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

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

- 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 @Autowired. @Named may also be used in place of @Autowired and @Inject.
@Named	A dependency injection mechanism that can replace @Autowired and @Inject. @Named may also be used in place of @Autowired and @Inject. @Named is also equivalent to @Component and @ManagedBean.
@ManagedBean	A dependency injection mechanism that can replace @Autowired and @Inject. @ManagedBean is also equivalent to @Component and @Named. 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 @Bean definitions from another configuration class.
@DependsOn	Indicates beans on which the current bean depends.
@ConstructorProperties	Used to explicitly name your constructor arguments.
@Lookup	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.
@AliasFor	Used to declare aliases for annotation attributes.

Configuration

Spring's configuration capability includes the following annotations:

@ComponentScan	Configures component scanning directives for use with @Configuration classes.
@Configuration	Indicates that the primary purpose of the annotated class is to provide a source of bean definitions.
@Order	Defines the sort order for an annotated component. Lower values have higher priority. @Priority can replace @Order.
@Priority	Defines the sort order for an annotated component. Lower values have higher priority. @Order can replace @Priority.
@Qualifier	Associates a value with a particular argument. More finely tuned way than @Order and @Priority to control selection.
@Target	Indicates the contexts in which an annotation type is applicable.
@Retention	Indicates how long annotations with the annotated type are to be retained.
@interface	Lets you create custom annotations to use as qualifiers.

@Repository	Indicates that an annotated class is a repository.
@Profile	Indicates that a component is eligible for registration when one or more specified profiles are active.
@Conditional	Indicates that a component is only eligible for registration when all specified conditions match.
@ImportResource	Allows for loading @Bean definitions from another configuration class. Also allows for importing XML configuration files.
@EnableLoadTimeWeaving	Enables load-time weaving, which is used by Spring to dynamically transform classes as they are loaded into the JVM.
@EventListener	Registers an event listener on a public method of a bean.
@Async	Specifies that an event listener is asynchronous. See also: Async under Task Execution and Scheduling.
@PropertySource	Adds a PropertySource to Spring's Environment.

JMS

Spring's support for JMS includes the following annotations:

@EnableJms	Add to an @configuration class to enable support for @JmsListener annotations.
@JmsListener	Indicates that a method of a managed bean is a JMS listener endpoint.
@JmsListeners	Aggregates several @JmsListener annotations. On Java 8, it can be replaced by repeatable @JmsListener annotations.
@Payload	Indicates that a method provides the payload of a message.
@SendTo	Indicates the method (or sometimes class) that responds to a message.

@SendToUser	Indicates that the return value of a message-handling method should be sent as a Message to the specified destination(s) prepended with /user/{username} where the user name is extracted from the headers of the input message being handled.

AMQP

Spring AMQP provides the following annotations:

@Queue	A queue definition used within the bindings attribute of an @QueueBinding.
@QueueBinding	Defines a queue, the exchange it is to be bound to, and an optional binding key. Used with @RabbitListener.
@Exchange	Defines an exchange to which to bind a RabbitListener queue.
@Argument	Represents an argument used when declaring queues etc within a QueueBinding.
@EnableRabbit	Enable Rabbit listener annotated endpoints that are created behind the scenes by a RabbitListenerContainerFactory. To be used on @Configuration classes.
@RabbitHandler	marks a method to be the target of a Rabbit message listener within a class that is annotated with @RabbitListener.
@RabbitListener	Marks a method as the target of a Rabbit message listener on the specified queues (or bindings).
@RabbitListeners	Container annotation that aggregates several @RabbitListener annotations.
@RabbitListenerTest	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:

@{BeanName}	For example, @foo will find a bean named foo, provided the evaluation context has been configured with a bean resolver.
@Bean	Indicates that a method instantiates, configures and initializes a new object to be managed by the Spring IoC container.

@Description	Adds a description to a bean.
@Component	Generic stereotype for any Spring-managed component.
@Lazy	Causes lazy resolution of a bean in the IoC container.
@Scope	Specifies a non-default scope for a component.
@PostConstruct	Identifies a method to be called after an instance of a bean has been constructed. Used to populate caches and similar operations.
@PreDestroy	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

@EnableWebMvc	Added to a @configuration class to enable Web MVC.
@Controller	Indicates that a class is an MVC controller.
@RESTController	Indicates that a class is a REST controller.
@RequestMapping	Associates a URI path with a method in a controller.
@ModelAttribute	Indicates that a method or argument contributes to a model.
@SessionAttribute	Provides access to pre-existing session attributes that are managed globally.
@SessionAttributes	Declares session attributes used by a specific handler.
@ResponseBody	Causes the return type to be written to the response body (rather than the model).
@RequestParam	Binds a request parameter to a method.

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

@GetMapping	Shortcut for @RequestMapping(method = RequestMethod.GET)
@PostMapping	Shortcut for @RequestMapping(method = RequestMethod.POST)
@PutMapping	Shortcut for @RequestMapping(method = RequestMethod.PUT)
@DeleteMapping	Shortcut for @RequestMapping(method = RequestMethod.DELETE)
@PatchMapping	Shortcut for @RequestMapping(method = RequestMethod.PATCH)

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 WebSecurityConfigurer (often by extending WebSecurityConfigurerAdapter). Must be added to an @Configuration class.
@EnableGlobalMethodSecurity	Class-level annotation that turns on method-level security. Must be on an @Configuration 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 TestExecutionListener classes (rather than all Spring TestExecutionListener 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 UserDetailsService.
@WithSecurityContext	Runs a test with a custom SecurityContext.
@PostFilter	After a method has been called, iterates through a returned collection and removes any item that doesn't match the filter.
@PreFilter	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).
@EnableWebMvcSecurity	Enables Spring Security integration with Spring MVC.
@AuthenticationPrincipal	Automatically resolves the current Authentication.getPrincipal() for Spring MVC arguments.
@EnableAuthorizationServer	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.
@EnableResourceServer	Convenience annotation for OAuth2 Resource Servers, enabling a Spring Security filter that authenticates requests via an incoming OAuth2 token.
@EhableOAuth2Client	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.
@SecuredChannel	Applies the ChannelSecurityInterceptor(s) using provided interceptor() bean name(s).

Spring WebSocket

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

@EnableWebSocket	Enables the processing of WebSocket requests. Must be added to an @Configuration class.
@EnableWebSocketMessageBroker	Enables broker-backed messaging over WebSocket using a higher-level messaging sub-protocol. Must be added to an @Configuration class.
@MessageMapping	Maps a Message onto message-handling methods by matching to the message destination.
@DestinationVariable	Indicates that a method parameter should be bound to a template variable in a destination template string.
@SubscribeMapping	Maps subscription messages onto specific handler methods based on the destination of a subscription.

REST

Spring Data REST provides the following annotations:

@RepositoryRestResource	Marks a repository for custom export mapping and rel attributes.
@Description	Describes the semantics of a resource.
@HandleAfterCreate	Indicates a component that should handle the afterCreate event.
@HandleAfterDelete	Indicates a component that should handle the afterDelete event.
@HandleAfterLinkSave	Indicates a component that should handle the afterLinkSave event.
@HandleAfterSave	Indicates a component that should handle the afterSave event.
@HandleBeforeCreate	Indicates a component that should handle the beforeCreate event.
@HandleBeforeDelete	Indicates a component that should handle the beforeDelete event.

@HandleBeforeLinkDelete	Indicates a component that should handle the beforeLinkDelete event.
@HandleBeforeLinkSave	Indicates a component that should handle the beforeLinkSave event.
@HandleBeforeSave	Indicates a component that should handle the beforeSave event.
@RepositoryEvenHandler	Class-level annotation that indicates that the class is an event handler for a repository.
@RestResource	Indicates how a repository should be exported and what the value of the rel attribute in links will be.
@Projection	Ties a particular projection type to a source type. Used to find projection interfaces at startup time.
@Version	Identifies a property to be used as version field to implement optimistic locking on entities.
@AccessType	Defines how Spring Data accesses values of persistent properties.
@CreatedBy	Declares a field as the one representing the principal that created the entity containing the field.
@CreatedDate	Declares a field as the one representing the date the entity containing the field was created.
@Id	Indicates an identifier.
@LastModifiedBy	Declares a field as the one representing the principal that recently modified the entity containing the field.
@LastModifiedDate	Declares a field as the one representing the date the entity containing the field was recently modified.
@PersistenceConstructor	Indicates that a constructor, even one that's package protected, as the primary constructor used by the persistence logic.

@Persistent	Indicates that a field should be persisted even if there are no getter and setter methods for it.
@QueryAnnotation	Meta-Annotation to mark a store-specific annotation as a query annotation. This allows generic special handing of finder methods on Repository interfaces.
@ReadOnlyProperty	Marks a field as read-only for the mapping framework. The field will not be persisted.
@Reference	Meta-annotation to indicate annotations that mark references to other objects.
@Transient	Marks a field to be transient for the mapping framework.
@TypeAlias	Lets String based type aliases to be used when writing type information for PersistentEntity objects.
@BasePathAwareController	Indicates a controller that declares request mappings to be augmented with a base URI in the Spring Data REST configuration.
@RepositoryRestController	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 {@code @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 {@code @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.

@GatewayHeaer

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.
@RibbonClient	Declares a ribbon client. Must go on an @Configuration class.
@RibbonClients	Convenience annotation that combines multiple @RibbonClient annotations on a single class (including in Java 7).
@EnableZuulProxy	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 DiscoveryClient.
@EnableAtlas	Enables Atlas metrics publishing.
@EnableBinding	Enables the binding of targets annotated with Input and Output to a broker, according to the list of interfaces passed as value to the annotation.
@StreamListener	Indicates a method that is a listener to the inputs declared through @EnableBinding.
@Input	Indicates that an input binding target will be created by the framework.
@Output	Indicates that an output binding target will be created by the framework.
@EnableRxJavaProcessor	Class annotation that identifies the class as an RxJava processor module.
@EnableZipkinStreamServer	Enables transporting of spans over a Spring Cloud Stream (such as RabbitMQ).
@SpanName	Names a span for use with a stream.
@LoadBalanced	Indicates that a RestTemplate bean should be configured to use a LoadBalancerClient.

Data

Spring includes the following annotations:

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

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

	registers HateoasPageableHandlerMethodArgumentResolver, HateoasSortHandlerMethodArgumentResolver, PagedResourcesAssembler, and SortHandlerMethodArgumentResolver.
@PageableDefault	Set defaults when injecting a Pageable into a controller method.
@SortDefault	Defines the default Sort options to be used when injecting a Sort instance into a controller handler method.
@SortDefaults	Wrapper annotation to allow declaring multiple SortDefault annotations on a method parameter.
@JsonPath	Declares a JSON Path expression on a projection interface.
@ProjectedPayload	Enables projection and projection method annotations that contain JSON or XPath expressions.
QuerydslPredicate	Customizes the binding of HTTP request parameters to a Querydsl com.mysema.query.types.Predicate in Spring MVC handler methods.
@Procedure	Declares a JPA 2.1 stored procedure mapping directly on a repository method.
@EnableTransactionManagement	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):

@EnableAspectJAutoProxy	Enables support for handling components marked with AspectJ's @Aspect annotation. Must be used on a class that is also marked with the @Configuration annotation (or another annotation that includes the @Configuration annotation).
@Aspect	Indicates that a class is an aspect.
@Pointcut	Defines a join point.
@target	Limits matching to join points. (Not to be confused with @Target for @Configuration classes.)
@args	Limits matching to join points
@within	Limits matching to join points within types that have the given annotation.
@annotation	limits matching to join points where the subject of the join point has the given annotation
@Transactional	Class annotation that specifies the default transaction semantics for the execution of any public operation in the class.
@Before	Declares pointcut advice that should run before methods matched by the pointcut.

@AfterReturning	Declares pointcut advice that should run after the methods matched by the pointcut. The methods must return normally. See @AfterThrowing and @After.
@AfterThrowing	Declares pointcut advice that should run after the methods matched by the pointcut have thrown an exception.
@After	Declares pointcut advice to run after the methods matched by the pointcut have run, whether they returned normally or threw an exception. Parallel to finally in a try-catch-finally block.
@Around	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.
@DeclareParents	Declares that matching types have a new parent.
@Configurable	Marks a class as eligible for Spring-driven configuration.
@EnableSpringConfigured	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:

@BootstrapWith	Specifies a custom TestContextBootstrapper class.
@ContextConfiguration	Defines class-level metadata that determines how to load and configure an ApplicationContext for integration tests.
@WebAppConfiguration	Class-level annotation that is used to declare that the ApplicationContext loaded for an integration test should be a WebApplicationContext.
@ContextHierarchy	Class-level annotation that defines a hierarchy of ApplicationContexts for integration tests.
@ActiveProfiles	Class-level annotation that indicates which bean definition profiles should be active.

@TestPropertySource	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.
@DirtiesContext	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.
@TestExecutionListeners	Defines class-level metadata for configuring the TestExecutionListener implementations that should be registered with the TestContextManager.
@Commit	Causes a transaction to commit rather than rollback during testing. Use only when you want a test to modify a database. See @Rollback.
@Rollback	indicates whether the transaction for a transactional test method should be rolled back after the test method has completed. See @Commit.
@BeforeTransaction	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 @Transactional annotation.
@AfterTransaction	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 @Transactional annotation.
@Sql	Annotates a test class or test method to configure SQL scripts to be executed against a given database during integration tests.
@SqlConfig	Defines metadata that is used to determine how to parse and execute SQL scripts configured via the @Sql annotation.
@SqlGroup	Container annotation that aggregates several @Sql annotations.

Unit Testing

Spring includes a set of annotations for unit testing:

@IfProfileValue	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.
@ProfileValueSourceConfiguration	Class-level annotation that specifies what type of ProfileValueSource to use when retrieving profile values configured through the @IfProfileValue annotation.
@Timed	Indicates that the annotated test method must finish execution in a specified time period (in milliseconds).
@Repeat	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):

@ManagedResource	Marks all instances of a Class as JMX managed resources.
@ManagedAttribute	Mark a getter or setter as one half of a JMX attribute.
@ManagedOperation	Mark a method as a JMX operation.
@ManagedOperationParameters	Define descriptions for operation parameters.
@ManagedOperationParameter	Define the descriptions for an operation parameter.
@EnableMBeanExport	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:

@Scheduled	Indicates that a method should be called on a scheduled basis.

@Async	Indicates that a method may be called asynchronously.
@EnableScheduling	Enables support for the @Schedule annotation.
@EnableAsync	Enables support for the @Async annotation.

Cache Abstraction

Spring includes a set of annotations for working with caching:

@Cacheable	Indicates a method whose result is cacheable.
@CacheEvict	Indicates that a method performs cache eviction (removes items from a cache).
@CachePut	Indicates a method whose result will always be put in a cache.
@Caching	Allows multiple nested @Cacheable, @CachePut and @CacheEvict to be used on the same method
@CacheConfig	Class-level annotation that allows sharing of the cache names, the custom KeyGenerator, the custom CacheManager, and the custom CacheResolver. Placing this annotation on the class does not turn on any caching operation. See @EnableCaching.
@EnableCaching	Turns on caching in an application. Must go on an @Configuration class.
@CacheResult	Similar to @Cacheable but can cache specific exceptions and force the execution of the method regardless of the content of the cache.
@CacheRemove	Similar to @CacheEvict but can support conditional removal if the method throws an exception.
@CacheRemoveAll	Similar to @CacheEvict(allEntries=true) but can support conditional removal if the method throws an exception.
@CacheDefaults	Similar to @CacheConfig.
@CacheValue	Usable with @CachePut to update the cache before or after method invocation.
@CacheKey	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:

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