**3. New Features and Enhancements in Spring Framework 4.0**

The Spring Framework was first released in 2004; since then there have been significant major revisions: Spring 2.0 provided XML namespaces and AspectJ support; Spring 2.5 embraced annotation-driven configuration; Spring 3.0 introduced a strong Java 5+ foundation across the framework codebase, and features such as the Java-based @Configuration model.

Version 4.0 is the latest major release of the Spring Framework and the first to fully support Java 8 features. You can still use Spring with older versions of Java, however, the minimum requirement has now been raised to Java SE 6. We have also taken the opportunity of a major release to remove many deprecated classes and methods.

A [migration guide for upgrading to Spring 4.0](https://github.com/spring-projects/spring-framework/wiki/Migrating-from-earlier-versions-of-the-spring-framework) is available on the [Spring Framework GitHub Wiki](https://github.com/spring-projects/spring-framework/wiki).

**3.1 Improved Getting Started Experience**

The new [spring.io](https://spring.io/) website provides a whole series of ["Getting Started"](https://spring.io/guides) guides to help you learn Spring. You can read more about the guides in the [Chapter 1, *Getting Started with Spring*](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview-getting-started-with-spring.html) section in this document. The new website also provides a comprehensive overview of the many additional projects that are released under the Spring umbrella.

If you are a Maven user you may also be interested in the helpful [bill of materials](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html#overview-maven-bom) POM file that is now published with each Spring Framework release.

**3.2 Removed Deprecated Packages and Methods**

All deprecated packages, and many deprecated classes and methods have been removed with version 4.0. If you are upgrading from a previous release of Spring, you should ensure that you have fixed any deprecated calls that you were making to outdated APIs.

For a complete set of changes, check out the [API Differences Report](http://docs.spring.io/spring-framework/docs/3.2.4.RELEASE_to_4.0.0.RELEASE/).

Note that optional third-party dependencies have been raised to a 2010/2011 minimum (i.e. Spring 4 generally only supports versions released in late 2010 or later now): notably, Hibernate 3.6+, EhCache 2.1+, Quartz 1.8+, Groovy 1.8+, and Joda-Time 2.0+. As an exception to the rule, Spring 4 requires the recent Hibernate Validator 4.3+, and support for Jackson has been focused on 2.0+ now (with Jackson 1.8/1.9 support retained for the time being where Spring 3.2 had it; now just in deprecated form).

**3.3 Java 8 (as well as 6 and 7)**

Spring Framework 4.0 provides support for several Java 8 features. You can make use of *lambda expressions* and *method references* with Spring’s callback interfaces. There is first-class support for java.time ([JSR-310](http://jcp.org/en/jsr/detail?id=310)), and several existing annotations have been retrofitted as @Repeatable. You can also use Java 8’s parameter name discovery (based on the -parameters compiler flag) as an alternative to compiling your code with debug information enabled.

Spring remains compatible with older versions of Java and the JDK: concretely, Java SE 6 (specifically, a minimum level equivalent to JDK 6 update 18, as released in January 2010) and above are still fully supported. However, for newly started development projects based on Spring 4, we recommend the use of Java 7 or 8.

**3.4 Java EE 6 and 7**

Java EE version 6 or above is now considered the baseline for Spring Framework 4, with the JPA 2.0 and Servlet 3.0 specifications being of particular relevance. In order to remain compatible with Google App Engine and older application servers, it is possible to deploy a Spring 4 application into a Servlet 2.5 environment. However, Servlet 3.0+ is strongly recommended and a prerequisite in Spring’s test and mock packages for test setups in development environments.

|  |  |
| --- | --- |
| [Note] | **Note** |
| If you are a WebSphere 7 user, be sure to install the JPA 2.0 feature pack. On WebLogic 10.3.4 or higher, install the JPA 2.0 patch that comes with it. This turns both of those server generations into Spring 4 compatible deployment environments. |

On a more forward-looking note, Spring Framework 4.0 supports the Java EE 7 level of applicable specifications now: in particular, JMS 2.0, JTA 1.2, JPA 2.1, Bean Validation 1.1, and JSR-236 Concurrency Utilities. As usual, this support focuses on individual use of those specifications, e.g. on Tomcat or in standalone environments. However, it works equally well when a Spring application is deployed to a Java EE 7 server.

Note that Hibernate 4.3 is a JPA 2.1 provider and therefore only supported as of Spring Framework 4.0. The same applies to Hibernate Validator 5.0 as a Bean Validation 1.1 provider. Neither of the two are officially supported with Spring Framework 3.2.

**3.5 Groovy Bean Definition DSL**

Beginning with Spring Framework 4.0, it is possible to define external bean configuration using a Groovy DSL. This is similar in concept to using XML bean definitions but allows for a more concise syntax. Using Groovy also allows you to easily embed bean definitions directly in your bootstrap code. For example:

def reader = **new** GroovyBeanDefinitionReader(myApplicationContext)

reader.beans {

dataSource(BasicDataSource) {

driverClassName = "org.hsqldb.jdbcDriver"

url = "jdbc:hsqldb:mem:grailsDB"

username = "sa"

password = ""

settings = [mynew:"setting"]

}

sessionFactory(SessionFactory) {

dataSource = dataSource

}

myService(MyService) {

nestedBean = { AnotherBean bean ->

dataSource = dataSource

}

}

}

For more information consult the GroovyBeanDefinitionReader [javadocs](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/beans/factory/groovy/GroovyBeanDefinitionReader.html).

**3.6 Core Container Improvements**

There have been several general improvements to the core container:

* Spring now treats [*generic types* as a form of *qualifier*](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-generics-as-qualifiers) when injecting Beans. For example, if you are using a Spring Data Repository you can now easily inject a specific implementation: @Autowired Repository<Customer> customerRepository.
* If you use Spring’s meta-annotation support, you can now develop custom annotations that [expose specific attributes from the source annotation](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-meta-annotations).
* Beans can now be *ordered* when they are [autowired into lists and arrays](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-autowired-annotation). Both the @Order annotation and Ordered interface are supported.
* The @Lazy annotation can now be used on injection points, as well as on @Bean definitions.
* The [@Description annotation has been introduced](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-java-bean-description) for developers using Java-based configuration.
* A generalized model for [conditionally filtering beans](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#beans-java-conditional) has been added via the @Conditional annotation. This is similar to @Profile support but allows for user-defined strategies to be developed programmatically.
* [CGLIB-based proxy classes](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop-api.html#aop-pfb-proxy-types) no longer require a default constructor. Support is provided via the [objenesis](http://code.google.com/p/objenesis/) library which is repackaged *inline* and distributed as part of the Spring Framework. With this strategy, no constructor at all is being invoked for proxy instances anymore.
* There is managed time zone support across the framework now, e.g. on LocaleContext.

**3.7 General Web Improvements**

Deployment to Servlet 2.5 servers remains an option, but Spring Framework 4.0 is now focused primarily on Servlet 3.0+ environments. If you are using the [Spring MVC Test Framework](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/testing.html#spring-mvc-test-framework) you will need to ensure that a Servlet 3.0 compatible JAR is in your *test classpath*.

In addition to the WebSocket support mentioned later, the following general improvements have been made to Spring’s Web modules:

* You can use the [new @RestController annotation](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-restcontroller) with Spring MVC applications, removing the need to add @ResponseBody to each of your @RequestMapping methods.
* The AsyncRestTemplate class has been added, [allowing non-blocking asynchronous support](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/remoting.html#rest-async-resttemplate) when developing REST clients.
* Spring now offers [comprehensive timezone support](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-timezone) when developing Spring MVC applications.

**3.8 WebSocket, SockJS, and STOMP Messaging**

A new spring-websocket module provides comprehensive support for WebSocket-based, two-way communication between client and server in web applications. It is compatible with [JSR-356](http://jcp.org/en/jsr/detail?id=356), the Java WebSocket API, and in addition provides SockJS-based fallback options (i.e. WebSocket emulation) for use in browsers that don’t yet support the WebSocket protocol (e.g. Internet Explorer < 10).

A new spring-messaging module adds support for STOMP as the WebSocket sub-protocol to use in applications along with an annotation programming model for routing and processing STOMP messages from WebSocket clients. As a result an @Controller can now contain both @RequestMapping and @MessageMapping methods for handling HTTP requests and messages from WebSocket-connected clients. The new spring-messaging module also contains key abstractions formerly from the [Spring Integration](http://projects.spring.io/spring-integration/) project such as Message, MessageChannel, MessageHandler, and others to serve as a foundation for messaging-based applications.

For further details, including a more thorough introduction, see the [Chapter 21, *WebSocket Support*](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html) section.

**3.9 Testing Improvements**

In addition to pruning of deprecated code within the spring-test module, Spring Framework 4.0 introduces several new features for use in unit and integration testing.

* Almost all annotations in the spring-test module (e.g., @ContextConfiguration, @WebAppConfiguration, @ContextHierarchy, @ActiveProfiles, etc.) can now be used as [meta-annotations](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/testing.html#integration-testing-annotations-meta) to create custom *composed annotations* and reduce configuration duplication across a test suite.
* Active bean definition profiles can now be resolved programmatically, simply by implementing a custom [ActiveProfilesResolver](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/testing.html#testcontext-ctx-management-env-profiles-ActiveProfilesResolver) and registering it via the resolver attribute of @ActiveProfiles.
* A new SocketUtils class has been introduced in the spring-core module which enables you to scan for free TCP and UDP server ports on localhost. This functionality is not specific to testing but can prove very useful when writing integration tests that require the use of sockets, for example tests that start an in-memory SMTP server, FTP server, Servlet container, etc.
* As of Spring 4.0, the set of mocks in the org.springframework.mock.web package is now based on the Servlet 3.0 API. Furthermore, several of the Servlet API mocks (e.g., MockHttpServletRequest, MockServletContext, etc.) have been updated with minor enhancements and improved configurability.

**Core Container Improvements**

* Annotations such as @Bean get detected and processed on Java 8 default methods as well, allowing for composing a configuration class from interfaces with default @Bean methods.
* Configuration classes may declare @Import with regular component classes now, allowing for a mix of imported configuration classes and component classes.
* Configuration classes may declare an @Order value, getting processed in a corresponding order (e.g. for overriding beans by name) even when detected through classpath scanning.
* @Resource injection points support an @Lazy declaration, analogous to @Autowired, receiving a lazy-initializing proxy for the requested target bean.
* The application event infrastructure now offers an [annotation-based model](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html#context-functionality-events-annotation) as well as the ability to publish any arbitrary event.
  + Any public method in a managed bean can be annotated with @EventListener to consume events.
  + @TransactionalEventListener provides transaction-bound event support.
* Spring Framework 4.2 introduces first-class support for declaring and looking up aliases for annotation attributes. The new @AliasFor annotation can be used to declare a pair of aliased attributes within a single annotation or to declare an alias from one attribute in a custom composed annotation to an attribute in a meta-annotation.
  + The following annotations have been retrofitted with @AliasFor support in order to provide meaningful aliases for their value attributes: @Cacheable, @CacheEvict, @CachePut, @ComponentScan, @ComponentScan.Filter, @ImportResource, @Scope, @ManagedResource, @Header, @Payload, @SendToUser, @ActiveProfiles, @ContextConfiguration, @Sql, @TestExecutionListeners, @TestPropertySource, @Transactional, @ControllerAdvice, @CookieValue, @CrossOrigin, @MatrixVariable, @RequestHeader, @RequestMapping, @RequestParam, @RequestPart, @ResponseStatus, @SessionAttributes, @ActionMapping, @RenderMapping, @EventListener, @TransactionalEventListener.
  + For example, @ContextConfiguration from the spring-test module is now declared as follows:

public @interface ContextConfiguration {

@AliasFor("locations")

String[] value() default {};

@AliasFor("value")

String[] locations() default {};

// ...

}

* Similarly, *composed annotations* that override attributes from meta-annotations can now use @AliasFor for fine-grained control over exactly which attributes are overridden within an annotation hierarchy. In fact, it is now possible to declare an alias for the value attribute of a meta-annotation.
* For example, one can now develop a composed annotation with a custom attribute override as follows.

@ContextConfiguration

public @interface MyTestConfig {

@AliasFor(annotation = ContextConfiguration.class, attribute = "value")

String[] xmlFiles();

// ...

}

* See [Spring Annotation Programming Model](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/annotation-programming-model.html).
  + Numerous improvements to Spring’s search algorithms used for finding meta-annotations. For example, locally declared *composed annotations* are now favored over inherited annotations.
  + *Composed annotations* that override attributes from meta-annotations can now be discovered on interfaces and on abstract, bridge, & interface methods as well as on classes, standard methods, constructors, and fields.
  + Maps representing annotation attributes (and AnnotationAttributes instances) can be *synthesized* (i.e., converted) into an annotation.
  + The features of field-based data binding (DirectFieldAccessor) have been aligned with the current property-based data binding (BeanWrapper). In particular, field-based binding now supports navigation for Collections, Arrays, and Maps.
  + DefaultConversionService now provides out-of-the-box converters for Stream, Charset, Currency, and TimeZone. Such converters can be added individually to any arbitrary ConversionService as well.
  + DefaultFormattingConversionService comes with out-of-the-box support for the value types in JSR-354 Money & Currency (if the 'javax.money' API is present on the classpath): namely, MonetaryAmount and CurrencyUnit. This includes support for applying @NumberFormat.
  + @NumberFormat can now be used as a meta-annotation.
  + JavaMailSenderImpl has a new testConnection() method for checking connectivity to the server.
  + ScheduledTaskRegistrar exposes scheduled tasks.
  + Apache commons-pool2 is now supported for a pooling AOP CommonsPool2TargetSource.
  + Introduced StandardScriptFactory as a JSR-223 based mechanism for scripted beans, exposed through the lang:std element in XML. Supports e.g. JavaScript and JRuby. (Note: JRubyScriptFactory and lang:jruby are deprecated now, in favor of using JSR-223.)

**5.2 Data Access Improvements**

* javax.transaction.Transactional is now supported via AspectJ.
* SimpleJdbcCallOperations now supports named binding.
* Full support for Hibernate ORM 5.0: as a JPA provider (automatically adapted) as well as through its native API (covered by the new org.springframework.orm.hibernate5 package).
* Embedded databases can now be automatically assigned unique names, and <jdbc:embedded-database> supports a new database-name attribute. See "Testing Improvements" below for further details.

**5.3 JMS Improvements**

* The autoStartup attribute can be controlled via JmsListenerContainerFactory.
* The type of the reply Destination can now be configured per listener container.
* The value of the @SendTo annotation can now use a SpEL expression.
* The response destination can be [computed at runtime using JmsResponse](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/jms.html#jms-annotated-response)
* @JmsListener is now a repeatable annotation to declare several JMS containers on the same method (use the newly introduced @JmsListeners if you’re not using Java8 yet).

**5.4 Web Improvements**

* HTTP Streaming and Server-Sent Events support, see [the section called “HTTP Streaming”](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-async-http-streaming).
* Built-in support for CORS including global (MVC Java config and XML namespace) and local (e.g. @CrossOrigin) configuration. See [Chapter 26, *CORS Support*](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/cors.html) for details.
* HTTP caching updates:
  + new CacheControl builder; plugged into ResponseEntity, WebContentGenerator, ResourceHttpRequestHandler.
  + improved ETag/Last-Modified support in WebRequest.
* Custom mapping annotations, using @RequestMapping as a meta-annotation.
* Public methods in AbstractHandlerMethodMapping to register and unregister request mappings at runtime.
* Protected createDispatcherServlet method in AbstractDispatcherServletInitializer to further customize the DispatcherServlet instance to use.
* HandlerMethod as a method argument on @ExceptionHandler methods, especially handy in @ControllerAdvice components.
* java.util.concurrent.CompletableFuture as an @Controller method return value type.
* Byte-range request support in HttpHeaders and for serving static resources.
* @ResponseStatus detected on nested exceptions.
* UriTemplateHandler extension point in the RestTemplate.
  + DefaultUriTemplateHandler exposes baseUrl property and path segment encoding options.
  + the extension point can also be used to plug in any URI template library.
* [OkHTTP](http://square.github.io/okhttp/) integration with the RestTemplate.
* Custom baseUrl alternative for methods in MvcUriComponentsBuilder.
* Serialization/deserialization exception messages are now logged at WARN level.
* Default JSON prefix has been changed from "{} && " to the safer ")]}', " one.
* New RequestBodyAdvice extension point and built-in implementation to support Jackson’s @JsonView on @RequestBody method arguments.
* When using GSON or Jackson 2.6+, the handler method return type is used to improve serialization of parameterized types like List<Foo>.
* Introduced ScriptTemplateView as a JSR-223 based mechanism for scripted web views, with a focus on JavaScript view templating on Nashorn (JDK 8).

**5.5 WebSocket Messaging Improvements**

* Expose presence information about connected users and subscriptions:
  + new SimpUserRegistry exposed as a bean named "userRegistry".
  + sharing of presence information across cluster of servers (see broker relay config options).
* Resolve user destinations across cluster of servers (see broker relay config options).
* StompSubProtocolErrorHandler extension point to customize and control STOMP ERROR frames to clients.
* Global @MessageExceptionHandler methods via @ControllerAdvice components.
* Heart-beats and a SpEL expression 'selector' header for subscriptions with SimpleBrokerMessageHandler.
* STOMP client for use over TCP and WebSocket; see [Section 25.4.13, “STOMP Client”](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html#websocket-stomp-client).
* @SendTo and @SendToUser can contain destination variable placeholders.
* Jackson’s @JsonView supported for return values on @MessageMapping and @SubscribeMapping methods.
* ListenableFuture and CompletableFuture as return value types from @MessageMapping and @SubscribeMapping methods.
* MarshallingMessageConverter for XML payloads.

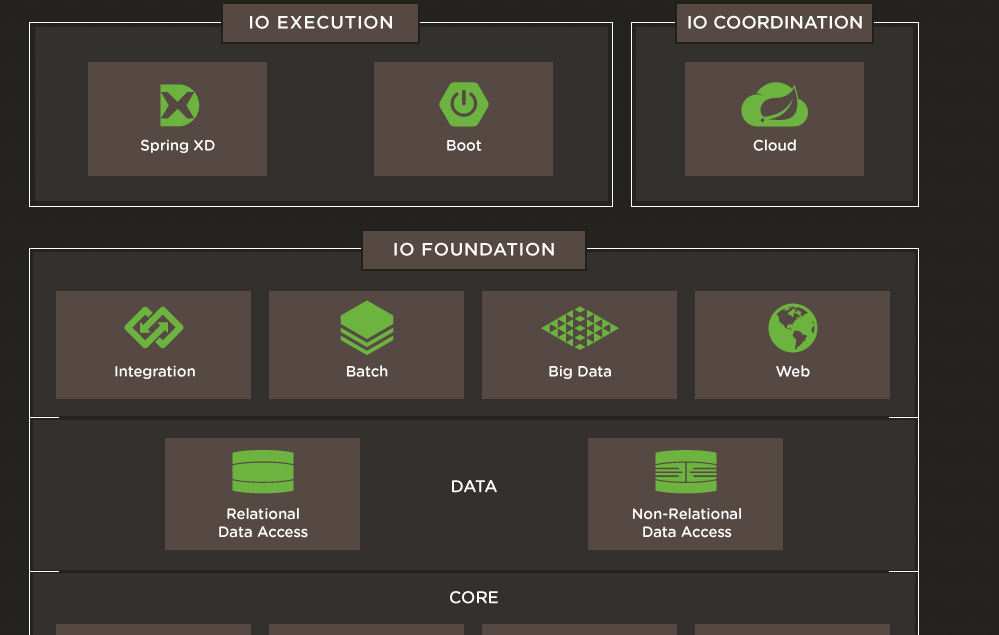
**5.6 Testing Improvements**

* JUnit-based integration tests can now be executed with JUnit rules instead of the SpringJUnit4ClassRunner. This allows Spring-based integration tests to be run with alternative runners like JUnit’s Parameterized or third-party runners such as the MockitoJUnitRunner.
  + See [the section called “Spring JUnit Rules”](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/integration-testing.html#testcontext-junit4-rules) for details.
* The Spring MVC Test framework now provides first-class support for HtmlUnit, including integration with Selenium’s WebDriver, allowing for page-based web application testing without the need to deploy to a Servlet container.
  + See [Section 14.6.2, “HtmlUnit Integration”](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/integration-testing.html#spring-mvc-test-server-htmlunit) for details.
* AopTestUtils is a new testing utility that allows developers to obtain a reference to the underlying target object hidden behind one or more Spring proxies.
  + See [Section 13.2.1, “General testing utilities”](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/unit-testing.html#unit-testing-utilities) for details.
* ReflectionTestUtils now supports setting and getting static fields, including constants.
* The original ordering of bean definition profiles declared via @ActiveProfiles is now retained in order to support use cases such as Spring Boot’s ConfigFileApplicationListener which loads configuration files based on the names of active profiles.
* @DirtiesContext supports new BEFORE\_METHOD, BEFORE\_CLASS, and BEFORE\_EACH\_TEST\_METHOD modes for closing the ApplicationContext *before* a test — for example, if some rogue (i.e., yet to be determined) test within a large test suite has corrupted the original configuration for the ApplicationContext.
* @Commit is a new annotation that may be used as a direct replacement for @Rollback(false).
* @Rollback may now be used to configure class-level *default rollback* semantics.
  + Consequently, @TransactionConfiguration is now deprecated and will be removed in a subsequent release.
* @Sql now supports execution of *inlined SQL statements* via a new statements attribute.
* The ContextCache that is used for caching ApplicationContexts between tests is now a public API with a default implementation that can be replaced for custom caching needs.
* DefaultTestContext, DefaultBootstrapContext, and DefaultCacheAwareContextLoaderDelegate are now public classes in the support subpackage, allowing for custom extensions.
* TestContextBootstrappers are now responsible for building the TestContext.
* In the Spring MVC Test framework, MvcResult details can now be logged at DEBUG level or written to a custom OutputStream or Writer. See the new log(), print(OutputStream), and print(Writer) methods in MockMvcResultHandlers for details.
* The JDBC XML namespace supports a new database-name attribute in <jdbc:embedded-database>, allowing developers to set unique names for embedded databases –- for example, via a SpEL expression or a property placeholder that is influenced by the current active bean definition profiles.
* Embedded databases can now be automatically assigned a unique name, allowing common test database configuration to be reused in different ApplicationContexts within a test suite.
  + See [Section 18.8.6, “Generating unique names for embedded databases”](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/jdbc.html#jdbc-embedded-database-unique-names) for details.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Layers\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

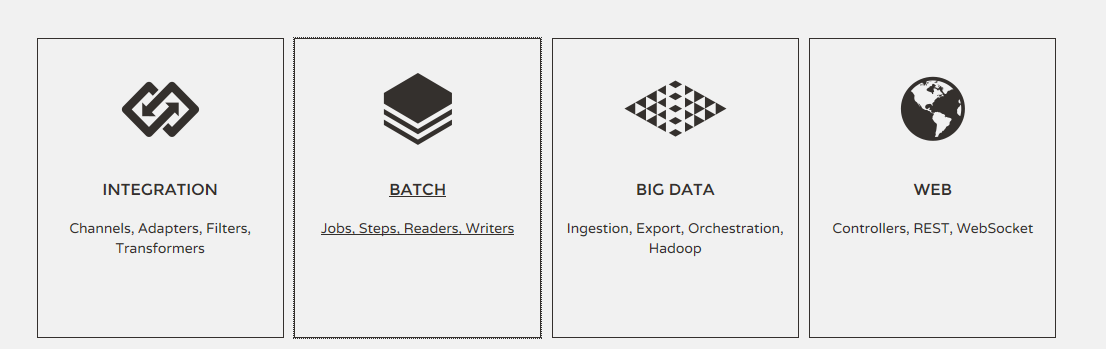
# Spring IO Foundation

The Spring IO Foundation layer is a cohesive set of APIs and embeddable runtime components that enable you to build applications addressing a wide range of enterprise requirements. For any given application you will of course only need to include those parts of the foundation that your application actually requires. The foundation makes it easy to identify components that work together; it does not force you to include components you wouldn't otherwise need.



## Spring-Powered Workloads

All of the major workload types are represented: enterprise integration, batch processing, big data processing, web applications (including WebSocket support), and the creation of server-side APIs in support of single-page apps, mobile clients, and services.



Spring's web stack has evolved to include comprehensive REST support for the creation of RESTful APIs – useful for supporting mobile and rich-client application architectures. Spring Framework 4 also adds support for the WebSocket programming model.

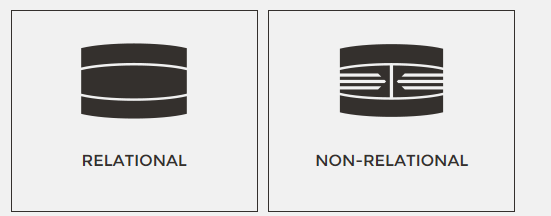
Integration workloads are another major use case for Spring. For these, Spring IO provides a rich programming model based on [Enterprise Integration Patterns](http://books.google.com/books/about/Enterprise_Integration_Patterns.html?id=qqB7nrrna_sC) and its notions of channels, adapters, filters, and transformers.

Spring IO also defines a domain language for batch processing – batch jobs are divided into processing steps with readers and writers used to read the data in and write it back out again. Spring IO's comprehensive batch framework enables you to build robust batch applications that process large volumes of records.

Of course any given application in the enterprise rarely falls neatly into just one of these buckets, and often contains elements of web, batch, and integration processing all in one. Spring's embedded runtime model makes this easy and natural to do. The emerging category of big data workloads is an example of this. It's one thing to lay down a Hadoop cluster but it's quite another to figure out how you're going to ingest data from all over your enterprise into that cluster, how you're going to orchestrate the batch workloads that process that data, and how you're going to get the results of that processing back into a form that can drive the rest of your business. A fundamental insight here is that these "big data problems" have a lot in common with traditional enterprise integration and batch processing. Spring IO supplements Spring's traditional strengths in these areas to provide all the support you need to build robust and maintainable big data applications.

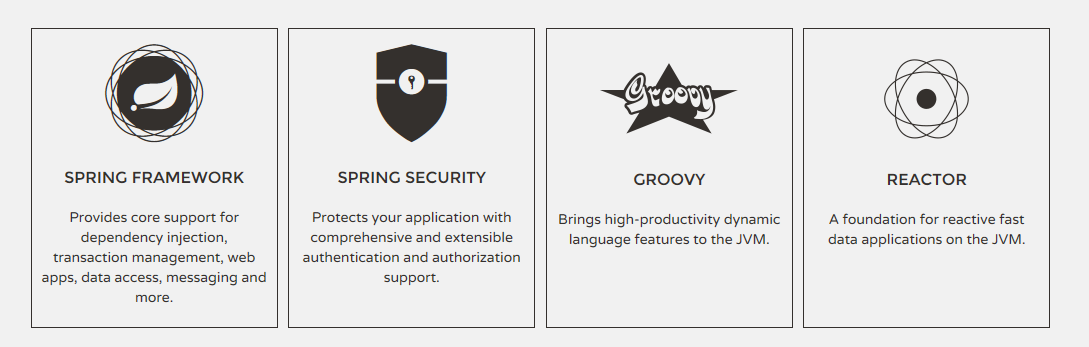
## Data

The data landscape is evolving rapidly. When Spring got started, 'data' pretty much meant a traditional relational database, and maybe a cache in front of it if you were lucky. Today we have a wide variety of data stores covering structured and unstructured data, in-memory and on-disk, standard-size and big. The industry has come to recognize that one size does not fit all, and it's a matter of choosing the most appropriate store for the task in hand. Spring IO addresses this modern data landscape—including full support for Apache Hadoop—whether it be document, graph, key-value, relational, or simply unstructured files.



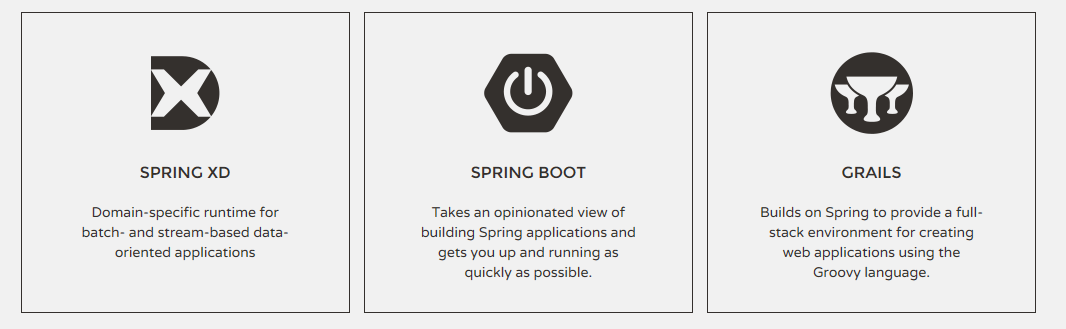
## Core

At the center of the Spring IO Foundation layer, and common across all workload types, we find the core framework itself. Via Spring Framework 4.0, Spring IO enables you to take full advantage of Java SE 8 and the latest updates from EE 7. Spring IO has full support for addressing security concerns, and Spring IO's Reactor provides a foundation for building reactive, asynchronous, event- and data-driven applications.



The popular Groovy dynamic language is supported as an integral part of the Spring IO platform. Groovy integrates seamlessly with your existing classes and libraries and works especially well with IO Execution DSRs such as Boot and Grails.

# Spring IO Execution

The Spring IO Execution layer provides domain-specific runtimes (DSRs) for applications built on the IO Foundation modules. A DSR may run standalone without requiring deployment to an external container. The first release of Spring IO will include three DSRs: Spring XD, Spring Boot, and Grails.

Spring XD provides a powerful runtime and DSL for describing big data ingestion and analytics, export, and Hadoop workflow management. In many cases, descriptions of streams, taps and jobs can be directly pushed to the XD DSR without requiring any code to be written at all. The set of XD modules (components that can be used in DSL expressions) is easily extended using building blocks based on Spring Integration and Spring Batch.

Spring Boot reduces the effort needed to create production-ready, DevOps-friendly, XML-free Spring applications. It simplifies bootstrapping of Spring projects with minimal code, implements an extensible set of operational features such as automated health checking and metrics endpoints, and supports embedded containers enabling the creation of self-contained executables.

Grails provides a productive and stream-lined full-stack web framework by combining the power of the Spring IO Foundation components with a set of comprehensive Groovy-based DSLs.