Chapter 2. What's new in Spring 2.0 and 2.5?

2.1. Introduction

If you have been using the Spring Framework for some time, you will be aware that Spring has undergone two major revisions: Spring 2.0, released in October 2006, and Spring 2.5, released in November 2007.

> **Java SE and Java EE Support**
>
> The Spring Framework continues to be compatible with all versions of Java since (and including) Java 1.4.2. This means that Java 1.4.2, Java 5 and Java 6 are supported, although some advanced functionality of the Spring Framework will not be available to you if you are committed to using Java 1.4.2. Spring 2.5 introduces dedicated support for Java 6, after Spring 2.0's in-depth support for Java 5 throughout the framework.
>
> Furthermore, Spring remains compatible with J2EE 1.3 and higher, while at the same time introducing dedicated support for Java EE 5. This means that Spring can be consistently used on application servers such as BEA WebLogic 8.1, 9.0, 9.2 and 10, IBM WebSphere 5.1, 6.0, 6.1 and 7, Oracle OC4J 10.1.3 and 11, JBoss 3.2, 4.0, 4.2 and 5.0, as well as Tomcat 4.1, 5.0, 5.5 and 6.0, Jetty 4.2, 5.1 and 6.1, Resin 2.1, 3.0 and 3.1 and GlassFish V1 and V2.
>
> *NOTE: We generally recommend using the most recent version of each application server generation. In particular, make sure you are using BEA WebLogic 8.1 SP6 or higher and WebSphere 6.0.2.19 / 6.1.0.9 or higher, respectively, when using those WebLogic and WebSphere generations with Spring 2.5.*

This chapter is a guide to the new and improved features of Spring 2.0 and 2.5. It is intended to provide a high-level summary so that seasoned Spring architects and developers can become immediately familiar with the new Spring 2.x functionality. For more in-depth information on the features, please refer to the corresponding sections hyperlinked from within this chapter.

2.2. The Inversion of Control (IoC) container

IoC容器 层面

One of the areas that contains a considerable number of 2.0 and 2.5 improvements is Spring's IoC container.

2.2.1. New bean scopes

新的Bean作用域

单实例、原型

Previous versions of Spring had IoC container level support for exactly two distinct bean scopes (singleton and prototype). Spring 2.0 improves on this by not only providing a number of additional scopes depending on the environment in which Spring is being deployed (for example, request and session scoped beans in a web environment), but also by providing integration points so that Spring users can create their own scopes.

请求、会话

It should be noted that although the underlying (and internal) implementation for singleton- and prototype-scoped beans has been changed, this change is totally transparent to the end user... no existing configuration needs to change, and no existing configuration will break.

Both the new and the original scopes are detailed in the section entitled Section 3.4, “Bean scopes”.

2.2.2. Easier XML configuration 更加简单的XML配置（基于XML模式的新的XML配置语法）

Spring XML configuration is now even easier, thanks to the advent of the new XML configuration syntax based on XML Schema. If you want to take advantage of the new tags that Spring provides (and the Spring team certainly suggest that you do because they make configuration less verbose and easier to read), then do read the section entitled Appendix A, *XML Schema-based configuration*.

On a related note, there is a new, updated DTD for Spring 2.0 that you may wish to reference if you cannot take advantage of the XML Schema-based configuration. The DOCTYPE declaration is included below for your convenience, but the interested reader should definitely read the `'spring-beans-2.0.dtd'` DTD included in the `'dist/resources'` directory of the Spring 2.5 distribution.

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                  "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

2.2.3. Extensible XML authoring 可扩展的XML创作

Not only is XML configuration easier to write, it is now also extensible.

What 'extensible' means in this context is that you, as an application developer, or (more likely) as a third party framework or product vendor, can write custom tags that other developers can then plug into their own Spring configuration files. This allows you to have your own domain specific language (the term is used loosely here) of sorts be reflected in the specific configuration of your own components.

Implementing custom Spring tags may not be of interest to every single application developer or enterprise architect using Spring in their own projects. We expect third-party vendors to be highly interested in developing custom configuration tags for use in Spring configuration files.

The extensible configuration mechanism is documented in Appendix B, *Extensible XML authoring*.

2.2.4. Annotation-driven configuration 注解驱动的配置

2.0引入了支持配置目的的各种注解

Spring 2.0 introduced support for various annotations for configuration purposes, such as `@Transactional`, `@Required` and `@PersistenceContext` /`@PersistenceUnit`.

一系列配置注解

Spring 2.5 introduces support for a complete set of configuration annotations: `@Autowired` in combination with support for the JSR-250 annotations `@Resource`, `@PostConstruct` and `@PreDestroy` .

Annotation-driven bean configuration is discussed in Section 3.11, “Annotation-based configuration”. Check out annotation support for Spring MVC as well: Section 2.5.3, “Annotation-based controllers”

基于注解的控制器

2.2.5. Autodetecting components in the classpath 类路径下组件自动探测

2.5引入了支持“组件扫描”：自动探测类路径下注解的组件

Spring 2.5 introduces support component scanning: autodetecting annotated components in the classpath. Typically, such component classes will be annotated with stereotypes such as `@Component`, `@Repository`, `@Service`, `@Controller`. Depending on the application context configuration, such component classes will be autodetected and turned into Spring bean definitions, not requiring explicit configuration for each such bean.

Annotation-driven bean configuration is discussed in Section 3.12.1, “`@Component` and further stereotype annotations”. 注解驱动的Bean配置

2.3. Aspect Oriented Programming (AOP)

面向切面编程

Spring 2.0 has a much improved AOP offering. The Spring AOP framework itself is markedly easier to configure in XML, and significantly less verbose as a result; and Spring 2.0 integrates with the AspectJ pointcut language and @AspectJ aspect declaration style. The chapter entitled Chapter 6, *Aspect Oriented Programming with Spring* is dedicated to describing this new support.

2.0集成AspectJ切入点语言和@AspectJ切面声明方式

2.3.1. Easier AOP XML configuration

更加简单的AOP XML配置

Spring 2.0 introduces new schema support for defining aspects backed by regular Java objects. This support takes advantage of the AspectJ pointcut language and offers fully typed advice (i.e. no more casting and `Object[]` argument manipulation). Details of this support can be found in the section entitled Section 6.3, "Schema-based AOP support".

基于“模式”的AOP支持

2.3.2. Support for @AspectJ aspects

@AspectJ切面 支持

Spring 2.0 also supports aspects defined using the @AspectJ annotations. These aspects can be shared between AspectJ and Spring AOP, and require (honestly!) only some simple configuration. Said support for @AspectJ aspects is discussed in Section 6.2, "@AspectJ support".

2.3.3. Support for bean name pointcut element

Bean名称切点元素支持

Spring 2.5 introduces support for the `bean(...)` pointcut element, matching specific named beans according to Spring-defined bean names. See Section 6.2.3.1, "Supported Pointcut Designators" for details.

2.3.4. Support for AspectJ load-time weaving

基于代理的AOP框架：AspectJ装载时编织支持

Spring 2.5 introduces explicit support AspectJ load-time weaving, as alternative to the proxy-based AOP framework. The new `context:load-time-weaver` configuration element automatically activates AspectJ aspects as defined in AspectJ's `META-INF/aop.xml` descriptor, applying them to the current application context through registering a transformer with the underlying ClassLoader. Note that this only works in environments with class transformation support. Check out Section 6.8.4, "Load-time weaving with AspectJ in the Spring Framework" for the capabilities and limitations.

2.4. The Middle Tier

中间层

在XML中简单配置“声明式的事务”

2.4.1. Easier configuration of declarative transactions in XML

The way that transactions are configured in Spring 2.0 has been changed significantly. The previous 1.2.x style of configuration continues to be valid (and supported), but the new style is markedly less verbose and is the recommended style. Spring 2.0 also ships with an AspectJ aspects library that you can use to make pretty much any object transactional - even objects not created by the Spring IoC container.

2.5支持方便的注解驱动事务管理

Spring 2.5 supports convenient annotation-driven transaction management in combination with load-time weaving, through the use of `context:load-time-weaver` in combination with `tx:annotation-driven mode="aspectj"`.

The chapter entitled Chapter 9, *Transaction management* contains all of the details.

事务管理

2.4.2. Full WebSphere transaction management support

Spring 2.5 explicitly supports IBM's WebSphere Application Server, in particular with respect to WebSphere's transaction manager. Transaction suspension is now fully supported through the use of WebSphere's new `UOWManager` API, which is available on WAS 6.0.2.19+ and 6.0.1.9+.

So if you run a Spring-based application on the WebSphere Application Server, we highly recommend to use Spring 2.5's `WebSphereUowTransactionManager` as your `PlatformTransactionManager` of choice. This is also IBM's official recommendation.

For automatic detection of the underlying JTA-based transaction platform, consider the use of Spring 2.5's new `tx:jta-transaction-manager` configuration element. This will autodetect BEA WebLogic and IBM WebSphere, registering the appropriate `PlatformTransactionManager`.

2.4.3. JPA

Spring 2.0 ships with a JPA abstraction layer that is similar in intent to Spring's JDBC abstraction layer in terms of scope and general usage patterns.

If you are interested in using a JPA-implementation as the backbone of your persistence layer, the section entitled Section 12.6, “JPA” is dedicated to detailing Spring's support and value-add in this area.

Spring 2.5 upgrades its OpenJPA support to OpenJPA 1.0, with support for advanced features such as savepoints.

2.4.4. Asynchronous JMS 异步JMS

Prior to Spring 2.0, Spring's JMS offering was limited to sending messages and the *synchronous* receiving of messages. This functionality (encapsulated in the `JmsTemplate` class) is great, but it doesn't address the requirement for the *asynchronous* receiving of messages.

Spring 2.0 now ships with full support for the reception of messages in an asynchronous fashion, as detailed in the section entitled Section 19.4.2, “Asynchronous Reception - Message-Driven POJOs”.

As of Spring 2.5, the JCA style of setting up asynchronous message listeners is supported as well, through the `GenericMessageEndpointManager` facility. This is an alternative to the standard JMS listener facility, allowing closer integration with message brokers such as ActiveMQ and JORAM. See Section 19.5, “Support for JCA Message Endpoints”.

Spring 2.5 also introduces an XML namespace for simplifying JMS configuration, offering concise configuration of a large numbers of listeners. This namespace supports both the standard JMS listener facility as well as the JCA setup style, with minimal changes in the configuration. See Section 19.6, “JMS Namespace Support”.

2.4.5. JDBC

There are some small (but nevertheless notable) new classes in the Spring Framework's JDBC support library. The first, `NamedParameterJdbcTemplate`, provides support for programming JDBC statements using named parameters (as opposed to programming JDBC statements using only classic placeholder (`'?'`) arguments.

Another of the new classes, the `SimpleJdbcTemplate`, is aimed at making using the `JdbcTemplate` even easier to use when you are developing against Java 5+ (Tiger).

Spring 2.5 significantly extends the functionality of `SimpleJdbcTemplate` and introduces `SimpleJdbcCall` and `SimpleJdbcInsert` operation objects.

2.5. The Web Tier

The web tier support has been *substantially* improved and expanded in Spring 2.0, with annotation-based controllers introduced in Spring 2.5.

2.5.1. Sensible defaulting in Spring MVC

For a lot of projects, sticking to established conventions and having reasonable defaults is just what the projects need... this theme of convention-over-configuration now has explicit support in Spring MVC. What this means is that if you establish a set of naming conventions for your `Controllers` and views, you can *substantially* cut down on the amount of XML configuration that is required to setup handler mappings, view resolvers, `ModelAndView` instances, etc. This is a great boon with regards to rapid prototyping, and can also lend a degree of (always good-to-have) consistency across a codebase.

Spring MVC's convention-over-configuration support is detailed in the section entitled Section 13.10, “Convention over configuration”

2.5.2. Portlet framework

Spring 2.0 ships with a Portlet framework that is conceptually similar to the Spring MVC framework. Detailed coverage of the Spring Portlet framework can be found in the section entitled Chapter 16, *Portlet MVC Framework*.

2.5.3. Annotation-based controllers 基于注解的控制器

2.5引入了用于MVC控制器的基于注解编程模型

Spring 2.5 introduces an annotation-based programming model for MVC controllers, using annotations such as `@RequestMapping`, `@RequestParam`, `@ModelAttribute`, etc. This annotation support is available for both Servlet MVC and Portlet MVC. Controllers implemented in this style do not have to extend specific base classes or implement specific interfaces. Furthermore, they do not usually have direct dependencies on Servlet or Portlet API's, although they can easily get access to Servlet or Portlet facilities if desired. For further details, see Section 13.11, “Annotation-based controller configuration”.

2.5.4. A form tag library for Spring MVC

A rich JSP tag library for Spring MVC was *the* JIRA issue that garnered the most votes from Spring users (by a wide margin).

Spring 2.0 ships with a full featured JSP tag library that makes the job of authoring JSP pages much easier when using Spring MVC; the Spring team is confident it will satisfy all of those developers who voted for the issue on JIRA. The new tag library is itself covered in the section entitled Section 14.2.4, “Using Spring's form tag library”, and a quick reference to all of the new tags can be found in the appendix entitled Appendix E, *spring-form.tld*.

2.5.5. Tiles 2 support

Spring 2.5 ships support for Tiles 2, the next generation of the popular Tiles templating framework. This

supersedes Spring's former support for Tiles 1, as included in Struts 1.x. See Section 14.3, “Tiles” for details.

2.5.6. JSF 1.2 support

Spring 2.5 supports JSF 1.2, providing a JSF 1.2 variant of Spring's `DelegatingVariableResolver` in the form of the new `SpringBeanFacesELResolver`.

2.5.7. JAX-WS support

Spring 2.5 fully supports JAX-WS 2.0/2.1, as included in Java 6 and Java EE 5. JAX-WS is the successor of JAX-RPC, allowing access to WSDL/SOAP-based web services as well as JAX-WS style exposure of web services.

2.6. Everything else

This final section outlines all of the other new and improved Spring 2.0/2.5 features and functionality.

2.6.1. Dynamic language support 动态语言支持

Spring 2.0 introduced support for beans written in languages other than Java, with the currently supported dynamic languages being JRuby, Groovy and BeanShell. This dynamic language support is comprehensively detailed in the section entitled Chapter 24, *Dynamic language support*.

Spring 2.5 refines the dynamic languages support with autowiring and support for the recently released JRuby 1.0.

2.6.2. Enhanced testing support 增强的测试支持

2.5引入了测试上下文框架，其提供注解驱动的单元和集成测试支持

Spring 2.5 introduces the *Spring TestContext Framework* which provides annotation-driven unit and integration testing support that is agnostic of the actual testing framework in use. The same techniques and annotation-based configuration used in, for example, a JUnit 3.8 environment can also be applied to tests written with JUnit 4.4, TestNG, etc.

In addition to providing generic and extensible testing infrastructure, the *Spring TestContext Framework* provides out-of-the-box support for Spring-specific integration testing functionality such as context management and caching, dependency injection of test fixtures, and transactional test management with default rollback semantics.

To discover how this new testing support can assist you with writing unit and integration tests, consult Section 8.3.7, “Spring TestContext Framework” of the revised testing chapter.

2.6.3. JMX support JMX支持

The Spring Framework 2.0 has support for `Notifications`; it is also possible to exercise declarative control over the registration behavior of MBeans with an `MBeanServer`.

- Section 20.7, “Notifications”
- Section 20.2.5, “Controlling the registration behavior”

Furthermore, Spring 2.5 provides a `context:mbean-export` configuration element for convenient registration of annotated bean classes, detecting Spring's `@ManagedResource` annotation.

2.6.4. Deploying a Spring application context as JCA adapter

Spring 2.5 supports the deployment of a Spring application context as JCA resource adapter, packaged as a JCA RAR file. This allows headless application modules to be deployed into J2EE servers, getting access to all the server's infrastructure e.g. for executing scheduled tasks, listening for incoming messages, etc.

2.6.5. Task scheduling 任务调度

Spring 2.0 offers an abstraction around the scheduling of tasks. For the interested developer, the section entitled Section 23.4, “The Spring `TaskExecutor` abstraction” contains all of the details.

The `TaskExecutor` abstraction is used throughout the framework itself as well, e.g. for the asynchronous JMS support. In Spring 2.5, it is also used in the JCA environment support.

2.6.6. Java 5 (Tiger) support

Find below pointers to documentation describing some of the new Java 5 support in Spring 2.0 and 2.5.

- Section 3.11, “Annotation-based configuration” 基于注解的配置
- Section 25.3.1, “`@Required`”
- Section 9.5.6, “Using `@Transactional`” 使用事务注解
- Section 11.2.3, “`SimpleJdbcTemplate`”
- Section 12.6, “JPA”
- Section 6.2, “@AspectJ support” @AspectJ支持
- Section 6.8.1, “Using AspectJ to dependency inject domain objects with Spring”
  使用AspectJ来依赖注入领域对象

2.7. Migrating to Spring 2.5

This final section details issues that may arise during any migration from Spring 1.2/2.0 to Spring 2.5.

Upgrading to Spring 2.5 from a Spring 2.0.x application should simply be a matter of dropping the Spring 2.5 jar into the appropriate location in your application's directory structure. We highly recommend upgrading to Spring 2.5 from any Spring 2.0 application that runs on JDK 1.4.2 or higher, in particular when running on Java 5 or higher, leveraging the significant configuration conveniences and performance improvements that Spring 2.5 has to offer.

Whether an upgrade from Spring 1.2.x will be as seamless depends on how much of the Spring APIs you are using in your code. Spring 2.0 removed pretty much all of the classes and methods previously marked as deprecated in the Spring 1.2.x codebase, so if you have been using such classes and methods, you will of course have to use alternative classes and methods (some of which are summarized below).

With regards to configuration, Spring 1.2.x style XML configuration is 100%, satisfaction-guaranteed

compatible with the Spring 2.5 library. Of course if you are still using the Spring 1.2.x DTD, then you won't be able to take advantage of some of the new Spring 2.0 functionality (such as scopes and easier AOP and transaction configuration), but nothing will blow up.

The suggested migration strategy is to drop in the Spring 2.5 jar(s) to benefit from the improved code present in the release (bug fixes, optimizations, etc.). You can then, on an incremental basis, choose to start using the new Spring 2.5 features and configuration. For example, you could choose to start configuring just your aspects in the new Spring 2 style; it is perfectly valid to have 90% of your configuration using the old-school Spring 1.2.x configuration (which references the 1.2.x DTD), and have the other 10% using the new Spring 2 configuration (which references the 2.0/2.5 DTD or XSD). Bear in mind that you are not forced to upgrade your XML configuration should you choose to drop in the Spring 2.5 libraries.

2.7.1. Changes

For a comprehensive list of changes, consult the `'changelog.txt'` file that is located in the top level directory of the Spring Framework distribution.

2.7.1.1. Supported JDK versions

As of Spring 2.5, support for JDK 1.3 has been removed, following Sun's official deprecation of JDK 1.3 in late 2006. If you haven't done so already, upgrade to JDK 1.4.2 or higher.

If you need to stick with an application server that only supports JDK 1.3, such as WebSphere 4.0 or 5.0, we recommend using the Spring Framework version 2.0.7/2.0.8 which still supports JDK 1.3.

2.7.1.2. Jar packaging in Spring 2.5

As of Spring 2.5, Spring Web MVC is no longer part of the `'spring.jar'` file. Spring MVC can be found in `'spring-webmvc.jar'` and `'spring-webmvc-portlet.jar'` in the `lib/modules` directory of the distribution. Furthermore, the Struts 1.x support has been factored out into `'spring-webmvc-struts.jar'`.

Note: The commonly used Spring's `DispatcherServlet` is part of Spring's Web MVC framework. As a consequence, you need to add `'spring-webmvc.jar'` (or `'spring-webmvc-portlet/struts.jar'`) to a `'spring.jar'` scenario, even if you are just using `DispatcherServlet` for remoting purposes (e.g. exporting Hessian or HTTP invoker services).

Spring 2.0's `'spring-jmx.jar'` and `'spring-remoting.jar'` have been merged into Spring 2.5's `'spring-context.jar'` (for the JMX and non-HTTP remoting support) and partly into `'spring-web.jar'` (for the HTTP remoting support).

Spring 2.0's `'spring-support.jar'` has been renamed to `'spring-context-support.jar'`, expressing the actual support relationship more closely. `'spring-portlet.jar'` has been renamed to `'spring-webmvc-portlet.jar'`, since it is technically a submodule of Spring's Web MVC framework. Analogously, `'spring-struts.jar'` has been renamed to `'spring-webmvc-struts.jar'`.

Spring 2.0's `'spring-jdo.jar'`, `'spring-jpa.jar'`, `'spring-hibernate3.jar'`, `'spring-toplink.jar'` and `'spring-ibatis.jar'` have been combined into Spring 2.5's coarse-granular `'spring-orm.jar'`.

Spring 2.5's `'spring-test.jar'` supersedes the previous `'spring-mock.jar'`, indicating the stronger focus on the test context framework. Note that `'spring-test.jar'` contains everything `'spring-mock.jar'` contained in previous Spring versions; hence it can be used as a straightforward replacement for unit and integration testing purposes.

Spring 2.5's `'spring-tx.jar'` supersedes the previous `'spring-dao.jar'` and `'spring-jca.jar'` files, indicating the stronger focus on the transaction framework.

Spring 2.5 ships its framework jars as OSGi-compliant bundles out of the box. This facilitates use of Spring in OSGi environments, not requiring custom packaging anymore.

2.7.1.3. XML configuration

Spring 2.0 ships with XSDs that describe Spring's XML metadata format in a much richer fashion than the DTD that shipped with previous versions. The old DTD is still fully supported, but if possible you are encouraged to reference the XSD files at the top of your bean definition files.

One thing that has changed in a (somewhat) breaking fashion is the way that bean scopes are defined. If you are using the Spring 1.2 DTD you can continue to use the `'singleton'` attribute. You can however choose to reference the new Spring 2.0 DTD which does not permit the use of the `'singleton'` attribute, but rather uses the `'scope'` attribute to define the bean lifecycle scope.

2.7.1.4. Deprecated classes and methods

A number of classes and methods that previously were marked as `@deprecated` have been removed from the Spring 2.0 codebase. The Spring team decided that the 2.0 release marked a fresh start of sorts, and that any deprecated 'cruft' was better excised now instead of continuing to haunt the codebase for the foreseeable future.

As mentioned previously, for a comprehensive list of changes, consult the `'changelog.txt'` file that is located in the top level directory of the Spring Framework distribution.

The following classes/interfaces have been removed as of Spring 2.0:

- `ResultReader` : Use the `RowMapper` interface instead.
- `BeanFactoryBootstrap` : Consider using a `BeanFactoryLocator` or a custom bootstrap class instead.

2.7.1.5. Apache OJB

As of Spring 2.0, support for Apache OJB was *totally removed* from the main Spring source tree. The Apache OJB integration library is still available, but can be found in its new home in the Spring Modules project.

2.7.1.6. iBATIS

Please note that support for iBATIS SQL Maps 1.3 has been removed. If you haven't done so already, upgrade to iBATIS SQL Maps 2.3.

2.7.1.7. Hibernate

As of Spring 2.5, support for Hibernate 2.1 and Hibernate 3.0 has been removed. If you haven't done so already, upgrade to Hibernate 3.1 or higher.

If you need to stick with Hibernate 2.1 or 3.0 for the time being, we recommend to keep using the Spring Framework version 2.0.7/2.0.8 which still supports those versions of Hibernate.

2.7.1.8. JDO

As of Spring 2.5, support for JDO 1.0 has been removed. If you haven't done so already, upgrade to JDO 2.0 or

higher.

If you need to stick with JDO 1.0 for the time being, we recommend to keep using the Spring Framework version 2.0.7/2.0.8 which still supports that version of JDO.

2.7.1.9. `UrlFilenameViewController`

Since Spring 2.0, the view name that is determined by the `UrlFilenameViewController` now takes into account the nested path of the request. This is a breaking change from the original contract of the `UrlFilenameViewController`, and means that if you are upgrading from Spring 1.x to Spring 2.x and you are using this class you *might* have to change your Spring Web MVC configuration slightly. Refer to the class level Javadocs of the `UrlFilenameViewController` to see examples of the new contract for view name determination.

2.8. Updated sample applications

A number of the sample applications have also been updated to showcase the new and improved features of Spring 2.0. So do take the time to investigate them. The aforementioned sample applications can be found in the `'samples'` directory of the full Spring distribution (`'spring-with-dependencies.[zip|tar.gz]'`).

Spring 2.5 features revised versions of the PetClinic and PetPortal sample applications, reengineered from the ground up for leveraging Spring 2.5's annotation configuration features. It also uses Java 5 autoboxing, generics, varargs and the enhanced for loop. A Java 5 or 6 SDK is now required to build and run the sample. Check out PetClinic and PetPortal to get an impression of what Spring 2.5 has to offer!

2.9. Improved documentation

The Spring reference documentation has also substantially been updated to reflect all of the above features new in Spring 2.0 and 2.5. While every effort has been made to ensure that there are no errors in this documentation, some errors may nevertheless have crept in. If you do spot any typos or even more serious errors, and you can spare a few cycles during lunch, please do bring the error to the attention of the Spring team by raising an issue.

Special thanks to Arthur Loder for his tireless proofreading of the Spring Framework reference documentation and JavaDocs.