**Upgrading to Spring Framework 6.x**

Rossen Stoyanchev edited this page Nov 3, 2023 · [69 revisions](https://github.com/spring-projects/spring-framework/wiki/Upgrading-to-Spring-Framework-6.x/_history)

**Pages 25**

* [Home](https://github.com/spring-projects/spring-framework/wiki)
* [Versions](https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-Versions)
* [Artifacts](https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-Artifacts)
* [Annotations](https://github.com/spring-projects/spring-framework/wiki/Spring-Annotation-Programming-Model)
* [HTTP/2](https://github.com/spring-projects/spring-framework/wiki/HTTP-2-support)

**Clone this wiki locally**

*This page provides guidance on upgrading to Spring Framework 6.x.*

**Upgrading to Version 6.1**

**Baseline upgrades**

Spring Framework 6.1 raises its minimum requirements with the following libraries:

* SnakeYAML 2.0
* Jackson 2.14
* Kotlin Coroutines 1.7
* Kotlin Serialization 1.5

**Removed APIs**

Several deprecated classes, constructors, and methods have been removed across the code base. See [29449](https://github.com/spring-projects/spring-framework/issues/29449) and [30604](https://github.com/spring-projects/spring-framework/issues/30604).

RPC-style remoting that has been officially and/or effectively deprecated for several years has been removed. This impacts Hessian, HTTP Invoker, JMS Invoker, and JAX-WS support, see [27422](https://github.com/spring-projects/spring-framework/issues/27422).

**Core Container**

Aligned with the deprecation of java.net.URL constructors in JDK 20, URL resolution is now consistently performed via URI, including the handling of relative paths. This includes behavioral changes for uncommon cases such as when specifying a full URL as a relative path. See [29481](https://github.com/spring-projects/spring-framework/issues/29481) and [28522](https://github.com/spring-projects/spring-framework/issues/28522).

LocalVariableTableParameterNameDiscoverer has been removed in 6.1. Compile your Java sources with the common Java 8+ -parameters flag for parameter name retention (instead of relying on the -debug compiler flag) in order to be compatible with StandardReflectionParameterNameDiscoverer. With the Kotlin compiler, we recommend the -java-parameters flag.

AutowireCapableBeanFactory.createBean(Class, int, boolean) is deprecated now, in favor of the convention-based createBean(Class). The latter is also consistently used internally in 6.1 – for example, in SpringBeanJobFactory for Quartz and SpringBeanContainer for Hibernate.

Array-to-collection conversion prefers a List result rather than a Set for a declared target type of Collection.

ThreadPoolTaskExecutor and ThreadPoolTaskScheduler enter a graceful shutdown phase when the application context starts to close. As a consequence, further task submissions are not accepted during stop or destroy callbacks in other components anymore. If the latter is necessary, switch the executor/scheduler's acceptTasksAfterContextClose flag to true, at the expense of a longer shutdown phase.

Message resolution through the ApplicationContext (accessing its internal MessageSource) is only allowed while the context is still active. After context close, getMessage attempts will throw an IllegalStateException now.

When building a native image, the verbose logging about pre-computed fields has been removed by default, and can be restored by passing -Dspring.native.precompute.log=verbose as a native-image compiler build argument to display related detailed logs.

**Data Access and Transactions**

@TransactionalEventListener rejects invalid @Transactional usage on the same method: only allowed as REQUIRES\_NEW (possibly in combination with @Async).

JPA bootstrapping now fails in case of an incomplete Hibernate Validator setup (e.g. without an EL provider), making such a scenario easier to debug.

Since JpaTransactionManager with HibernateJpaDialect translates commit/rollback exceptions to DataAccessException subclasses wherever possible, a Hibernate transaction exception formerly propagated as a generic JpaSystemException may show up as e.g. CannotAcquireLockException now. For a non-translatable fallback exception, JpaSystemException will be consistently thrown for commit/rollback now, instead of the former TransactionSystemException propagated from rollback.

**Web Applications**

Spring MVC and WebFlux now have built-in method validation support for controller method parameters with @Constraint annotations. To be in effect, you need to 1) opt out of AOP-based method validation by removing @Validated at the controller class level, 2) ensure mvcValidator or webFluxValidator beans are of type jakarta.validation.Validator (for example, LocalValidatorFactoryBean), and 3) have constraint annotations directly on method parameters. Where method validation is required (i.e. constraint annotations are present), model attribute and request body arguments with @Valid are also validated at the method level, and in that case no longer validated at the argument resolver level, thereby avoiding double validation. BindingResult arguments are still respected, but if not present or if method validation fails on other parameters, then a MethodValidationException raised. That's not handled yet in 6.1 M1, but will be in M2 with [30644](https://github.com/spring-projects/spring-framework/issues/30644). See [29825](https://github.com/spring-projects/spring-framework/issues/29825) for more details on the support in M1, and also the umbrella issue [30645](https://github.com/spring-projects/spring-framework/issues/30645) for all other related tasks and for providing feedback.

The format for MethodArgumentNotValidException and WebExchangeBindException message arguments has changed. Errors are now joined with ", and ", without single quotes and brackets. Field errors are resolved through the MessageSource with nothing further such as the field name added. This gives applications full control over the error format by customizing individual error codes. See [30198](https://github.com/spring-projects/spring-framework/issues/30198) and also planned documentation improvement [30653](https://github.com/spring-projects/spring-framework/issues/30653).

The default order of mappings has been refined to be more consistent by changing RouterFunctionMapping order from 3 to -1 in Spring MVC. That means RouterFunctionMapping is now always ordered before RequestMappingHandlerMapping in both Spring MVC and Spring WebFlux. See [30278](https://github.com/spring-projects/spring-framework/issues/30278) for more details.

The throwExceptionIfNoHandlerFound property of DispatcherHandler is now set to true by default and is deprecated. The resulting exception is handled by default as a 404 error so it should result in the same outcome. Likewise, ResourceHttpRequestHandler now raises NoResourceFoundException, which is also handled by default as a 404, and should have the same outcome for most applications. See [29491](https://github.com/spring-projects/spring-framework/issues/29491).

@RequestParam, @RequestHeader, and other controller method argument annotations now use the defaultValue if the input is a non-empty String without text.

ResponseBodyEmitter now completes the response if the exception is not an IOException, see issue [30687](https://github.com/spring-projects/spring-framework/issues/30687).

Preflight checks are now executed at the start of the HandlerInteceptor chain and not at the end.

The [HTTP interface client](https://docs.spring.io/spring-framework/reference/integration/rest-clients.html#rest-http-interface) no longer enforces a 5 second default timeout on methods with a blocking signature, instead relying on default timeout and configuration settings of the underlying HTTP client. See [30248](https://github.com/spring-projects/spring-framework/issues/30248).

The HTTP server Observability instrumentation in WebFlux was limited and was not properly observing errors. As a result, the WebFlux ServerHttpObservationFilter is now deprecated in favor of direct instrumentation on the WebHttpHandlerBuilder. See [30013](https://github.com/spring-projects/spring-framework/issues/30013).

ReactorResourceFactory class has been moved from the org.springframework.http.client.reactive package to the org.springframework.http.client one.

**Messaging Applications**

The [RSocket interface client](https://docs.spring.io/spring-framework/reference/rsocket.html#rsocket-interface) no longer enforces a 5 second default timeout on methods with a blocking signature, instead relying on default timeout and configuration settings of the RSocket client, and the underlying RSocket transport. See [30248](https://github.com/spring-projects/spring-framework/issues/30248).

In an effort to reduce the potential for security vulnerabilities in the Spring Expression Language (SpEL) to adversely affect Spring applications, the team has decided to disable support for evaluating SpEL expressions from untrusted sources by default. Within the core Spring Framework, this applies to the SpEL-based selector header support in WebSocket messaging, specifically in the DefaultSubscriptionRegistry. The selector header support will remain in place but will have to be explicitly enabled beginning with Spring Framework 6.1 (see [30550](https://github.com/spring-projects/spring-framework/issues/30550)). For example, a custom implementation of WebSocketMessageBrokerConfigurer can override the configureMessageBroker() method and configure the selector header name as follows: registry.enableSimpleBroker().setSelectorHeaderName("selector");.

**Upgrading to Version 6.0**

**Core Container**

The JSR-330 based @Inject annotation is to be found in jakarta.inject now. The corresponding JSR-250 based annotations @PostConstruct and @PreDestroy are to be found in jakarta.annotation. For the time being, Spring keeps detecting their javax equivalents as well, covering common use in pre-compiled binaries.

The core container performs basic bean property determination without java.beans.Introspector by default. For full backwards compatibility with 5.3.x in case of sophisticated JavaBeans usage, specify the following content in a META-INF/spring.factories file which enables 5.3-style full java.beans.Introspector usage: org.springframework.beans.BeanInfoFactory=org.springframework.beans.ExtendedBeanInfoFactory

When staying on 5.3.x for the time being, you may enforce forward compatibility with 6.0-style property determination (and better introspection performance!) through a custom META-INF/spring.factories file: org.springframework.beans.BeanInfoFactory=org.springframework.beans.SimpleBeanInfoFactory

LocalVariableTableParameterNameDiscoverer is deprecated now and logs a warning for each successful resolution attempt (it only kicks in when StandardReflectionParameterNameDiscoverer has not found names). Compile your Java sources with the common Java 8+ -parameters flag for parameter name retention (instead of relying on the -debug compiler flag) in order to avoid that warning, or report it to the maintainers of the affected code. With the Kotlin compiler, we recommend the -java-parameters flag for completeness.

LocalValidatorFactoryBean relies on standard parameter name resolution in Bean Validation 3.0 now, just configuring additional Kotlin reflection if Kotlin is present. If you refer to parameter names in your Bean Validation setup, make sure to compile your Java sources with the Java 8+ -parameters flag.

ListenableFuture has been deprecated in favor of CompletableFuture. See [27780](https://github.com/spring-projects/spring-framework/issues/27780).

Methods annotated with @Async must return either Future or void. This has long been documented but is now also actively checked and enforced, with an exception thrown for any other return type. See [27734](https://github.com/spring-projects/spring-framework/issues/27734).

SimpleEvaluationContext disables array allocations now, aligned with regular constructor resolution.

**Caching**

The org.springframework.cache.ehcache package has been removed as it was providing support for ehcache 2.x - with this version, net.sf.ehcache is using JavaEE APIs and [is about to be End Of Life'd](https://github.com/ehcache/ehcache2). Ehcache3 is the direct replacement. You should revisit your dependency management to use org.ehcache:ehcache (with the jakarta classifier) instead and look [into the official migration guide or reach out to the ehcache community for assistance](https://www.ehcache.org/documentation/3.10/migration-guide.html). We did not replace org.springframework.cache.ehcache with an updated version, as using ehcache through the JCache API or its new native API is preferred.

**Data Access and Transactions**

Due to the Jakarta EE migration, make sure to upgrade to Hibernate ORM 5.6.x with the hibernate-core-jakarta artifact, alongside switching your javax.persistence imports to jakarta.persistence (Jakarta EE 9). Alternatively, consider migrating to Hibernate ORM 6.1 right away (exclusively based on jakarta.persistence, compatible with EE 9 as well as EE 10) which is the Hibernate version that Spring Boot 3.0 comes with.

The corresponding Hibernate Validator generation is 7.0.x, based on jakarta.validation (Jakarta EE 9). You may also choose to upgrade to Hibernate Validator 8.0 right away (aligned with Jakarta EE 10).

For EclipseLink as the persistence provider of choice, the reference version is 3.0.x (Jakarta EE 9), with EclipseLink 4.0 as the most recent supported version (Jakarta EE 10).

Spring's default JDBC exception translator is the JDBC 4 based SQLExceptionSubclassTranslator now, detecting JDBC driver subclasses as well as common SQL state indications (without database product name resolution at runtime). As of 6.0.3, this includes a common SQL state check for DuplicateKeyException, addressing a long-standing difference between SQL state mappings and legacy default error code mappings.

CannotSerializeTransactionException and DeadlockLoserDataAccessException are deprecated as of 6.0.3 due to their inconsistent JDBC semantics, in favor of the PessimisticLockingFailureException base class and consistent semantics of its common CannotAcquireLockException subclass (aligned with JPA/Hibernate) in all default exception translation scenarios.

For full backwards compatibility with database-specific error codes, consider re-enabling the legacy SQLErrorCodeSQLExceptionTranslator. This translator kicks in for user-provided sql-error-codes.xml files. It can simply pick up Spring's legacy default error code mappings as well when triggered by an empty user-provided file in the root of the classpath.

**Web Applications**

Due to the Jakarta EE migration, make sure to upgrade to Tomcat 10, Jetty 11, or Undertow 2.2.19 with the undertow-servlet-jakarta artifact, alongside switching your javax.servlet imports to jakarta.servlet (Jakarta EE 9). For the latest server generations, consider Tomcat 10.1 and Undertow 2.3 (Jakarta EE 10).

Several outdated Servlet-based integrations have been dropped: e.g. Apache Commons FileUpload (org.springframework.web.multipart.commons.CommonsMultipartResolver), and Apache Tiles as well as FreeMarker JSP support in the corresponding org.springframework.web.servlet.view subpackages. We recommend org.springframework.web.multipart.support.StandardServletMultipartResolver for multipart file uploads and regular FreeMarker template views if needed, and a general focus on REST-oriented web architectures.

Spring MVC and Spring WebFlux no longer detect controllers based solely on a type-level @RequestMapping annotation. That means interface-based AOP proxying for web controllers may no longer work. Please, enable class-based proxying for such controllers; otherwise the interface must also be annotated with @Controller. See [22154](https://github.com/spring-projects/spring-framework/issues/22154).

HttpMethod is now a class and no longer an enum. Though the public API has been maintained, some migration might be necessary (i.e. change from EnumSet<HttpMethod> to Set<HttpMethod>, use if else instead of switch). For the rationale behind this decision, see [27697](https://github.com/spring-projects/spring-framework/issues/27697).

The Kotlin extension function to WebTestClient.ResponseSpec::expectBody now returns the Java BodySpec type and no longer uses the workaround type KotlinBodySpec. Spring 6.0 uses Kotlin 1.6, which fixed the bug that needed this workaround ([KT-5464](https://youtrack.jetbrains.com/issue/KT-5464)). This means that consumeWith is no longer available.

RestTemplate, or rather the HttpComponentsClientHttpRequestFactory, now requires Apache HttpClient 5.

The Spring-provided Servlet mocks (MockHttpServletRequest, MockHttpSession) require Servlet 6.0 now, due to a breaking change between the Servlet 5.0 and 6.0 API jars. They can be used for testing Servlet 5.0 based code but need to run against the Servlet 6.0 API (or newer) on the test classpath. Note that your production code may still compile against Servlet 5.0 and get integration-tested with Servlet 5.0 based containers; just mock-based tests need to run against the Servlet 6.0 API jar.

SourceHttpMessageConverter is not configured by default anymore in Spring MVC and RestTemplate. As a consequence, Spring web applications using javax.xml.transform.Source now need to configure SourceHttpMessageConverter explicitly. Note that the order of converter registration is important, and SourceHttpMessageConverter should typically be registered before "catch-all" converters like MappingJackson2HttpMessageConverter for example.