If you are still on an older 4.x branch (even 4.2.x reached its end of support in the meantime), we strongly recommend an upgrade to the latest Spring Framework 4.3.x release for continued production support.

The 4.3.x line will enjoy an extended support life until 2020, within the general Spring Framework 4 system requirements (JDK 6+, Servlet 2.5+) but with a focus on recent servers such as Tomcat 8.5+ and WebSphere 8.5+ on JDK 7 and 8. If you can deploy to a Servlet 3.1+ baseline on JDK 8, you may also consider an upgrade straight to Spring Framework 5.

Upgrading to Version 4.3

For an overview of new features, refer to New Features and Enhancements in Spring Framework 4.3 in the reference documentation.

Third-party dependencies

Spring 4.3 supports all current versions of its optionally integrated libraries, including *Hibernate ORM 5.2* and *Jackson 2.8/2.9* as well as *OkHttp 3.x*. Furthermore, it embeds the updated *ASM 5.2* and *Objenesis 2.6*.

Please note that several minimum dependency versions have been raised: *Jetty 9.1+*, *Jackson 2.6+*, *FreeMarker 2.3.21+*, *XStream 1.4.5+*. Spring's support for *Hibernate 3.x* and *Velocity* has been deprecated and scheduled for removal in 5.0.

Default handling of HEAD and OPTIONS requests

As of 4.3, Spring MVC processes HEAD and OPTIONS requests by default if there are no explicit bindings for those HTTP methods on a given path, along the lines of what HttpServlet does by default. While this should be a reasonable enhancement to all common Spring web applications, there may be subtle interaction side effects.

Injection points declared as Collection, Map or array

Spring 4.3 refines autowiring support for beans which are declared as a Collection, Map or array type, matching them by type against injection points of a compatible type directly. This support lives side by side with Spring's existing support for finding beans by the element/value type of a declared Collection, Map or array injection point. There might be subtle side effects if custom application context arrangements rely on Collection/Map beans not matching directly.

CORS with Credentials

The `allowCredentials` flag has been changed from `true` to `false` by default to align with similar behavior change previously made in the 5.x branch. This can be re-enabled by setting it explicitly, e.g. via `@CrossOrigin` annotations.

Upgrading to Version 4.2

For an overview of new features, refer to New Features and Enhancements in Spring Framework 4.2 in the reference documentation.

Third-party dependencies

Spring's Reactor support is based on *Reactor 2.0* now. Spring 4.2 also introduces support for *Hibernate ORM 5.0* next to the existing support for Hibernate ORM 3.6 and 4.2 / 4.3. Note that support for Apache Tiles 2.2 is deprecated in the meantime, in favor of Tiles 3.0.

All other dependency ranges in 4.2 remain the same as with Spring Framework 4.1.x, with - as usual - the latest minor versions of all common frameworks being fully supported now: e.g. Hibernate Validator 5.2, Jackson 2.6, Undertow 1.2, Apache HttpComponents 4.5.

Modern web interaction defaults

Note that Spring Framework 4.2 comes with revised HTTP cache header processing for resource handling; for common use cases, HTTP caching related headers now have new defaults which do not involve HTTP 1.0 headers such as `"Expires"` or `"Pragma"`. See WebContentGenerator.setCacheSeconds and the new CacheControl class for more details.

Also, Spring Framework now enforces a different default mode for HTML escaping: namely, taking the response encoding into account and therefore reducing the effort to basic XML character escaping in case of UTF-* encodings (since all other characters can be natively represented in UTF anyway). This can be overridden through setting the "responseEncodedHtmlEscape" context-param to "false", restoring the previous default behavior of full HTML character escaping in any case.

Configuration class ordering

Spring Framework 4.2 comes with significant fine-tuning in configuration class processing. There may be subtle differences in the order of registration compared to 4.1; however, those are considered fixes of behavior that wasn't well-defined previously. If you are relying on a specific order, e.g. for overriding beans by name, please consider using 4.2's new facilities, in particular @Order annotations on config classes.

Upgrading to Version 4.1

For an overview of new features, refer to New Features and Enhancements in Spring Framework 4.1 in the reference documentation.

In general, there are no special migration steps from 4.0 to 4.1. Make sure that you satisfy all of 4.0's requirements as stated in the 4.0 section below. 4.1 will then just be a minor upgrade, with the most noticeable part being the enforcement of several optional dependency versions...

Enforced minimum dependency versions

As of Spring Framework 4.1, several optional dependencies are required to be within the supported range: in particular, *EhCache 2.5+, Quartz 2.1.4+, Jackson 2.1+, and ROME 1.5+* (see below for details about 4.x dependency declarations in the Spring Framework 4.0 section). Older versions have still been tolerated in the Spring Framework 4.0.x line for a smoother upgrade path; as of 4.1, the intended minimum versions are being enforced. *Most importantly, don't use Quartz 1.8.x anymore; upgrade to Quartz 2.1.4+ instead!*

Note that as of Spring Framework 4.1.4, Apache HttpComponents HttpClient needs to be 4.3+ across the framework. The previous partial tolerance of old HttpClient versions had to be dropped in order to fix several configuration issues when running against 4.3+.

Upgrading to Version 4.0

For a general overview of new features, refer to New Features and Enhancements in Spring Framework 4.0 in the reference documentation.

JDK 6

Spring Framework 4.0 requires Java SE 6 or above: specifically, a minimum API level equivalent to JDK 6 update 18, a.k.a. 1.6.0_18, as released in January 2010. You have to update to a recent version of JDK 6 at least: From a known

bug-fix level, we require a minimum of JDK 6 update 21, and we suggest JDK 6 update 25 or higher from a support perspective. Java 7 and 8 are recommended for use with Spring Framework 4.0, with Java 8 supported in production since Oracle's JDK 8 launch in March 2014.

A note on the IBM JDK

Spring generally supports equivalent generations of the IBM JDK and JRE, in particular as shipped with the WebSphere application server. Since not only WebSphere 7 but also WebSphere 8 and 8.5 have been shipping with IBM JDK 6 as the default JDK up until 2013, we intend to retain our JDK 6 support for the entire Spring Framework 4.x line, as long as those generations of WebSphere are supported by IBM themselves. At the same time, if available to you, we recommend IBM JDK 7 with WebSphere 8.5.

Java EE 6

If you deploy your Spring applications to Java EE servers, we recommend server generations certified for Java EE 6. Of particular importance are the JPA 2.0 and Servlet 3.0 specifications. That said, it is still possible to deploy Spring 4 applications to servers with a Servlet 2.5 container (e.g. Google App Engine, WebSphere 7, WebLogic 10.3); however, some Servlet 3.0 based Spring features won't be available then. For Tomcat, the same rules apply: We recommend Tomcat 7 or 8 but still support Tomcat 6 as well.

A note on Java EE levels

The Spring Framework generally doesn't require a specific level of Java EE overall but rather specific minimum levels of individual specifications, such as JPA 2.0. This approach allows for running on "intermediate" server generations which selectively introduce new specifications while being based on an older EE platform level: e.g. WebSphere 7.0.0.9 with its JPA 2.0 feature pack, WebLogic 10.3.4 with its JPA 2.0 patch, or now WebLogic 12.1.3 (as released in June 2014) with its JPA 2.1 support and WebSphere 8.5 Liberty Profile (December 2014 edition) with its Servlet 3.1 and JSR-236 Concurrency support on an EE 6 baseline.

Dependency updates

As of Spring Framework 4.0.3, we declare the following minimum (optional) dependencies as officially supported. Those versions and later are what the Spring team recommends, in particular with respect to providing support for Spring applications that interact with those servers and libraries. Note that earlier versions may still work with Spring to some degree, as long as the fundamental system requirements are met; however, when using older versions, you are on your own from a support perspective.

Specifications

- Servlet 3.0 *(2.5 supported for deployment)*
- JPA 2.0
- Bean Validation 1.0
- JSF 2.0
- JCache 1.0
- JDO 3.0

Servers

- Tomcat 6.0.33 / 7.0.20 / 8.0.9
- Jetty 7.5
- JBoss AS 6.1 *(note: JBoss EAP 6 recommended, since AS 6/7 community releases have many unresolved bugs)*
- GlassFish 3.1 *(note: deployment workaround necessary for non-serializable session attributes)*
- Oracle WebLogic 10.3.4 *(with JPA 2.0 patch applied)*
- IBM WebSphere 7.0.0.9 *(with JPA 2.0 feature pack installed)*

Libraries

- Hibernate Validator 4.3
- Hibernate ORM 3.6.10 *(note: phasing out as of Spring Framework 4.2, with Hibernate 4.2/4.3 recommended)*
- Apache Tiles 2.2.2 *(note: to be deprecated as of Spring Framework 4.2, with Tiles 3.0.5 recommended)*
- Apache HttpComponents 4.3 *(required for Spring's http.client package, and for all of Spring as of 4.1.4)*
- EhCache 2.4.7 *(note: minimum 2.5 as of Spring Framework 4.1, with EhCache 2.8 or later recommended)*
- Quartz 1.8.6 *(note: minimum 2.1.4 as of Spring Framework 4.1, with Quartz 2.2.1 recommended)*
- Jackson 1.8.6 *(note: minimum 2.1 as of Spring Framework 4.1, with Jackson 2.3 or later recommended)*
- Rome 1.0 *(note: minimum 1.5 as of Spring Framework 4.1, dependency group id has also changed from "rome" to "com.rometools")*
- Groovy 1.8.6 *(note: 2.3 or later recommended)*
- Joda-Time 2.1 *(note: 2.3 or later recommended)*
- Hessian 4.0.33
- XStream 1.4
- Apache Velocity 1.7
- Apache POI 3.8
- Apache Derby 10.8
- JUnit 4.7 *(note: minimum 4.9 as of Spring Framework 4.1, with JUnit 4.11 or later recommended)*

Deprecated code

The following classes and methods have been deprecated in Spring Framework 4.0 or will be deprecated along the Spring Framework 4.x line. They will be removed at a future date, so please check the javadocs and migrate to the suggested alternatives...

Hibernate 3.6.10

The `org.springframework.orm.hibernate3` package will be phasing out as of Spring Framework 4.2. We keep supporting it for the time being; however, we strongly recommend an upgrade to Hibernate 4.2/4.3 or 5.0.

As of Spring Framework 4.0.1, we provide a HibernateTemplate variant in `org.springframework.orm.hibernate4` to ease migration for common Hibernate 3.x data access code, in particular if your motivation for an upgrade is the lack of bug fixes in the Hibernate 3.x line. Note that newly written code is recommended to use Hibernate's native `SessionFactory.getCurrentSession()` style.

On a related note, HibernateInterceptor is deprecated in `org.springframework.orm.hibernate3` and doesn't exist anymore in `org.springframework.orm.hibernate4`. As a replacement for basic Session binding needs outside of transactions, consider the use of the new OpenSessionInterceptor variant, available for both Hibernate 3 and 4 as of Spring Framework 4.0.2.

Note: The Spring Framework 4.0.0 release accidentally restricted HibernateTemplate's List element types to Object only. This has been fixed as of 4.0.2 (https://jira.springsource.org/browse/SPR-11402), allowing for immediate casts to specifically typed Lists again. If you run into any issues migrating existing Hibernate access code, please upgrade to Spring Framework 4.0.2 first.

Tiles 2.2.2

While Spring Framework 4.0 and 4.1 still fully support Tiles 2.2.2, the corresponding `org.springframework.web.servlet.view.tiles2` package will be deprecated as of Spring Framework 4.2. We recommend a timely upgrade to Tiles 3.0.5, supported in `org.springframework.web.servlet.view.tiles3`.

Quartz 1.8

Quartz 1.8 support in the `org.springframework.scheduling.support` package is deprecated and has been removed for Spring Framework 4.1, with that package only working with Quartz 2.1.4+ from then onwards.

Jackson 1.8/1.9

All Jackson v1 support is deprecated in favor of Jackson v2 and has been removed in Spring Framework 4.1:

- `JacksonObjectMapperFactoryBean`
- `MappingJacksonHttpMessageConverter`
- `MappingJacksonJsonView`
- `MappingJacksonMessageConverter` for JMS

Burlap

Burlap is no longer under active development and support will be dropped entirely in the future:

- `BurlapClientInterceptor`
- `BurlapExporter`
- `BurlapProxyFactoryBean`
- `BurlapServiceExporter`
- `SimpleBurlapServiceExporter`

Outdated JBoss classes

The following classes are deprecated since they no longer work with current JBoss releases:

- `JBossWorkManagerTaskExecutor`
- `JBossWorkManagerUtils`

JAX-WS feature configuration

- `AbstractJaxWsServiceExporter.setWebServiceFeatures(Object[] webServiceFeatures)`
- `JaxWsPortClientInterceptor.setWebServiceFeatures(Object[] webServiceFeatures)`

Generic type resolution

Several methods from the `GenericTypeResolver` have been deprecated. For new code the `ResolvableType` class provides a good alternative to `GenericTypeResolver` & `GenericCollectionTypeResolver`:

- `GenericTypeResolver.getTargetType(MethodParameter methodParam)`
- `GenericTypeResolver.resolveType(Type genericType, Map<TypeVariable, Type> map)`
- `GenericTypeResolver.getTypeVariableMap(Class<?> clazz)`

Default cache key generator

The default `KeyGenerator` used by Spring's cache abstraction has changed from `DefaultKeyGenerator` to `SimpleKeyGenerator`. The new generator does not suffer from key collisions and less likely to cause a cached method to return incorrect results. You will need to configure the deprecated `DefaultKeyGenerator` if you prefer the previous key strategy, or create a custom `KeyGenerator` implementation yourself.

MVC namespace

The Spring MVC namespace XSD had been updated to correct the casing used for a couple of attributes. When upgrading to `spring-mvc-4.0.xsd`, you should replace `enableMatrixVariables` and `ignoreDefaultModelOnRedirect` with `enable-matrix-variables` and `ignore-default-model-on-redirect` respectively.

Using Spring's JSP tags with FreeMarker

As of Spring 4.0, Spring's JSP tag library uses specifically declared attribute types, relying on externally applied EL parsing as supported by JSP 2.0+. As a consequence, some attributes that used to accept Object values - e.g. for String EL expressions or Boolean target values - are now specifically declared for the actual target type - e.g. boolean, suggesting a target type for external EL parsing.

Native JSP parsing can handle this change transparently, just requiring a recompile of the JSP pages. However, FreeMarker's JSP tag support cannot handle String values against a boolean target type - e.g. readonly='false' - and requires a change to an actual boolean value - e.g. readonly=false. In case you happened to specify such values as Strings before, please adapt those values to the actual target type.

Spring MVC Test and Java 6

An issue with compiling Spring MVC Test framework tests with JDK 1.6 has been identified and fixed for version 4.0.1.