This page provides guidance on upgrading to Spring Framework 5.0, 5.1, 5.2, and 5.3. See also the [[Spring-Framework-5-FAQ]] and [[What's New in Spring Framework 5.x]].

Note that Spring Framework 4.3.x and therefore Spring Framework 4 overall reached its EOL cut-off on December 31st, 2020, along with the 5.0.x and 5.1.x lines. Please upgrade to Spring Framework 5.2+ at your earliest convenience!

Upgrading to Version 5.3

Third-Party APIs and Libraries

For Kotlin: * Kotlin support has been upgraded to Kotlin 1.4 and is still compatible with Kotlin 1.3+. * Kotlin Coroutines 1.4 (which builds on Kotlin 1.4) or above is now required for coroutines support. * For Kotlin scripting, Kotlin 1.4 users should declare the `kotlin-scripting-jsr223` dependency instead of `kotlin-scripting-jsr223-embeddable`.

Spring Framework 5.3 ships with a WildFly manifest that makes Objenesis work on JDK 9+. This is known to cause an incompatibility with WildFly 9; please upgrade to a more recent version of WildFly - or patch your copy of `spring-core.jar` to drop the `Dependencies` manifest entry.

Hibernate support has been upgraded to a Hibernate ORM 5.2+ baseline, with a focus on ORM 5.4.x. Please note that Hibernate Search needs to be upgraded to 5.11.6 for Spring Framework 5.3 JPA compatibility; see Hibernate JIRA.

Jackson support covers Jackson 2.9 to 2.12 now, generally with the latest releases in each branch.

Groovy 3.0 is the officially supported version now, with Groovy 2.x support getting phased out.

Support for RxJava 1.x is deprecated; RxJava 2.x is the new baseline and 3.x is now supported.

JCA CCI support is deprecated, in favor of specific data access APIs (or native CCI usage if there is no alternative).

Several remoting technologies have been deprecated with no direct replacement (Hessian, RMI, HTTP Invoker, JMS Invoker).

`MimeMessageHelper` has been aligned with JavaMail 1.5+, not explicitly encoding attachment filenames by default anymore.

Core Container

The properties-based bean definition format and all support classes based on it (such as `PropertiesBeanDefinitionReader`, `JdbcBeanDefinitionReader` and `ResourceBundleViewResolver`) are deprecated now, in favor of Spring's common bean definition formats and/or custom reader implementations.

`InstantiationAwareBeanPostProcessorAdapter` is deprecated now, in favor of the existing default methods in `(Smart)InstantiationAwareBeanPostProcessor`.

`BeanNameAutoProxyCreator` now honors the configured `beanNames` list when applying a custom `TargetSourceCreator`. Consequently, a `BeanNameAutoProxyCreator` no longer proxies beans whose names do not match the configured `beanNames` list. See gh-24915.

`@EventListener` methods use an implicit order value of `Ordered.LOWEST_PRECEDENCE` now, in alignment with transaction synchronizations and `@TransactionalEventListener` methods. If custom ordering is needed, please consistently declare `@Order` values on all listener methods.

Caching within the core container is consistently optimized for bean definitions with stable bean types per bean name, even for prototype beans. This is partially also the case for the 5.2.x line. Performance regressions might require some bean definition redesign; see e.g. gh-26369.

Data Access and Transactions

Several `JdbcTemplate` signatures with `Object[]` arguments are deprecated, in favor of their existing varargs equivalents.

`HibernateJpaVendorAdapter` exposes `Session(Factory)` as `EntityManager(Factory)` extension interface by default (following Hibernate ORM 5.2+).

`TransactionSynchronization` extends the `Ordered` interface by default now, as a replacement for the deprecated `TransactionSynchronizationAdapter`. As a consequence, `@Order` annotations on synchronization instances - which were not officially supported despite effectively working since 4.2 - do not work anymore. Synchronization objects are low-level callback objects, please express any attached order with a programmatic `getOrder` implementation.

Reactive transactions consistently roll back on a Reactive Streams cancel signal now, preventing partial commits for common datastore transactions.

Web Applications

CORS configuration where `allowCredentials` is set to true now requires an explicit declaration of specific domains to allow via `allowedOrigins` or use of the newly added `allowedOriginPatterns`.

When using the `PathPatternParser` for request mapping, patterns with double-wildards in the middle of the pattern, such as `"/path/**/other"`, are now rejected. This parser is used by default in WebFlux applications and can be used as an opt-in for MVC applications as of Spring Framework 5.3. See gh-24952.

The `ForwardedHeaderFilter` (Servlet) and `ForwardedHeaderTransformer` (WebFlux) have been enhanced and now support multiple values in

X-Forwarded-Prefix and the new X-Forwarded-For / Forwarded: for= HTTP request headers. See gh-25254 and gh-23582.

@ExceptionHandler methods now check all exception causes when looking for a match. Previously, going back to 4.3 only the first cause was checked.

@RequestParam, @RequestHeader, and other controller method argument annotations that depend on type conversion from String values to other types such as UUID, Long, and others, now detect a null conversion result value and treat as missing. For example, a query parameter with an empty value is now treated as missing if it requires type conversion and the conversion results in null. In order to allow an empty value to be injected as a null argument, either set required=false on the argument annotation, e.g. @RequestParam(required=false), or declare the argument as @Nullable.

WebSocketConfigurationSupport and WebSocketMessageBrokerConfigurationSupport have been refactored to not require CGLIB proxies, see related commit.

Spring MVC

LocaleResolver, ThemeResolver, FlashMapManager and RequestToViewNameTranslator beans are now declared at WebMvcConfigurationSupport and DelegatingWebMvcConfiguration level with @Bean annotations for improved consistency with other Spring MVC default beans and to improve GraalVM compatibility by reducing reflection done otherwise in DispatcherServlet. Spring Boot or XML application context based projects shouldn't be impacted, but non Spring Boot projects using JavaConfig overriding one of those default beans may require an update since the bean declaration should now happen in the configuration class annotated with @EnableWebMvc (still using their well-known names), for example for the LocaleResolver:

```
@Configuration
public class WebConfig extends DelegatingWebMvcConfiguration {

    @Bean
    @Override
    public LocaleResolver localeResolver() {
        return new FixedLocaleResolver(new Locale("fr", "FR"));
    }
}
```

@RequestParam and @RequestPart enforce at least one element in a MultipartFile and Servlet Part collection/array when the argument is required (i.e. not explicitly marked as optional), consistent with individual MultipartFile/Part declarations, resolving the argument to null otherwise.

Spring MVC no longer performs .* suffix pattern matching by default, and likewise path extensions are no longer used by default to interpret the requested content type (e.g. `/person.pdf`, `/person.xml`, etc). Please, see the "Suffix Match" section of the reference documentation.

In case of JSON serialization errors while writing to the HTTP response, the JSON `HttpMessageConverter` will still flush to the response and clean up resources. If you were previously relying on the response not being written to, note that this was not an intended behavior and that we can't guarantee error handling for this case. See gh-26246 for an example of that.

A `java.security.Principal` argument is no longer resolved eagerly, if it is annotated in some way such as `@AuthenticationPrincipal`, thus allowing a custom resolver to resolve it first, before using default resolution resolution via `HttpServletRequest#getUserPrincipal`. This can cause issues for existing applications that are trying to inject the Spring Security `Authentication` but also have it annotated with `@AuthenticationPrincipal` which now results in the injection of `Authentication#getPrincipal` as per the intent for the annotation. Removing `@AuthenticationPrincipal` results in the injection of the top level `Authentication` object which is also a `Principal` and would be resolved via `HttpServletRequest#getUserPrincipal` after the change.

Spring WebFlux

`@RequestPart` with `List<T>` now converts the 1st part to `List<T>` consistent with Spring MVC and with how it works for `T[]`. Previously each part was converted to `T`, see gh-22973.

`@EnableWebFlux` now includes a declaration of a `WebSocketHandlerAdapter` bean. It should not interfere with any declared by the application due to a lower priority but if you have one, you can remove it.

WebClient now wraps emitted exceptions in either a `WebClientRequestException`, or a `WebClientResponseException`. `CodecExceptions` are not wrapped and still propagated. as before. See gh-23842.

A `UriComponentsBuilder` argument injected into an `@Controller` method is now application relative (i.e. it includes the contextPath) where previously it contained no path at all.

`org.synchronoss.cloud:nio-multipart-parser` is no longer a required dependency for multipart support in WebFlux. Instead, there is now the `DefaultPartHttpMessageReader` with no dependencies, see gh-21659

Testing

The *Spring TestContext Framework* now provides first-class support for inheriting and overriding test-related annotations from enclosing classes. This improves the programming model for using JUnit Jupiter `@Nested` test classes with Spring's testing support. Note, however, that annotations from enclosing classes will now be inherited by default. This is a change in behavior that may cause some of your `@Nested` test classes to fail after upgrading to Spring Framework 5.3. To revert to the behavior present in Spring Framework 5.0 - 5.2.x, you can annotate top-level, enclosing classes for your `@Nested` test classes with `@NestedTestConfiguration(OVERRIDE)`. To switch to `OVERRIDE` mode for an entire project, you can configure `spring.test.enclosing.configuration=override` via a JVM system property or an entry in a `spring.properties` file in the root of the classpath (for example, in `src/test/resources/spring.properties`).

Consult the Javadoc for `@NestedTestConfiguration` and the reference manual for details.

Users of `MockMvc` Kotlin DSL could experience some breakages when using property syntax like `isOk`, `isOk()` should be used instead. There maybe be some other slight variations due to the improved Kotlin DSL, see this commit for more details.

Upgrading to Version 5.2

Libraries

Spring Framework 5.2 now requires Jackson 2.9.7+ and explicitly supports the recently released Jackson 2.10 GA. See gh-23522.

In Reactor Core 3.3, the Kotlin extensions are deprecated and replaced by a dedicated reactor-kotlin-extensions project/repo. You may have to add `io.projectreactor.kotlin:reactor-kotlin-extensions` dependency to your project and update related packages to use the non-deprecated variants.

Core Container

Spring's annotation retrieval algorithms have been completely revised for efficiency and consistency, as well as for potential optimizations through annotation presence hints (e.g. from a compile-time index). This may have side effects – for example, finding annotations in places where they haven't been found before or not finding annotations anymore where they have previously been found accidentally. While we don't expect common Spring applications to be affected, annotation declaration accidents in application code may get uncovered when you upgrade to 5.2. For example, all annotations must now be annotated with `@Retention(RetentionPolicy.RUNTIME)` in order for Spring to find them. See gh-23901, gh-22886, and gh-22766.

Web Applications

`@RequestMapping` without path attribute `@RequestMapping()` and meta-annotated variants `@GetMapping()`, `PostMapping()`, etc., without explicitly declared `path` patterns are now equivalent to `RequestMapping("")` and match only to URLs with no path. In the absence of declared patterns previously the path was not checked thereby matching to any path. If you would like to match to all paths, please use `"/**"` as the pattern. gh-22543

`@EnableWebMvc` and `@EnableWebFlux` Infrastructure `@Bean` methods in `Web**ConfigurationSupport` now declare bean dependencies as method arguments rather than use method calls to make it possible to avoid creating proxies for bean methods via `@Configuration(proxyBeanMethods=false)` which Spring Boot 2.2 now does. This should not affect existing applications but if subclassing `Web**ConfigurationSupport` (or `DelegatingWeb**Configuration`) and using `proxyBeanMethods=false` be sure to also to declare dependent beans as method arguments rather than using method calls. See gh-22596

Deprecation of `MediaType.APPLICATION_JSON_UTF8` and `MediaType.APPLICATION_PROBLEM_JSON_UTF8` Since the related Chrome bug is now fixed since September 2017, Spring Framework 5.2 deprecates `MediaType.APPLICATION_JSON_UTF8` and `MediaType.APPLICATION_PROBLEM_JSON_UTF8` in favor of `MediaType.APPLICATION_JSON` and `MediaType.APPLICATION_PROBLEM_JSON` and uses them by default. As a consequence, integration tests relying on the default JSON content type may have to be updated. See gh-22788 for more details.

CORS handling CORS handling has been significantly updated in Spring Framework 5.2: - CORS processing is now only used for CORS-enabled endpoints - CORS processing for skipped for same-origin requests with an `Origin` header - Vary headers are added for non-CORS requests on CORS endpoints

These changes introduce an `AbstractHandlerMapping#hasCorsConfigurationSource` method (in both Spring MVC and WebFlux) in order to be able to check CORS endpoints efficiently. When upgrading to Spring Framework 5.2, handler mapping extending `AbstractHandlerMapping` and supporting CORS should override `hasCorsConfigurationSource` with their custom logic.

Use of Path Extensions Deprecated in Spring MVC Config options for suffix pattern matching in `RequestMappingHandlerMapping` have been deprecated, and likewise config options to resolve acceptable media types from the extension of a request path in `ContentNegotiationManagerFactoryBean` have also been deprecated. This is aligned with defaults in Spring Boot auto configuration and Spring WebFlux does not offer such options. See gh-24179 and related issues for details and further plans towards 5.3.

`Encoder` Contract Custom implementations of `Encoder` must implement the new encodeValue which is invoked from `ServerSentEventHttpMessageWriter` or otherwise that would fail at runtime.

Testing

The mock JNDI support in the `spring-test` module has been deprecated. If you have been using classes such as the `SimpleNamingContext` and `SimpleNamingContextBuilder`, you are encouraged to migrate to a complete JNDI solution from a third party such as Simple-JNDI. [gh-22779]

Upgrading to Version 5.1

JDK 11

Spring Framework 5.1 requires JDK 8 or higher and specifically supports JDK 11 (as the next long-term support release) for the first time. We strongly recommend an upgrade to Spring Framework 5.1 for any applications targeting JDK 11, delivering a warning-free experience on the classpath as well as the module path.

Please note that developing against JDK 11 is not officially supported with any older version of the core framework. Spring Framework 5.0.9+ and 4.3.19+ just support deploying Java 8 based applications to a JVM 11 runtime environment (using `-target 1.8`; see below), accepting JVM-level warnings on startup.

ASM Spring Framework 5.1 uses a patched ASM 7.0 fork which is prepared for JDK 11/12 and their new bytecode levels but not battle-tested yet. Spring Framework 5.1.x will track further ASM revisions on the way to JDK 12, also hardening bytecode compatibility with JDK 11.

For a defensive upgrade strategy, consider compiling your application code with JDK 8 as a target (`-target 1.8`), simply deploying it to JDK 11. This makes your bytecode safer to parse not only for Spring's classpath scanning but also for other bytecode processing tools.

CGLIB Spring Framework 5.1 uses a patched CGLIB 3.2 fork that delegates to JDK 9+ API for defining classes at runtime. Since this code path is only active when actually running on JDK 9 or higher (in particular necessary on JDK 11 where an alternative API for defining classes has been removed), side effects might show up when upgrading existing applications to JDK 11.

Spring has a fallback in place which tries to mitigate class definition issues, possibly leading to a JVM warning being logged, whereas the standard code path delivers a warning-free experience on JDK 11 for regular class definition purposes. Consider revisiting your class definitions and bytecode processing tools in such a scenario, upgrading them to JDK 11 policies.

Core Container

The core container has been fine-tuned for Graal compatibility (native images on Substrate VM) and generally optimized for less startup overhead and less garbage collection pressure. As part of this effort, several introspection algorithms have been streamlined towards avoiding unnecessary reflection steps, potentially causing side effects for annotations declared outside of well-defined places.

Nested Configuration Class Detection As per their original definition, nested configuration classes are only detected on top-level `@Configuration` or other `@Component`-stereotyped classes now, not on plain usage of `@Import` or `@ComponentScan`. Older versions of Spring over-introspected nested classes on non-stereotyped classes, causing significant startup overhead in some scenarios.

In case of accidentally relying on nested class detection on plain classes, simply declare those containing classes with a configuration/component stereotype.

Web Applications

Forwarded Headers `"Forwarded"` and `"X-Forwarded-*"` headers, which reflect the client's original address, are no longer checked individually in places where they apply, e.g. same origin CORS checks, `MvcUriComponentsBuilder`, etc.

Applications are expected to use one of: * Spring Framework's own `ForwardedHeaderFilter`. * Support for forwarded headers from the HTTP server or proxy.

Note that `ForwardedHeaderFilter` can be configured in a safe mode where it checks and discards such headers so they cannot impact the application.

Encoding Mode of `DefaultUriBuilderFactory` The encoding mode of `DefaultUriBuilderFactory` has been switched to enforce stricter encoding of URI variables by default. This could impact any application using the `WebClient` with default settings, or any application using `DefaultUriBuilderFactory` directly. See the "Encoding URIs" section and also the Javadoc for `DefaultUriBuilderFactory#setEncodingMode`.

Content Negotiation for Error Responses The produces condition of an `@RequestMapping` no longer impacts the content type of error responses.

Multipart and Query Values The integration with Apache Commons FileUpload now aggregates multipart parameter values with other request parameters from the query, as required by Servlet spec, section 3.1. Previously it returned only multipart parameter values if present.

HTTP OPTIONS The built-in support for HTTP OPTIONS in `@RequestMapping` methods now consistently adds HTTP OPTIONS as one of the supported HTTP methods, whereas previously it did not.

Upgrading to Version 5.0

Baseline update

Spring Framework 5.0 requires JDK 8 (Java SE 8), since its entire codebase is based on Java 8 source code level and provides full compatibility with JDK 9 on the classpath as well as the module path (Jigsaw). Java SE 8 update 60 is suggested as the minimum patch release for Java 8, but it is generally recommended to use a recent patch release.

The Java EE 7 API level is required in Spring's corresponding modules now, with runtime support for the EE 8 level:

- Servlet 3.1 / 4.0
- JPA 2.1 / 2.2
- Bean Validation 1.1 / 2.0
- JMS 2.0
- JSON Binding API 1.0 (as an alternative to Jackson / Gson)

Web Servers

- Tomcat 8.5+
- Jetty 9.4+
- WildFly 10+
- WebSphere 9+
- with the addition of Netty 4.1 and Undertow 1.4 for Spring WebFlux

Libraries

- Jackson 2.9+
- EhCache 2.10+
- Hibernate 5.0+
- OkHttp 3.0+
- XmlUnit 2.0+
- XStream 1.4.7+

Removed Packages, Classes and Methods

- Package beans.factory.access (BeanFactoryLocator mechanism).
  - Including `SpringBeanAutowiringInterceptor` for EJB3 which was based on such a statically shared context. Preferably integrate a Spring backend via CDI instead.
- Package jdbc.support.nativejdbc (NativeJdbcExtractor mechanism).
  - Superseded by the native `Connection.unwrap` mechanism in JDBC 4. There is generally very little need for unwrapping these days, even against the Oracle JDBC driver.
- Package `mock.staticmock` removed from `spring-aspects` module.
  - No support for `AnnotationDrivenStaticEntityMockingControl` anymore.

- Packages `web.view.tiles2` and `orm.hibernate3/hibernate4` dropped.
  - Minimum requirement: Tiles 3 and Hibernate 5 now.
- Many deprecated classes and methods removed across the codebase.
  - A few compromises made for commonly used methods in the ecosystem.
- Note that several deprecated methods have been removed from the JSP tag library as well.
  - e.g. FormTag's "commandName" attribute, superseded by "modelAttribute" years ago.

Dropped support

The Spring Framework no longer supports: Portlet, Velocity, JasperReports, XMLBeans, JDO, Guava (replaced by the Caffeine support). If those are critical to your project, you should stay on Spring Framework 4.3.x (supported until 2020). Alternatively, you may create custom adapter classes in your own project (possibly derived from Spring Framework 4.x).

Commons Logging setup

Spring Framework 5.0 comes with its own Commons Logging bridge in the form of the 'spring-jcl' module that 'spring-core' depends on. This replaces the former dependency on the 'commons-logging' artifact which required an exclude declaration for switching to 'jcl-over-slf4j' (SLF4J / Logback) and an extra bridge declaration for 'log4j-jcl' (Log4j 2.x).

Now, 'spring-jcl' itself is a very capable Commons Logging bridge with first-class support for Log4j 2, SLF4J and JUL (java.util.logging), working out of the box without any special excludes or bridge declarations for all three scenarios.

You may still exclude 'spring-jcl' from 'spring-core' and bring in 'jcl-over-slf4j' as your choice, in particular for upgrading an existing project. However, please note that 'spring-jcl' can easily supersede 'jcl-over-slf4j' by default for a streamlined Maven dependency setup, reacting to the plain presence of the Log4j 2.x / Logback core jars at runtime.

Please note: For a clean classpath arrangement (without several variants of Commons Logging on the classpath), you might have to declare explicit excludes for 'commons-logging' and/or 'jcl-over-slf4j' in other libraries that you're using.

CORS support

CORS support has been updated to be more secured by default and more flexible.

When upgrading, be aware that `allowCredentials` default value has been changed to `false` and now requires to be explicitly set to `true` if cookies or authentication are needed in CORS requests. This can be done at controller level via `@CrossOrigin(allowCredentials="true")` or configured globally via `WebMvcConfigurer#addCorsMappings`.

CORS configuration combination logic has also been slightly modified to differentiate user defined `*` values where additive logic should be used and default `@CrossOrigin` values which should be replaced by any user provided values.