Part II. What's New in Spring 3

2. New Features and Enhancements in Spring Framework 3.0

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. It is now time for a third overhaul resulting in Spring Framework 3.0.

第三次大修改

Java SE and Java EE Support

The Spring Framework is now based on Java 5, and Java 6 is fully supported.

Furthermore, Spring is compatible with J2EE 1.4 and Java EE 5, while at the same time introducing some early support for Java EE 6.

2.1 Java 5

The entire framework code has been revised to take advantage of Java 5 features like generics, varargs and other language improvements. We have done our best to still keep the code backwards compatible. We now have consistent use of generic Collections and Maps, consistent use of generic FactoryBeans, and also consistent resolution of bridge methods in the Spring AOP API. Generic ApplicationListeners automatically receive specific event types only. All callback interfaces such as TransactionCallback and HibernateCallback declare a generic result value now. Overall, the Spring core codebase is now freshly revised and optimized for Java 5.

任务执行器抽象框架已集成Java 5的并发工具

Spring's TaskExecutor abstraction has been updated for close integration with Java 5's java.util.concurrent facilities. We provide first-class support for Callables and Futures now, as well as ExecutorService adapters, ThreadFactory integration, etc. This has been aligned with JSR-236 (Concurrency Utilities for Java EE 6) as far as possible. Furthermore, we provide support for asynchronous method invocations through the use of the new @Async annotation (or EJB 3.1's @Asynchronous annotation).

异步方法调用

2.2 Improved documentation

The Spring reference documentation has also substantially been updated to reflect all of the changes and new features for Spring Framework 3.0. 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.

2.3 New articles and tutorials

新的文章和教程

许多优秀的文章和教程

There are many excellent articles and tutorials that show how to get started with Spring Framework 3 features. Read them at the Spring Documentation page.

Learn Spring

The samples have been improved and updated to take advantage of the new features in Spring Framework 3. Additionally, the samples have been moved out of the source tree into a dedicated SVN repository available at:

```
https://anonsvn.springframework.org/svn/spring-samples/
```

As such, the samples are no longer distributed alongside Spring Framework 3 and need to be downloaded separately from the repository mentioned above. However, this documentation will continue to refer to some samples (in particular Petclinic) to illustrate various features.

Note

For more information on Subversion (or in short SVN), see the project homepage at: `http://subversion.apache.org/`

2.4 New module organization and build system 新的模块组织

The framework modules have been revised and are now managed separately with one source-tree per module jar:

- org.springframework.aop
- org.springframework.beans
- org.springframework.context
- org.springframework.context.support
- org.springframework.expression
- org.springframework.instrument
- org.springframework.jdbc
- org.springframework.jms
- org.springframework.orm
- org.springframework.oxm
- org.springframework.test
- org.springframework.transaction
- org.springframework.web
- org.springframework.web.portlet
- org.springframework.web.servlet
- org.springframework.web.struts

Note:

The spring.jar artifact that contained almost the entire framework is no longer provided.

We are now using a new Spring build system as known from Spring Web Flow 2.0. This gives us:

- Ivy-based "Spring Build" system

- consistent deployment procedure
- consistent dependency management
- consistent generation of OSGi manifests

2.5 Overview of new features 新功能概述

This is a list of new features for Spring Framework 3.0. We will cover these features in more detail later in this section. Spring 3.0的新功能

- Spring Expression Language Spring表达式语言
- IoC enhancements/Java based bean metadata 基于Bean元数据的IoC增强
- General-purpose type conversion system and field formatting system 类型转换系统和字段格式化系统
- Object to XML mapping functionality (OXM) moved from Spring Web Services project
- Comprehensive REST support 强大的RESTful支持
- @MVC additions
- Declarative model validation 声明式的模型校验
- Early support for Java EE 6
- Embedded database support

Core APIs updated for Java 5

BeanFactory interface returns typed bean instances as far as possible:

- T getBean(Class<T> requiredType)
- T getBean(String name, Class<T> requiredType)
- Map<String, T> getBeansOfType(Class<T> type)

Spring's TaskExecutor interface now extends `java.util.concurrent.Executor`:

- extended AsyncTaskExecutor supports standard Callables with Futures

New Java 5 based converter API and SPI:

- stateless ConversionService and Converters
- superseding standard JDK PropertyEditors

Typed ApplicationListener<E>

Spring Expression Language Spring表达式语言

Spring introduces an expression language which is similar to Unified EL in its syntax but offers significantly more features. The expression language can be used when defining XML and Annotation

based bean definitions and also serves as the foundation for expression language support across the Spring portfolio. Details of this new functionality can be found in the chapter Spring Expression Language (SpEL).

The Spring Expression Language was created to provide the Spring community a single, well supported expression language that can be used across all the products in the Spring portfolio. Its language features are driven by the requirements of the projects in the Spring portfolio, including tooling requirements for code completion support within the Eclipse based SpringSource Tool Suite.

The following is an example of how the Expression Language can be used to configure some properties of a database setup

```
<bean class="mycompany.RewardsTestDatabase">
    <property name="databaseName"
        value="#{systemProperties.databaseName}"/>
    <property name="keyGenerator"
        value="#{strategyBean.databaseKeyGenerator}"/>
</bean>
```

This functionality is also available if you prefer to configure your components using annotations:

```
@Repository
public class RewardsTestDatabase {

    @Value("#{systemProperties.databaseName}")
    public void setDatabaseName(String dbName) { … }

    @Value("#{strategyBean.databaseKeyGenerator}")
    public void setKeyGenerator(KeyGenerator kg) { … }
}
```

The Inversion of Control (IoC) container

Java based bean metadata 基于Bean元数据的注解

Some core features from the JavaConfig project have been added to the Spring Framework now. This means that the following annotations are now directly supported:

- @Configuration
- @Bean
- @DependsOn
- @Primary
- @Lazy
- @Import
- @ImportResource
- @Value

Here is an example of a Java class providing basic configuration using the new JavaConfig features:

```
package org.example.config;

@Configuration
public class AppConfig {
    private @Value("#{jdbcProperties.url}") String jdbcUrl;
    private @Value("#{jdbcProperties.username}") String username;
    private @Value("#{jdbcProperties.password}") String password;

    @Bean
    public FooService fooService() {
        return new FooServiceImpl(fooRepository());
    }

    @Bean
    public FooRepository fooRepository() {
        return new HibernateFooRepository(sessionFactory());
    }

    @Bean
    public SessionFactory sessionFactory() {
        // wire up a session factory
        AnnotationSessionFactoryBean asFactoryBean =
            new AnnotationSessionFactoryBean();
        asFactoryBean.setDataSource(dataSource());
        // additional config
        return asFactoryBean.getObject();
    }

    @Bean
    public DataSource dataSource() {
        return new DriverManagerDataSource(jdbcUrl, username, password);
    }
}
```

To get this to work you need to add the following component scanning entry in your minimal application context XML file.

```
<context:component-scan base-package="org.example.config"/>
<util:properties id="jdbcProperties" location="classpath:org/example/config/
jdbc.properties"/>
```

Or you can bootstrap a `@Configuration` class directly using `AnnotationConfigApplicationContext`:

```
public static void main(String[] args) {
    ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
    FooService fooService = ctx.getBean(FooService.class);
    fooService.doStuff();
}
```

See the section called “Instantiating the Spring container using `AnnotationConfigApplicationContext`” for full information on `AnnotationConfigApplicationContext`.

Defining bean metadata within components 在组件中定义Bean元数据

`@Bean` annotated methods are also supported inside Spring components. They contribute a factory bean definition to the container. See Defining bean metadata within components for more information

通用的类型转换系统和字段格式化系统

General purpose type conversion system and field formatting system

A general purpose type conversion system has been introduced. The system is currently used by SpEL for type conversion, and may also be used by a Spring Container and DataBinder when binding bean property values.

In addition, a formatter SPI has been introduced for formatting field values. This SPI provides a simpler and more robust alternative to JavaBean PropertyEditors for use in client environments such as Spring MVC.

The Data Tier

Object to XML mapping functionality (OXM) from the Spring Web Services project has been moved to the core Spring Framework now. The functionality is found in the `org.springframework.oxm` package. More information on the use of the `OXM` module can be found in the Marshalling XML using O/X Mappers chapter.

The Web Tier

The most exciting new feature for the Web Tier is the support for building RESTful web services and web applications. There are also some new annotations that can be used in any web application.

Comprehensive REST support 强大的RESTful支持

Server-side support for building RESTful applications has been provided as an extension of the existing annotation driven MVC web framework. Client-side support is provided by the `RestTemplate` class in the spirit of other template classes such as `JdbcTemplate` and `JmsTemplate`. Both server and client side REST functionality make use of `HttpConverters` to facilitate the conversion between objects and their representation in HTTP requests and responses.

The `MarshallingHttpMessageConverter` uses the *Object to XML mapping* functionality mentioned earlier.

Refer to the sections on MVC and the RestTemplate for more information.

@MVC additions 极大地简化了MVC配置

A `mvc` namespace has been introduced that greatly simplifies Spring MVC configuration.

Additional annotations such as `@CookieValue` and `@RequestHeaders` have been added. See Mapping cookie values with the @CookieValue annotation and Mapping request header attributes with the @RequestHeader annotation for more information.

Declarative model validation 声明式地模型校验

Several validation enhancements, including JSR 303 support that uses Hibernate Validator as the default provider.

Early support for Java EE 6 异步方法调用

We provide support for asynchronous method invocations through the use of the new @Async annotation (or EJB 3.1's @Asynchronous annotation).

JSR 303, JSF 2.0, JPA 2.0, etc

Support for embedded databases

Convenient support for embedded Java database engines, including HSQL, H2, and Derby, is now provided.

3. New Features and Enhancements in Spring Framework 3.1

This is a list of new features for Spring Framework 3.1. A number of features do not have dedicated reference documentation but do have complete Javadoc. In such cases, fully-qualified class names are given. See also Appendix C, *Migrating to Spring Framework 3.1*

3.1 Cache Abstraction 缓存抽象

- Chapter 29, *Cache Abstraction*
- Cache Abstraction (SpringSource team blog) Spring 3.1 M1: Cache Abstraction

3.2 Bean Definition Profiles

- XML profiles (SpringSource Team Blog)
- Introducing @Profile (SpringSource Team Blog)
- See org.springframework.context.annotation.Configuration Javadoc
- See org.springframework.context.annotation.Profile Javadoc

3.3 Environment Abstraction

- Environment Abstraction (SpringSource Team Blog)
- See org.springframework.core.env.Environment Javadoc

3.4 PropertySource Abstraction

- Unified Property Management (SpringSource Team Blog)
- See org.springframework.core.env.Environment Javadoc
- See org.springframework.core.env.PropertySource Javadoc
- See org.springframework.context.annotation.PropertySource Javadoc

3.5 Code equivalents for Spring's XML namespaces

Code-based equivalents to popular Spring XML namespace elements <context:component-scan/>, <tx:annotation-driven/> and <mvc:annotation-driven> have been developed, most in the form of `@Enable` annotations. These are designed for use in conjunction with Spring's `@Configuration` classes, which were introduced in Spring Framework 3.0.

- See org.springframework.context.annotation.Configuration Javadoc
- See org.springframework.context.annotation.ComponentScan Javadoc
- See org.springframework.transaction.annotation.EnableTransactionManagement Javadoc
- See org.springframework.cache.annotation.EnableCaching Javadoc

- See org.springframework.web.servlet.config.annotation.EnableWebMvc Javadoc
- See org.springframework.scheduling.annotation.EnableScheduling Javadoc
- See org.springframework.scheduling.annotation.EnableAsync Javadoc
- See org.springframework.context.annotation.EnableAspectJAutoProxy Javadoc
- See org.springframework.context.annotation.EnableLoadTimeWeaving Javadoc
- See org.springframework.beans.factory.aspectj.EnableSpringConfigured Javadoc

3.6 Support for Hibernate 4.x

- See Javadoc for classes within the new org.springframework.orm.hibernate4 package

3.7 TestContext framework support for @Configuration classes and bean definition profiles

The `@ContextConfiguration` annotation now supports supplying `@Configuration` classes for configuring the Spring `TestContext`. In addition, a new `@ActiveProfiles` annotation has been introduced to support declarative configuration of active bean definition profiles in `ApplicationContext` integration tests.

- Spring 3.1 M2: Testing with @Configuration Classes and Profiles (SpringSource Team Blog)
- See the section called “Spring TestContext Framework”
- See the section called “Context configuration with annotated classes” and `org.springframework.test.context.ContextConfiguration` Javadoc
- See `org.springframework.test.context.ActiveProfiles` Javadoc
- See `org.springframework.test.context.SmartContextLoader` Javadoc
- See `org.springframework.test.context.support.DelegatingSmartContextLoader` Javadoc
- See `org.springframework.test.context.support.AnnotationConfigContextLoader` Javadoc

3.8 c: namespace for more concise constructor injection

- the section called “XML shortcut with the c-namespace” `c:`命名空间用于更加精确的构造器注入

3.9 Support for injection against non-standard JavaBeans setters

Prior to Spring Framework 3.1, in order to inject against a property method it had to conform strictly to JavaBeans property signature rules, namely that any 'setter' method must be void-returning. It is now possible in Spring XML to specify setter methods that return any object type. This is useful when considering designing APIs for method-chaining, where setter methods return a reference to 'this'.

3.10 Support for Servlet 3 code-based configuration of Servlet Container

The new `WebApplicationInitializer` builds atop Servlet 3.0's `ServletContainerInitializer` support to provide a programmatic alternative to the traditional web.xml.

- See org.springframework.web.WebApplicationInitializer Javadoc
- Diff from Spring's Greenhouse reference application demonstrating migration from web.xml to `WebApplicationInitializer`

3.11 Support for Servlet 3 MultipartResolver

- See org.springframework.web.multipart.support.StandardServletMultipartResolver Javadoc

3.12 JPA EntityManagerFactory bootstrapping without persistence.xml

In standard JPA, persistence units get defined through `META-INF/persistence.xml` files in specific jar files which will in turn get searched for `@Entity` classes. In many cases, persistence.xml does not contain more than a unit name and relies on defaults and/or external setup for all other concerns (such as the DataSource to use, etc). For that reason, Spring Framework 3.1 provides an alternative: `LocalContainerEntityManagerFactoryBean` accepts a 'packagesToScan' property, specifying base packages to scan for `@Entity` classes. This is analogous to `AnnotationSessionFactoryBean`'s property of the same name for native Hibernate setup, and also to Spring's component-scan feature for regular Spring beans. Effectively, this allows for XML-free JPA setup at the mere expense of specifying a base package for entity scanning: a particularly fine match for Spring applications which rely on component scanning for Spring beans as well, possibly even bootstrapped using a code-based Servlet 3.0 initializer.

3.13 New HandlerMethod-based Support Classes For Annotated Controller Processing 新的基于处理器方法的支持类

Spring Framework 3.1 introduces a new set of support classes for processing requests with annotated controllers:

- `RequestMappingHandlerMapping`
- `RequestMappingHandlerAdapter`
- `ExceptionHandlerExceptionResolver`

These classes are a replacement for the existing:

- `DefaultAnnotationHandlerMapping`
- `AnnotationMethodHandlerAdapter`
- `AnnotationMethodHandlerExceptionResolver`

The new classes were developed in response to many requests to make annotation controller support classes more customizable and open for extension. Whereas previously you could configure a custom annotated controller method argument resolver, with the new support classes you can customize the processing for any supported method argument or return value type.

- See org.springframework.web.method.support.HandlerMethodArgumentResolver Javadoc
- See org.springframework.web.method.support.HandlerMethodReturnValueHandler Javadoc

A second notable difference is the introduction of a `HandlerMethod` abstraction to represent an `@RequestMapping` method. This abstraction is used throughout by the new support classes as the `handler` instance. For example a `HandlerInterceptor` can cast the `handler` from `Object` to `HandlerMethod` and get access to the target controller method, its annotations, etc.

The new classes are enabled by default by the MVC namespace and by Java-based configuration via `@EnableWebMvc`. The existing classes will continue to be available but use of the new classes is recommended going forward.

See the section called “New Support Classes for `@RequestMapping` methods in Spring MVC 3.1” for additional details and a list of features not available with the new support classes.

3.14 "consumes" and "produces" conditions in `@RequestMapping` 消费端和生产端的条件

Improved support for specifying media types consumed by a method through the `'Content-Type'` header as well as for producible types specified through the `'Accept'` header. See the section called “Consumable Media Types” and the section called “Producible Media Types”

3.15 Flash Attributes and `RedirectAttributes`

Flash attributes can now be stored in a `FlashMap` and saved in the HTTP session to survive a redirect. For an overview of the general support for flash attributes in Spring MVC see Section 17.6, “Using flash attributes”.

In annotated controllers, an `@RequestMapping` method can add flash attributes by declaring a method argument of type `RedirectAttributes`. This method argument can now also be used to get precise control over the attributes used in a redirect scenario. See the section called “Specifying redirect and flash attributes” for more details.

3.16 URI Template Variable Enhancements

URI template variables from the current request are used in more places:

- URI template variables are used in addition to request parameters when binding a request to `@ModelAttribute` method arguments.
- @PathVariable method argument values are merged into the model before rendering, except in views that generate content in an automated fashion such as JSON serialization or XML marshalling.
- A redirect string can contain placeholders for URI variables (e.g. `"redirect:/blog/{year}/{month}"`). When expanding the placeholders, URI template variables from the current request are automatically considered.

- An `@ModelAttribute` method argument can be instantiated from a URI template variable provided there is a registered Converter or PropertyEditor to convert from a String to the target object type.

3.17 `@Valid` On `@RequestBody` Controller Method Arguments

用于调用自动校验

An `@RequestBody` method argument can be annotated with `@Valid` to invoke automatic validation similar to the support for `@ModelAttribute` method arguments. A resulting `MethodArgumentNotValidException` is handled in the `DefaultHandlerExceptionResolver` and results in a `400` response code.

3.18 `@RequestPart` Annotation On Controller Method Arguments

This new annotation provides access to the content of a "multipart/form-data" request part. See the section called “Handling a file upload request from programmatic clients” and Section 17.10, “Spring's multipart (file upload) support”.

3.19 `UriComponentsBuilder` and `UriComponents`

URI组件及构建者

A new `UriComponents` class has been added, which is an immutable container of URI components providing access to all contained URI components. A new `UriComponentsBuilder` class is also provided to help create `UriComponents` instances. Together the two classes give fine-grained control over all aspects of preparing a URI including construction, expansion from URI template variables, and encoding.

In most cases the new classes can be used as a more flexible alternative to the existing `UriTemplate` especially since `UriTemplate` relies on those same classes internally.

A `ServletUriComponentsBuilder` sub-class provides static factory methods to copy information from a Servlet request. See Section 17.7, “Building `URIs`”.

4. New Features and Enhancements in Spring Framework 3.2

This section covers what's new in Spring Framework 3.2. See also Appendix D, *Migrating to Spring Framework 3.2*

4.1 Support for Servlet 3 based asynchronous request processing 支持基于异步请求处理的Servlet 3

The Spring MVC programming model now provides explicit Servlet 3 async support. `@RequestMapping` methods can return one of:

- `java.util.concurrent.Callable` to complete processing in a separate thread managed by a task executor within Spring MVC.
- `org.springframework.web.context.request.async.DeferredResult` to complete processing at a later time from a thread not known to Spring MVC — for example, in response to some external event (JMS, AMQP, etc.)
- `org.springframework.web.context.request.async.AsyncTask` to wrap a `Callable` and customize the timeout value or the task executor to use.

See the section called “Asynchronous Request Processing”. 异步请求处理

4.2 Spring MVC Test framework MVC测试框架

First-class support for testing Spring MVC applications with a fluent API and without a Servlet container. Server-side tests involve use of the `DispatcherServlet` while client-side REST tests rely on the `RestTemplate`. See the section called “Spring MVC Test Framework”.

4.3 Content negotiation improvements

A `ContentNegotiationStrategy` is now available for resolving the requested media types from an incoming request. The available implementations are based on the file extension, query parameter, the 'Accept' header, or a fixed content type. Equivalent options were previously available only in the ContentNegotiatingViewResolver but are now available throughout.

`ContentNegotiationManager` is the central class to use when configuring content negotiation options. For more details see the section called “Configuring Content Negotiation”.

The introduction of `ContentNegotiationManger` also enables selective suffix pattern matching for incoming requests. For more details, see the Javadoc of RequestMappingHandlerMapping.setUseRegisteredSuffixPatternMatch.

4.4 @ControllerAdvice annotation

Classes annotated with `@ControllerAdvice` can contain `@ExceptionHandler`, `@InitBinder`, and `@ModelAttribute` methods and those will apply to `@RequestMapping` methods across controller hierarchies as opposed to the controller hierarchy within which they are declared.

`@ControllerAdvice` is a component annotation allowing implementation classes to be auto-detected through classpath scanning.

4.5 Matrix variables

A new `@MatrixVariable` annotation adds support for extracting matrix variables from the request URI. For more details see the section called “Matrix Variables”.

4.6 Abstract base class for code-based Servlet 3+ container initialization

Servlet 3容器初始化抽象基类

An abstract base class implementation of the `WebApplicationInitializer` interface is provided to simplify code-based registration of a DispatcherServlet and filters mapped to it. The new class is named `AbstractDispatcherServletInitializer` and its sub-class `AbstractAnnotationConfigDispatcherServletInitializer` can be used with Java-based Spring configuration. For more details see Section 17.14, “Code-based Servlet container initialization”.

4.7 `ResponseEntityExceptionHandler` class

A convenient base class with an `@ExceptionHandler` method that handles standard Spring MVC exceptions and returns a `ResponseEntity` that allowing customizing and writing the response with HTTP message converters. This serves as an alternative to the `DefaultHandlerExceptionResolver`, which does the same but returns a `ModelAndView` instead.

See the revised Section 17.11, “Handling exceptions” including information on customizing the default Servlet container error page.

4.8 Support for generic types in the `RestTemplate` and in `@RequestBody` arguments

The `RestTemplate` can now read an HTTP response to a generic type (e.g. `List<Account>`). There are three new `exchange()` methods that accept `ParameterizedTypeReference`, a new class that enables capturing and passing generic type info.

In support of this feature, the `HttpMessageConverter` is extended by `GenericHttpMessageConverter` adding a method for reading content given a specified parameterized type. The new interface is implemented by the `MappingJacksonHttpMessageConverter` and also by a new `Jaxb2CollectionHttpMessageConverter` that can read read a generic `Collection` where the generic type is a JAXB type annotated with `@XmlRootElement` or `@XmlType`.

4.9 Jackson JSON 2 and related improvements

The Jackson JSON 2 library is now supported. Due to packaging changes in the Jackson library, there are separate classes in Spring MVC as well. Those are `MappingJackson2HttpMessageConverter` and `MappingJackson2JsonView`. Other related configuration improvements include support for pretty printing as well as a `JacksonObjectMapperFactoryBean` for convenient customization of an `ObjectMapper` in XML configuration.

4.10 Tiles 3

Tiles 3 is now supported in addition to Tiles 2.x. Configuring it should be very similar to the Tiles 2 configuration, i.e. the combination of `TilesConfigurer`, `TilesViewResolver` and `TilesView` except using the `tiles3` instead of the `tiles2` package.

Also note that besides the version number change, the tiles dependencies have also changed. You will need to have a subset or all of `tiles-request-api`, `tiles-api`, `tiles-core`, `tiles-servlet`, `tiles-jsp`, `tiles-el`.

4.11 @RequestBody improvements

An `@RequestBody` or an `@RequestPart` argument can now be followed by an `Errors` argument making it possible to handle validation errors (as a result of an `@Valid` annotation) locally within the `@RequestMapping` method. `@RequestBody` now also supports a required flag.

4.12 HTTP PATCH method

The HTTP request method `PATCH` may now be used in `@RequestMapping` methods as well as in the `RestTemplate` in conjunction with Apache HttpComponents HttpClient version 4.2 or later. The JDK `HttpURLConnection` does not support the `PATCH` method.

4.13 Excluded patterns in mapped interceptors

Mapped interceptors now support URL patterns to be excluded. The MVC namespace and the MVC JavaConfig both expose these options.

4.14 Using meta-annotations for injection points and for bean definition methods

用于元数据注解

As of 3.2, Spring allows for `@Autowired` and `@Value` to be used as meta-annotations, e.g. to build custom injection annotations in combination with specific qualifiers. Analogously, you may build custom `@Bean` definition annotations for `@Configuration` classes, e.g. in combination with specific qualifiers, @Lazy, @Primary, etc.

4.15 Initial support for JCache 0.5

Spring provides a CacheManager adapter for JCache, building against the JCache 0.5 preview release. Full JCache support is coming next year, along with Java EE 7 final.

4.16 Support for `@DateTimeFormat` without Joda Time

The `@DateTimeFormat` annotation can now be used without needing a dependency on the Joda Time library. If Joda Time is not present the JDK `SimpleDateFormat` will be used to parse and print date patterns. When Joda Time is present it will continue to be used in preference to `SimpleDateFormat`.

4.17 Global date & time formatting

全局的日期和时间格式化

It is now possible to define global formats that will be used when parsing and printing date and time types. See Section 7.7, “Configuring a global date & time format” for details.

4.18 New Testing Features 新的测试功能

In addition to the aforementioned inclusion of the Spring MVC Test Framework in the `spring-test` module, the *Spring TestContext Framework* has been revised with support for integration testing web applications as well as configuring application contexts with context initializers. For further details, consult the following.

- Configuring and loading a WebApplicationContext in integration tests
- Configuring context hierarchies in integration tests
- Testing request and session scoped beans
- Improvements to Servlet API mocks
- Configuring test application contexts with ApplicationContextInitializers

4.19 Concurrency refinements across the framework 并发改进

Spring Framework 3.2 includes fine-tuning of concurrent data structures in many parts of the framework, minimizing locks and generally improving the arrangements for highly concurrent creation of scoped/prototype beans.

新的基于Gradle的构建和源码控制移到GitHub上

4.20 New Gradle-based build and move to GitHub

Building and contributing to the framework has never been simpler with our move to a Gradle-based build system and source control at GitHub. See the building from source section of the README and the contributor guidelines for complete details.

4.21 Refined Java SE 7 / OpenJDK 7 support 精简的JDK 7支持

Last but not least, Spring Framework 3.2 comes with refined Java 7 support within the framework as well as through upgraded third-party dependencies: specifically, CGLIB 3.0, ASM 4.0 (both of which come as inlined dependencies with Spring now) and AspectJ 1.7 support (next to the existing AspectJ 1.6 support).