody</u>	Causes the return type to be written to the response body (rather than the model).
<u>@RequestParam</u>	Binds a request parameter to a method.
<u>@RequestPart</u>	Associates a handler method argument with part of a multi-part request.
<u>@PathVariable</u>	Binds a method argument to the value of a URI template variable.
<u>@Header</u>	Indicates that a method parameter's value should be retrieved from the message headers.
<u>@Headers</u>	Deprecated. Use @header.
<u>@RequestHeader</u>	Binds a method parameter to a request header.
<u>@RequestBody</u>	Binds a method parameter to the value of the HTTP request body.
<u>@ResponseStatus</u>	Indicates a business exception.
<u>@ControllerAdvice</u>	Allows implementation classes to be auto-detected through classpath scanning. It is automatically enabled when using the MVC namespace or the MVC Java config.
<u>@RestControllerAdvice</u>	As @ControllerAdvice (previous) but @ExceptionHandler methods assume @ResponseBody semantics by default.
<u>@InitBinder</u>	Configures web data binding directly within a controller class.
<u>@MatrixVariable</u>	Identifies the value part of a name/value pair in a URI path.
<u>@cookieValue</u>	Binds a method parameter to the value of an HTTP cookie.
<u>@RequestScope</u>	A specialization of @Scope for a component whose lifecycle is bound to the current web request.
<u>@SessionScope</u>	A specialization of @Scope for a component whose lifecycle is bound to the current web session.
<u>@ApplicationScope</u>	A specialization of @Scope for a component whose lifecycle is bound to the current web application.

Spring MVC provides the following convenience annotations for request mapping:

<u>@GetMapping</u>	Shortcut for @RequestMapping(method = RequestMethod.GET)
<u>@PostMapping</u>	Shortcut for @RequestMapping(method = RequestMethod.POST)

@PutMapping	Shortcut for <code>@RequestMapping(method = RequestMethod.PUT)</code>
@DeleteMapping	Shortcut for <code>@RequestMapping(method = RequestMethod.DELETE)</code>
@PatchMapping	Shortcut for <code>@RequestMapping(method = RequestMethod.PATCH)</code>

CORS Support

Spring MVC/Web includes a single annotation for managing Cross-Origin Resource Support (CORS):

@CrossOrigin	Enables cross-origin resource sharing (CORS) on a path.
------------------------------	---

Security

Spring Security provides the following annotations:

@EnableWebSecurity	Adds Spring Security configuration defined in a <code>WebSecurityConfigurer</code> (often by extending <code>WebSecurityConfigurerAdapter</code>). Must be added to an <code>@Configuration</code> class.
@EnableGlobalMethodSecurity	Class-level annotation that turns on method-level security. Must be on an <code>@Configuration</code> class. You must add other annotations to each method to be secured.
@Secured	Defines a list of security configuration attributes for business methods.
@PreAuthorize	Determines whether a method can actually be invoked or not, usually based on a user role.
@PostAuthorize	Performs an access-control check after the method has been invoked.
@SecurityTestExecutionListeners	Enables only the Spring Security <code>TestExecutionListener</code> classes (rather than all Spring <code>TestExecutionListener</code> classes)
@WithMockUser	Runs a test as a specified mock user.
@WithAnonymousUser	Runs a test as an anonymous user.
@WithUserDetails	Runs a test with user details provided by a custom <code>UserDetailsService</code> .
@WithSecurityContext	Runs a test with a custom <code>SecurityContext</code> .

<u>@PostFilter</u>	After a method has been called, iterates through a returned collection and removes any item that doesn't match the filter.
<u>@PreFilter</u>	Before a method is called, iterates through a collection and removes any item that doesn't match the filter. (Used much more rarely than @PostFilter).
<u>@EnableWebMvcSecurity</u>	Enables Spring Security integration with Spring MVC.
<u>@AuthenticationPrincipal</u>	Automatically resolves the current Authentication.getPrincipal() for Spring MVC arguments.
<u>@EnableAuthorizationServer</u>	Convenience annotation for enabling an authorization Server (that is, an AuthorizationEndpoint and a TokenEndpoint) in the current application context, which must be a DispatcherServlet context.
<u>@EnableResourceServer</u>	Convenience annotation for OAuth2 Resource Servers, enabling a Spring Security filter that authenticates requests via an incoming OAuth2 token.
<u>@EnableOAuth2Client</u>	Enables configuration for an OAuth2 client in a web application that wants to use the Authorization Code Grant from one or more OAuth2 Authorization servers.
<u>@SecuredChannel</u>	Applies the ChannelSecurityInterceptor(s) using provided interceptor() bean name(s).

Spring WebSocket

Spring MVC/Web includes the following annotations for working with WebSockets:

<u>@EnableWebSocket</u>	Enables the processing of WebSocket requests. Must be added to an @Configuration class.
<u>@EnableWebSocketMessageBroker</u>	Enables broker-backed messaging over WebSocket using a higher-level messaging sub-protocol. Must be added to an @Configuration class.
<u>@MessageMapping</u>	Maps a Message onto message-handling methods by matching to the message destination.

<u>@DestinationVariable</u>	Indicates that a method parameter should be bound to a template variable in a destination template string.
<u>@SubscribeMapping</u>	Maps subscription messages onto specific handler methods based on the destination of a subscription.

REST

Spring Data REST provides the following annotations:

<u>@RepositoryRestResource</u>	Marks a repository for custom export mapping and rel attributes.
<u>@Description</u>	Describes the semantics of a resource.
<u>@HandleAfterCreate</u>	Indicates a component that should handle the afterCreate event.
<u>@HandleAfterDelete</u>	Indicates a component that should handle the afterDelete event.
<u>@HandleAfterLinkSave</u>	Indicates a component that should handle the afterLinkSave event.
<u>@HandleAfterSave</u>	Indicates a component that should handle the afterSave event.
<u>@HandleBeforeCreate</u>	Indicates a component that should handle the beforeCreate event.
<u>@HandleBeforeDelete</u>	Indicates a component that should handle the beforeDelete event.
<u>@HandleBeforeLinkDelete</u>	Indicates a component that should handle the beforeLinkDelete event.
<u>@HandleBeforeLinkSave</u>	Indicates a component that should handle the beforeLinkSave event.
<u>@HandleBeforeSave</u>	Indicates a component that should handle the beforeSave event.
<u>@RepositoryEventHandler</u>	Class-level annotation that indicates that the class is an event handler for a repository.
<u>@RestResource</u>	Indicates how a repository should be exported and what the value of the rel attribute in links will be.
<u>@Projection</u>	Ties a particular projection type to a source type. Used to find projection interfaces at startup time.

<u>@Version</u>	Identifies a property to be used as version field to implement optimistic locking on entities.
<u>@AccessType</u>	Defines how Spring Data accesses values of persistent properties.
<u>@CreatedBy</u>	Declares a field as the one representing the principal that created the entity containing the field.
<u>@CreatedDate</u>	Declares a field as the one representing the date the entity containing the field was created.
<u>@Id</u>	Indicates an identifier.
<u>@LastModifiedBy</u>	Declares a field as the one representing the principal that recently modified the entity containing the field.
<u>@LastModifiedDate</u>	Declares a field as the one representing the date the entity containing the field was recently modified.
<u>@PersistenceConstructor</u>	Indicates that a constructor, even one that's package protected, as the primary constructor used by the persistence logic.
<u>@Persistent</u>	Indicates that a field should be persisted even if there are no getter and setter methods for it.
<u>@QueryAnnotation</u>	Meta-Annotation to mark a store-specific annotation as a query annotation. This allows generic special handing of finder methods on Repository interfaces.
<u>@ReadOnlyProperty</u>	Marks a field as read-only for the mapping framework. The field will not be persisted.
<u>@Reference</u>	Meta-annotation to indicate annotations that mark references to other objects.
<u>@Transient</u>	Marks a field to be transient for the mapping framework.
<u>@TypeAlias</u>	Lets String based type aliases to be used when writing type information for PersistentEntity objects.
<u>@BasePathAwareController</u>	Indicates a controller that declares request mappings to be augmented with a base URI in the Spring Data REST configuration.
<u>@RepositoryRestController</u>	Identifies Spring MVC controllers provided by Spring Data REST.

HATEOAS

Spring HATEOAS provides the following annotations:

@EnableHypermediaSupport

Enables support for a particular hypermedia representation type.

@EnableEntityLinks

Enables dependency injection of EntityLinks objects.

@ExposesResourceFor

Class-level annotation for controllers. Indicates which model type the controller manages.

@Relation

Indicates the relation to be used when embedding objects in hypermedia.

Session

Spring Session provides the following annotations:

@EnableRedisHttpSession

Exposes the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by Redis. Must be added to an @Configuration class.

@EnableGemFireHttpSession

Exposes the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by Pivotal GemFire or Apache Geode. Must be added to an @Configuration class.

@EnableJdbcHttpSession

Exposes the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by a relational database. Must be added to an @Configuration class.

@EnableMongoHttpSession

Exposes the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by Mongo. Must be added to an @Configuration class.

@EnableHazelcastHttpSession

Exposes the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by Hazelcast. Must be added to an @Configuration class.

@EnableSpringHttpSession

Exposes the SessionRepositoryFilter as a bean named "springSessionRepositoryFilter" and backed by a user provided

implementation of SessionRepository. Must be added to an @Configuration class.

Boot

Spring Boot provides the following annotations:

[@SpringBootApplication](#)

Convenience annotation that includes @Configuration, @EnableAutoConfiguration, and @ComponentScan

[@EnableAutoConfiguration](#)

tells Spring Boot to determine how you will want to configure Spring, based on the jar dependencies that you have added.

[@EntityScan](#)

Configures the base packages used by auto-configuration when scanning for entity classes.

[@ConfigurationProperties](#)

Identifies a class a configuration properties class, which can then be used to control and validate configuration.

[@@EnableConfigurationProperties](#)

Enables support for @ConfigurationProperties annotated beans.

[@ConfigurationPropertiesBinding](#)

Qualifier for beans that are needed to configure the binding of ConfigurationProperties (often converters).

[@JsonComponent](#)

Provides JsonSerializer and/or JsonDeserializer implementations to be registered with Jackson when JsonComponentModule is in use.

[@ServletComponentScan](#)

Enables automatic registration of classes annotated with @WebServlet, @WebFilter, and @WebListener.

[@EnableOAuth2Sso](#)

Enables OAuth2 Single Sign On (SSO).

[@SpringBootTest](#)

Creates an ApplicationContext object that supports testing a Spring Boot application.

[@AutoConfigureMockMvc](#)

Configures a `MockMvc` object for use when testing Spring Boot applications.

[@SpringBootConfiguration](#)

Indicates that a class provides Spring Boot application `@Configuration`. Can be used as an alternative to the Spring's standard `@Configuration` annotation so that configuration can be found automatically (for example in tests). An application should include only one `@SpringBootConfiguration`, and most idiomatic Spring Boot applications will inherit it from `@SpringBootApplication`.

[@TestConfiguration](#)

`@Configuration` that can be used to define additional beans or customizations for a test. Unlike regular `@Configuration` classes the use of `@TestConfiguration` does not prevent auto-detection of `@SpringBootConfiguration`.

[@LocalServerPort](#)

Annotation at the field or method/constructor parameter level that injects the HTTP port that got allocated at runtime. Provides a convenient alternative for `@Value("${local.server.port}")`.

[@MockBean](#)

Adds a mock bean to a Spring `ApplicationContext`.

[@Spybean](#)

Applies Mockito spies to a Spring `ApplicationContext`.

[@ImportAutoConfiguration](#)

Imports and applies the specified auto-configuration classes. Sometimes useful for testing. Generally, `@EnableAutoConfiguration` should be preferred.

[@JsonTest](#)

Auto-configures Jackson `ObjectMapper`, any `@JsonComponent` beans and any Jackson Modules.

[@WebMvcTest](#)

Used with `@RunWith(SpringRunner.class)` for a typical Spring MVC test. Can be used when a test focuses only on Spring MVC components. Using this annotation will disable full auto-configuration and instead apply only configuration relevant to MVC tests.

[@DataJpaTest](#)

Annotation that can be used in combination with `@RunWith(SpringRunner.class)` for a typical JPA test. Can be used when a test focuses **only** on JPA components.

[@AutoConfigureTestEntityManager](#)

Can be applied to a test class to enable auto-configuration of a `TestEntityManager`.

@AutoConfigureTestDatabase

Can be applied to a test class to configure a test database to use instead of any application defined or auto-configured `DataSource`.

@JdbcTest

Annotation that can be used in combination with `@RunWith(SpringRunner.class)` for a typical JDBC test. By default, it will also configure an in-memory embedded database and a `JdbcTemplate`.

@DataMongoTest

Used to test MongoDB applications. By default, it will configure an in-memory embedded MongoDB (if available), configure a `MongoTemplate`, scan for `@Document` classes and configure Spring Data MongoDB repositories.

@RestClientTest

Used to test REST clients. By default, it will auto-configure Jackson and GSON support, configure a `RestTemplateBuilder` and add support for `MockRestServiceServer`.

@AutoConfigureRestDocs

Used to use Spring REST Docs in your tests. It will automatically configure `MockMvc` to use Spring REST Docs and remove the need for Spring REST Docs' JUnit rule.

@WebIntegrationTest

Test class annotation signifying that the tests are "web integration tests" and therefore require full startup in the same way as a production application (listening on normal ports). Normally used in conjunction with `@SpringApplicationConfiguration`.

@SpringApplicationConfiguration

Class-level annotation that is used to determine how to load and configure an `ApplicationContext` for integration tests. Similar to the standard `ContextConfiguration` but uses Spring Boot's `SpringApplicationContextLoader`.

@ConditionalOnClass

Matches only when the specified classes are on the classpath.

@ConditionalOnMissingBean

Matches only when the specified bean classes and/or names are not already contained in the `BeanFactory`.

@AutoConfigureBefore

Used when configurations need to be loaded in a particular order.

@AutoConfigureAfter

Used when configurations need to be loaded in a particular order.

@AutoconfigureOrder

Allows for ordering certain auto-configurations that shouldn't have any direct knowledge of each other.

@ConditionalOnProperty

Checks whether the specified properties have the specified value.

@ConditionalOnResource

Lets configuration to be included only when a specific resource is present.

@ConditionalOnWebApplication

Matches only when the application context is a web application

@ConditionalOnNotWebApplication

Matches only when the application context is not a web application.

@ConditionalOnExpression

Lets configuration be included based on the result of a SpEL expression.

@ManagementContextConfiguration

Specialized @Configuration class that defines configuration specific for the management context. Configurations should be registered in /META-INF/spring.factories under the org.springframework.boot.actuate.autoconfigure.ManagementContextConfiguration key.

@ExportMetricWriter

Qualifier annotation for a metric repository that is to be used to export metrics from the ExportMetricReader readers.

@ExportMetricReader

Qualifier annotation for a metric reader that can be exported (to distinguish it from others that might be installed by the user for other purposes).

@FlywayDataSource

Specifies a DataSource to be injected into Flyway. If used for a second data source, the other (main) one would normally be marked as `@Primary`.

@LiquibaseDataSource

Specifies a DataSource to be injected into Liquibase. If used for a second data source, the other (main) one would normally be marked as `@Primary`.

@DeprecatedConfigurationProperty

Indicates that a getter in a ConfigurationProperties object is deprecated. This annotation has no bearing on the actual binding processes, but it is used by the spring-boot-configuration-processor to add deprecation meta-data.

@NestedConfigurationProperty

Indicates that a field in a ConfigurationProperties object should be treated as if it were a nested type.

Integration

Spring integration includes the following annotations:

@EnableIntegration

Enables Spring Integration infrastructure, registers built-in beans, adds BeanFactoryPostProcessors, adds BeanPostProcessors, and adds annotation processors.

@EnableIntegrationManagement

Enables default configuration of management in Spring Integration components in an existing application.

@EnableMessageHistory

Enables MessageHistory for Integration components.

@EnablePublisher

Provides the registration for the PublisherAnnotationBeanPostProcessor to allow the use of the Publisher annotation.

@IntegrationComponentScan

Configures component scanning directives for use with Configuration classes.

@Publisher

Indicates that a method (or all public methods if applied at class-level) should publish Messages.

@GlobalChannelInterceptor

ChannelInterceptor components with this annotation will be applied as global channel interceptors using the provided patterns to match channel names.

@Aggregator

Indicates that a method is capable of aggregating messages.

@BridgeFrom

Marks a Bean method for a MessageChannel to produce a BridgeHandler and Consumer Endpoint.

@BridgeTo

Marks a Bean method for a MessageChannel to produce a BridgeHandler and Consumer Endpoint.

@CorrelationStrategy

Indicates that a given method is capable of determining the correlation key of a message sent as parameter.

@Filter

Indicates that a method is capable of playing the role of a Message Filter.

@Gateway

Indicates that an interface method is capable of mapping its parameters to a message or message payload.

@GatewayHeader

Provides the message header value or expression.

@IdempotentReceiver

A method that has a Messaging annotation (@code, @ServiceActivator, @Router etc.) that also has this annotation has an IdempotentReceiverInterceptor applied to the associated MessageHandler.handleMessage(org.springframework.messaging.Message<?>) method.

@InboundChannelAdapter

Indicates that a method is capable of producing a Message payload.

@IntegrationComponentScan

Configures component scanning directives for use with Configuration classes.

@MessageEndpoint

Stereotype annotation indicating that a class is capable of serving as a Message endpoint.

@MessagingGateway

Provides an Integration Messaging Gateway Proxy (<gateway/>) as an abstraction over the messaging API.

@Payloads

Marks a method parameter as being a list of message payloads, for POJO handlers that deal with lists of messages.

@Poller

Provides the `PollerMetadata` options for the Messaging annotations for polled endpoints.

@Publisher

Indicates that a method (or all public methods if applied at class-level) should publish Messages.

@ReleaseStrategy

Indicates that a method is capable of asserting if a list of messages or payload objects is complete.

@Role

Assigns endpoints to a role. The assigned endpoints can be started and stopped as a group.

@Router

Indicates that a method is capable of resolving to a channel or channel name based on a message, message header(s), or both.

@ServiceActivator

Indicates that a method is capable of handling a message or message payload.

@Splitter

Indicates that a method is capable of splitting a single message or message payload to produce multiple messages or payloads.

@Transformer

Indicates that a method is capable of transforming a message, message header, or message payload.

@IntegrationConverter

Registers `Converter`, `GenericConverter` Or `ConverterFactory` beans for the `integrationConversionService`.

@EnableIntegrationMBeanExport

Enables default exporting for Spring Integration components in an existing application and all `@ManagedResource` annotated beans.

@EnableMBeanExport

Enables default exporting of all standard MBeans from the Spring context and all `@ManagedResource` annotated beans.

@IntegrationManagedResource

Clone of ManagedResource limiting beans thus annotated so that they will only be exported by the IntegrationMBeanExporter and prevented from being exported by other MBeanExporters (if present).

[@EnableIntegrationGraphController](#)

Enables the IntegrationGraphController if DispatcherServlet is present in the classpath.

Cloud

Spring Cloud includes the following annotations:

{JB: Start here: <http://cloud.spring.io/spring-cloud-static/spring-cloud.html>}

@RefreshScope	Lets beans be refreshed dynamically at runtime.
@EnableDiscoveryClient	looks for implementations of the DiscoveryClient interface via META-INF/spring.factories.
@EnableConfigServer	Embeds the Spring Cloud Config Server in another Spring application.
@EnableEurekaServer	Enables the Netflix Eureka Service Discovery client.
@EnableCircuitBreaker	Enables a circuit breaker implementation for an application.
@EnableHystrix	Enables the Hystrix circuit breaker. Must go on an application class (such as a class marked with @SpringBootApplication).
@HystrixCommand	Indicates that a bean should be wrapped in a proxy that is connected to the Hystrix circuit breaker.
@HystrixProperty	Sets a property for the @HystrixCommand annotation. See the @Hystrix wiki .
@EnableHystrixDashboard	Enables the Hystrix dashboard. Must go on a Spring Boot main class.
@EnableTurbine	Enables the Turbine application for a Spring application. Must go on a Spring Boot main class.
@EnableTurbineStream	Enables the Turbine Stream application for a Spring application. Must go on a Spring Boot main class.
@FeignClient	Declares that a REST client should be created for the specified interface.
@EnableFeignClients	Scans for interfaces that declare they are feign clients. Must go on the application class.

<u>@RibbonClient</u>	Declares a ribbon client. Must go on an <code>@Configuration</code> class.
<u>@RibbonClients</u>	Convenience annotation that combines multiple <code>@RibbonClient</code> annotations on a single class (including in Java 7).
<u>@EnableZuulProxy</u>	Sets up a Zuul server endpoint and installs some reverse proxy filters in it, so it can forward requests to backend servers. The backends can be registered manually through configuration or via a <code>DiscoveryClient</code> .
<u>@EnableAtlas</u>	Enables Atlas metrics publishing.
<u>@EnableBinding</u>	Enables the binding of targets annotated with <code>Input</code> and <code>Output</code> to a broker, according to the list of interfaces passed as value to the annotation.
<u>@StreamListener</u>	Indicates a method that is a listener to the inputs declared through <code>@EnableBinding</code> .
<u>@Input</u>	Indicates that an input binding target will be created by the framework.
<u>@Output</u>	Indicates that an output binding target will be created by the framework.
<u>@EnableRxJavaProcessor</u>	Class annotation that identifies the class as an RxJava processor module.
<u>@EnableZipkinStreamServer</u>	Enables transporting of spans over a Spring Cloud Stream (such as RabbitMQ).
<u>@SpanName</u>	Names a span for use with a stream.
<u>@LoadBalanced</u>	Indicates that a <code>RestTemplate</code> bean should be configured to use a <code>LoadBalancerClient</code> .

Data

Spring includes the following annotations:

<u>@EnableJpaRepositories</u>	Class-level annotation that enables JPA repositories. By default, scans the package of the annotated <code>@Configuration</code> class for Spring Data repositories.
<u>@EnableJpaAuditing</u>	Class-level annotation that enables auditing in JPA. Must be on an <code>@Configuration</code> class.
<u>@RepositoryDefinition</u>	Indicates the interfaces for which a repository proxy is to be created. Annotating an interface with <code>@RepositoryDefinition</code> will cause the same behavior as extending <code>Repository</code> .

<u>@NoRepositoryBean</u>	Indicates an interface for which Spring should not create an instance at runtime.
<u>@Document</u>	Identifies a domain object to be persisted to MongoDB.
<u>@DBRef</u>	Indicates that the annotated field is to be stored using a DBRef.
<u>@Field</u>	Defines custom metadata for a document field.
<u>@Language</u>	Marks a property as being a language field.
<u>@TextScore</u>	Marks a property to be considered when doing a full-text search. Important: The property will not be saved when the entity is saved.
<u>@EnableMongoRepositories</u>	Activates MongoDB repositories. If no base package is configured through <code>value()</code> , <code>basePackages()</code> , or <code>basePackageClasses()</code> , it will trigger scanning of the package of annotated class.
<u>@Query</u>	Declares a finder query on a repository method.
<u>@EntityGraph</u>	Configures the JPA 2.1 EntityGraphs that should be used on repository methods.
<u>@Lock</u>	Specifies the <code>LockModeType</code> to be used when executing a query.
<u>@Modifying</u>	Indicates a method should be regarded as a modifying query.
<u>@QueryHints</u>	Applies JPA query hints to a query declared in a repository interface.
<u>@Temporal</u>	Declares an appropriate <code>TemporalType</code> on query method parameters. Can only be used on parameters of type <code>Date</code> .
<u>@DomainEvents</u>	Indicates a method that publishes domain events. A method marked with <code>@AfterDomainEventsPublication</code> can then be used to manipulated the published events.
<u>@AfterDomainEventPublication</u>	Indicates a method that manipulates published domain events (often for selecting only events that meet some particular criterion).
<u>@EnableSpringDataWebSupport</u>	Automatically register the following beans for usage with Spring MVC: <code>DomainClassConverter</code> , <code>PageableHandlerMethodArgumentResolver</code> , and <code>SortHandlerMethodArgumentResolver</code> . If Spring HATEOAS is on the classpath, it also registers <code>HateoasPageableHandlerMethodArgumentResolver</code> , <code>HateoasSortHandlerMethodArgumentResolver</code> , <code>PagedResourcesAssembler</code> , and <code>SortHandlerMethodArgumentResolver</code> .
<u>@PageableDefault</u>	Set defaults when injecting a <code>Pageable</code> into a controller method.
<u>@SortDefault</u>	Defines the default <code>Sort</code> options to be used when injecting a <code>Sort</code> instance into a controller handler method.

<u>@SortDefaults</u>	Wrapper annotation to allow declaring multiple SortDefault annotations on a method parameter.
<u>@JsonPath</u>	Declares a JSON Path expression on a projection interface.
<u>@ProjectedPayload</u>	Enables projection and projection method annotations that contain JSON or XPath expressions.
<u>QuerydslPredicate</u>	Customizes the binding of HTTP request parameters to a Querydsl com.mysema.query.types.Predicate in Spring MVC handler methods.
<u>@Procedure</u>	Declares a JPA 2.1 stored procedure mapping directly on a repository method.
<u>@EnableTransactionManagement</u>	Enables Spring's annotation-driven transaction management capability.

Batch

Spring Batch includes the following annotations:

[@AfterChunk](#)

Marks a method to be called after a chunk is executed.

[@AfterChunkError](#)

Marks a method to be called after a has failed and been marked for rollback.

[@AfterJob](#)

Marks a method to be called after a Job has completed.

[@AfterProcess](#)

Marks a method to be called after an item is passed to an ItemProcessor.

[@AfterRead](#)

Marks a method to be called after an item is read from an ItemReader.

[@AfterStep](#)

Marks a method to be called after a Step has completed.

[@AfterWrite](#)

Marks a method to be called after an item is passed to an ItemWriter.

[@BeforeChunk](#)

Marks a method to be called before a chunk is executed.

[@BeforeJob](#)

Marks a method to be called before a `Job` is executed, which comes after a `JobExecution` is created and persisted, but before the first step is executed.

@BeforeProcess

Marks a method to be called before an item is passed to an `ItemProcessor`.

@BeforeRead

Marks a method to be called before an item is read from an `ItemReader`.

@BeforeStep

Marks a method to be called before a step is executed, which comes after a `StepExecution` is created and persisted, but before the first item is read.

@BeforeWrite

Marks a method to be called before an item is passed to an `ItemWriter`.

@OnProcessError

Marks a method to be called if an exception is thrown by an `ItemProcessor`.

@OnReadError

Marks a method to be called if an exception is thrown by an `ItemReader`.

@OnSkipInProcess

Marks a method to be called when an item is skipped due to an exception thrown in the `ItemProcessor`.

@OnSkipInRead

Marks a method to be called when an item is skipped due to an exception thrown in the `ItemReader`.

@OnSkipInWrite

Marks a method to be called when an item is skipped due to an exception thrown in the `ItemWriter`.

@OnWriteError

Marks a method to be called if an exception is thrown by an `ItemWriter`.

@EnableBatchProcessing

Enable Spring Batch features and provide a base configuration for setting up batch jobs in an `@Configuration` class, roughly equivalent to using the `<batch:*>` XML namespace.

@JobScope

Convenience annotation for job-scoped beans that defaults the proxy mode, so that it doesn't have to be specified explicitly on every bean definition.

@StepScope

Convenience annotation for step-scoped beans that defaults the proxy mode, so that it doesn't have to be specified explicitly on every bean definition.

@Classifier

Mark a method as capable of classifying its input to an instance of its output.

Aspect-oriented Programming

Spring includes a set of annotations for working with Aspect-oriented Programming (AOP):

<u>@EnableAspectJAutoProxy</u>	Enables support for handling components marked with AspectJ's <code>@Aspect</code> annotation. Must be used on a class that is also marked with the <code>@Configuration</code> annotation (or another annotation that includes the <code>@Configuration</code> annotation).
<u>@Aspect</u>	Indicates that a class is an aspect.
<u>@Pointcut</u>	Defines a join point.
<u>@target</u>	Limits matching to join points. (Not to be confused with <code>@Target</code> for <code>@Configuration</code> classes.)
<u>@args</u>	Limits matching to join points
<u>@within</u>	Limits matching to join points within types that have the given annotation.
<u>@annotation</u>	limits matching to join points where the subject of the join point has the given annotation
<u>@Transactional</u>	Class annotation that specifies the default transaction semantics for the execution of any public operation in the class.
<u>@Before</u>	Declares pointcut advice that should run before methods matched by the pointcut.
<u>@AfterReturning</u>	Declares pointcut advice that should run after the methods matched by the pointcut. The methods must return normally. See <code>@AfterThrowing</code> and <code>@After</code> .
<u>@AfterThrowing</u>	Declares pointcut advice that should run after the methods matched by the pointcut have thrown an exception.
<u>@After</u>	Declares pointcut advice to run after the methods matched by the pointcut have run, whether they returned normally or threw an exception. Parallel to <code>finally</code> in a try-catch-finally block.

<u>@Around</u>	Declares advice that runs around (potentially both before and after) the methods matched by the pointcut. Can also determine if pointcut methods run. Do not use if @Before or @After suffice.
<u>@DeclareParents</u>	Declares that matching types have a new parent.
<u>@Configurable</u>	Marks a class as eligible for Spring-driven configuration.
<u>@EnableSpringConfigured</u>	Tells the current application context to apply dependency injection to non-managed classes that are instantiated outside of the Spring bean factory.

Integration Testing

Spring includes a set of annotations for working with integration testing:

<u>@BootstrapWith</u>	Specifies a custom TestContextBootstrapper class.
<u>@ContextConfiguration</u>	Defines class-level metadata that determines how to load and configure an ApplicationContext for integration tests.
<u>@WebAppConfiguration</u>	Class-level annotation that is used to declare that the ApplicationContext loaded for an integration test should be a WebApplicationContext.
<u>@ContextHierarchy</u>	Class-level annotation that defines a hierarchy of ApplicationContexts for integration tests.
<u>@ActiveProfiles</u>	Class-level annotation that indicates which bean definition profiles should be active.
<u>@TestPropertySource</u>	Class-level annotation that configures the locations of properties files and inlined properties to be added to the set of PropertySources in the Environment for an ApplicationContext loaded for an integration test.
<u>@DirtiesContext</u>	Indicates that the underlying Spring ApplicationContext has been modified or corrupted in some manner during the execution of a test and should be closed. When an application context is marked as dirty, it is removed from the testing framework's cache and closed.
<u>@TestExecutionListeners</u>	Defines class-level metadata for configuring the TestExecutionListener implementations that should be registered with the TestContextManager.
<u>@Commit</u>	Causes a transaction to commit rather than rollback during testing. Use only when you want a test to modify a database. See @Rollback.

<u>@Rollback</u>	indicates whether the transaction for a transactional test method should be rolled back after the test method has completed. See <u>@Commit</u> .
<u>@BeforeTransaction</u>	Indicates that the annotated void method should be executed before a transaction is started for test methods configured to run within a transaction via Spring's <u>@Transactional</u> annotation.
<u>@AfterTransaction</u>	Indicates that the annotated void method should be executed after a transaction is ended for test methods configured to run within a transaction via Spring's <u>@Transactional</u> annotation.
<u>@Sql</u>	Annotates a test class or test method to configure SQL scripts to be executed against a given database during integration tests.
<u>@SqlConfig</u>	Defines metadata that is used to determine how to parse and execute SQL scripts configured via the <u>@Sql</u> annotation.
<u>@SqlGroup</u>	Container annotation that aggregates several <u>@Sql</u> annotations.

Unit Testing

Spring includes a set of annotations for unit testing:

<u>@IfProfileValue</u>	Indicates that, if the value returned by the name argument matches the value of the value argument, the annotated test is enabled for a specific testing environment.
<u>@ProfileValueSourceConfiguration</u>	Class-level annotation that specifies what type of <u>ProfileValueSource</u> to use when retrieving profile values configured through the <u>@IfProfileValue</u> annotation.
<u>@Timed</u>	Indicates that the annotated test method must finish execution in a specified time period (in milliseconds).
<u>@Repeat</u>	Indicates that the annotated test method must be executed a number of times.

JMX

Spring includes a set of annotations for working with Java Managed Extensions (JMX):

<code>@ManagedResource</code>	Marks all instances of a Class as JMX managed resources.
<code>@ManagedAttribute</code>	Mark a getter or setter as one half of a JMX attribute.
<code>@ManagedOperation</code>	Mark a method as a JMX operation.
<code>@ManagedOperationParameters</code>	Define descriptions for operation parameters.
<code>@ManagedOperationParameter</code>	Define the descriptions for an operation parameter.
<code>@EnableMBeanExport</code>	Class-level application that indicates whether an application is an exporter of managed beans.

Task Execution and Scheduling

Spring includes a set of annotations to support task execution and scheduling:

<code>@Scheduled</code>	Indicates that a method should be called on a scheduled basis.
<code>@Async</code>	Indicates that a method may be called asynchronously.
<code>@EnableScheduling</code>	Enables support for the <code>@Schedule</code> annotation.
<code>@EnableAsync</code>	Enables support for the <code>@Async</code> annotation.

Cache Abstraction

Spring includes a set of annotations for working with caching:

<code>@Cacheable</code>	Indicates a method whose result is cacheable.
<code>@CacheEvict</code>	Indicates that a method performs cache eviction (removes items from a cache).
<code>@CachePut</code>	Indicates a method whose result will always be put in a cache.
<code>@Caching</code>	Allows multiple nested <code>@Cacheable</code> , <code>@CachePut</code> and <code>@CacheEvict</code> to be used on the same method
<code>@CacheConfig</code>	Class-level annotation that allows sharing of the cache names, the custom <code>KeyGenerator</code> , the custom <code>CacheManager</code> , and the custom <code>CacheResolver</code> . Placing this annotation on the class does not turn on any caching operation. See <code>@EnableCaching</code> .
<code>@EnableCaching</code>	Turns on caching in an application. Must go on an <code>@Configuration</code> class.

<u>@CacheResult</u>	Similar to @Cacheable but can cache specific exceptions and force the execution of the method regardless of the content of the cache.
<u>@CacheRemove</u>	Similar to @CacheEvict but can support conditional removal if the method throws an exception.
<u>@CacheRemoveAll</u>	Similar to @CacheEvict(allEntries=true) but can support conditional removal if the method throws an exception.
<u>@CacheDefaults</u>	Similar to @CacheConfig.
<u>@CacheValue</u>	Usable with @CachePut to update the cache before or after method invocation.
<u>@CacheKey</u>	Optional method parameter annotation to indicate which argument(s) should be the key. (The default is to construct the key from all the parameters, unless one or more parameters are marked with @CacheKey.)

Other

Spring includes a few other annotations that don't fit into the preceding categories:

<u>@ExceptionHandler</u>	Annotation for handling exceptions in specific handler classes and/or handler methods.
<u>@NumberFormat</u>	Declares that a field or method parameter should be formatted as a number.
<u>@DateTimeFormat</u>	Declares that a field or method parameter should be formatted as a date or time.