|  |  |  |
| --- | --- | --- |
| **1. Introduction to Spring Framework** | | |
|  |  |  |

**Part I. Overview of Spring Framework**

# Introduction to Spring Framework

Spring Framework یک پلتفرم جاوا است که پشتیبانی زیرساختی جامعی را برای توسعه برنامه های جاوا فراهم می کند. Spring زیرساخت را کنترل می کند تا بتوانید روی برنامه خود تمرکز کنید.

Spring شما را قادر می سازد تا برنامه هایی را از "اشیاء ساده قدیمی جاوا" (POJOs) بسازید و خدمات سازمانی را به صورت غیر تهاجمی برای POJO ها اعمال کنید. این قابلیت برای مدل برنامه نویسی Java SE و برای Java EE کامل و جزئی اعمال می شود.

نمونه هایی از نحوه استفاده شما به عنوان یک توسعه دهنده اپلیکیشن از مزیت پلتفرم Spring:

یک روش جاوا را در یک تراکنش پایگاه داده بدون نیاز به پرداختن به API های تراکنش اجرا کنید.

یک روش جاوا محلی را بدون نیاز به پرداختن به APIهای راه دور، به یک روش راه دور تبدیل کنید.

یک روش محلی جاوا را بدون نیاز به مقابله با APIهای JMX به یک عملیات مدیریت تبدیل کنید.

یک روش جاوا محلی را بدون نیاز به سر و کار داشتن با JMS API به عنوان یک کنترل کننده پیام تبدیل کنید.

## 1.1 Dependency Injection and Inversion of Control

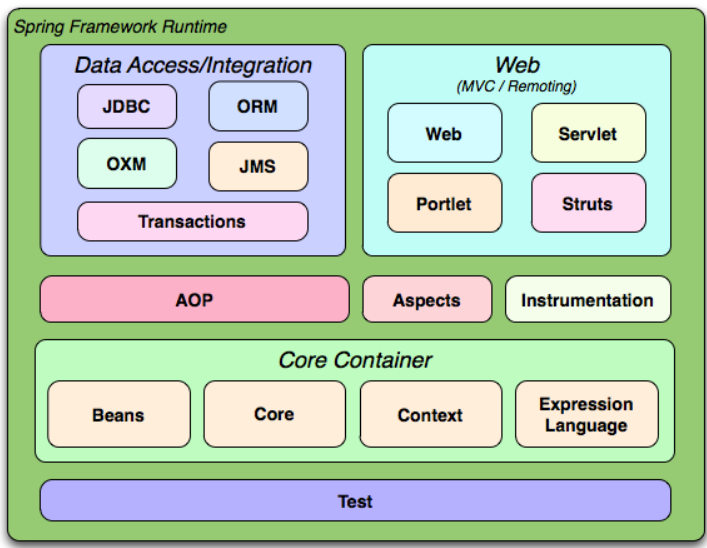
برنامه‌های کاربردی جاوا - یک اصطلاح آزاد که وسعت را از اپلت‌های محدود به برنامه‌های کاربردی سازمانی سمت سرور n-سطح اجرا می‌کند - معمولاً از اشیایی تشکیل می‌شوند که برای تشکیل برنامه مناسب با یکدیگر همکاری می‌کنند. بنابراین اشیا در یک برنامه به یکدیگر وابستگی دارند.

اگرچه پلتفرم جاوا تعداد زیادی از قابلیت‌های توسعه برنامه‌ها را فراهم می‌کند، اما فاقد ابزاری برای سازماندهی بلوک‌های ساختمانی اساسی در یک کل منسجم است و این وظیفه را به معماران و توسعه‌دهندگان واگذار می‌کند. درست است، می‌توانید از الگوهای طراحی مانند Factory، Abstract Factory، Builder، Decorator و Service Locator برای ترکیب کلاس‌ها و نمونه‌های شی مختلف که یک برنامه کاربردی را تشکیل می‌دهند، استفاده کنید. با این حال، این الگوها به سادگی عبارتند از: بهترین شیوه‌هایی که یک نام داده می‌شود، با توضیحی در مورد اینکه الگو چه کاری انجام می‌دهد، کجا باید آن را اعمال کرد، مشکلاتی که به آن پرداخته می‌شود و غیره. الگوها بهترین شیوه های رسمی هستند که باید خودتان در برنامه خود پیاده سازی کنید.

جزء Spring Framework Inversion of Control (IoC) این نگرانی را با ارائه یک ابزار رسمی برای ترکیب اجزای متفاوت در یک برنامه کاربردی کاملاً کارآمد و آماده برای استفاده برطرف می کند. Spring Framework الگوهای طراحی رسمی شده را به عنوان اشیاء درجه یک کدگذاری می کند که می توانید آنها را در برنامه(های) خود ادغام کنید. سازمان‌ها و مؤسسات متعددی از Spring Framework به این روش برای مهندسی برنامه‌های کاربردی قوی و قابل نگهداری استفاده می‌کنند.

## Modules

Spring Framework از ویژگی هایی تشکیل شده است که در حدود 20 ماژول سازماندهی شده اند. این ماژول ها همانطور که در نمودار زیر نشان داده شده است در Core Container، Data Access/Integration، Web، AOP (Aspect Oriented Programming)، Instrumentation و Test گروه بندی می شوند.



## Core Container

Core Container از ماژول های Core، Beans، Context و Expression Language تشکیل شده است.

ماژول‌های Core و Beans بخش‌های اساسی چارچوب، از جمله ویژگی‌های IoC و Dependency Injection را ارائه می‌کنند. BeanFactory یک پیاده سازی پیچیده از الگوی کارخانه است. نیاز به singelton برنامه ای را از بین می برد و به شما امکان می دهد پیکربندی و مشخصات وابستگی ها را از منطق برنامه واقعی خود جدا کنید.

ماژول Context بر پایه پایه محکم ارائه شده توسط ماژول های Core و Beans ساخته شده است: این ابزاری است برای دسترسی به اشیاء به شیوه ای چارچوبی که شبیه به یک رجیستری JNDI است. ماژول Context ویژگی‌های خود را از ماژول Beans به ارث می‌برد و پشتیبانی برای بین‌المللی‌سازی (برای مثال با استفاده از بسته‌های منابع)، انتشار رویداد، بارگذاری منابع، و ایجاد شفاف زمینه‌ها توسط، برای مثال، یک کانتینر servlet اضافه می‌کند. ماژول Context همچنین از ویژگی های Java EE مانند EJB، JMX و کنترل از راه دور اولیه پشتیبانی می کند. رابط ApplicationContext نقطه کانونی ماژول Context است.

ماژول Expression Language یک زبان عبارت قدرتمند برای پرس و جو و دستکاری یک نمودار شی در زمان اجرا فراهم می کند. این یک فرمت از زبان بیان یکپارچه (یکپارچه EL) است که در مشخصات JSP 2.1 مشخص شده است. این زبان از تنظیم و دریافت مقادیر ویژگی، تخصیص ویژگی، فراخوانی روش، دسترسی به زمینه آرایه ها، مجموعه ها و نمایه سازها، عملگرهای منطقی و حسابی، متغیرهای نامگذاری شده و بازیابی اشیاء با نام از ظرف IoC Spring پشتیبانی می کند. همچنین از طرح ریزی و انتخاب لیست و همچنین تجمیع لیست رایج پشتیبانی می کند.

#### Data Access/Integration

لایه دسترسی/ادغام داده ها از ماژول های JDBC، ORM، OXM، JMS و Transaction تشکیل شده است.

ماژول JDBC یک لایه انتزاعی JDBC را فراهم می کند که نیاز به کدگذاری خسته کننده JDBC و تجزیه کدهای خطای خاص فروشنده پایگاه داده را برطرف می کند.

ماژول ORM لایه‌های یکپارچه‌سازی را برای APIهای نگاشت شی رابطه‌ای محبوب از جمله JPA، JDO، Hibernate و iBatis فراهم می‌کند. با استفاده از بسته ORM می‌توانید از همه این چارچوب‌های O/R-mapping در ترکیب با سایر ویژگی‌هایی که Spring ارائه می‌دهد استفاده کنید، مانند ویژگی ساده مدیریت تراکنش اعلامی که قبلاً ذکر شد.

ماژول OXM یک لایه انتزاعی ارائه می‌کند که از پیاده‌سازی نگاشت Object/XML برای JAXB، Castor، XMLBeans، JiBX و XStream پشتیبانی می‌کند.

ماژول سرویس پیام رسانی جاوا (JMS) دارای ویژگی هایی برای تولید و مصرف پیام است.

ماژول Transaction از مدیریت تراکنش های برنامه ای و اعلامی برای کلاس هایی که رابط های ویژه را پیاده سازی می کنند و برای همه POJO های شما (اشیاء ساده قدیمی جاوا) پشتیبانی می کند.

#### Web

لایه وب از ماژول های Web، Web-Servlet، Web-Struts و Web-Portlet تشکیل شده است.

ماژول وب اسپرینگ ویژگی‌های یکپارچه‌سازی مبتنی بر وب را فراهم می‌کند، مانند قابلیت آپلود فایل چند قسمتی و مقداردهی اولیه ظرف IoC با استفاده از شنونده‌های سرولت و زمینه برنامه کاربردی وب‌گرا. همچنین شامل بخش های مربوط به وب پشتیبانی از راه دور Spring است.

ماژول Web-Servlet شامل پیاده سازی Spring's model-view-controller (MVC) برای برنامه های کاربردی وب است. چارچوب MVC Spring یک جداسازی تمیز بین کد مدل دامنه و فرم‌های وب فراهم می‌کند و با سایر ویژگی‌های Spring Framework یکپارچه می‌شود.

ماژول Web-Struts شامل کلاس های پشتیبانی برای ادغام یک لایه وب کلاسیک Struts در یک برنامه Spring است. توجه داشته باشید که این پشتیبانی اکنون از Spring 3.0 منسوخ شده است. مهاجرت برنامه خود را به Struts 2.0 و ادغام Spring آن یا به یک راه حل Spring MVC در نظر بگیرید.

ماژول Web-Portlet پیاده سازی MVC را برای استفاده در محیط پورتلت فراهم می کند و عملکرد ماژول Web-Servlet را منعکس می کند.

#### AOP and Instrumentation

ماژول AOP Spring یک پیاده‌سازی برنامه‌نویسی جنبه‌محور منطبق با AOP Alliance را ارائه می‌کند که به شما امکان می‌دهد، برای مثال، متد-رهگیرها و نقاط برش را برای جدا کردن کدهایی که عملکردهایی را که باید از هم جدا شوند، تعریف کنید. با استفاده از عملکرد metaData سطح منبع، می‌توانید اطلاعات رفتاری را به روشی مشابه ویژگی‌های .NET در کد خود بگنجانید.

ماژول جداگانه Aspects یکپارچه سازی با AspectJ را فراهم می کند.

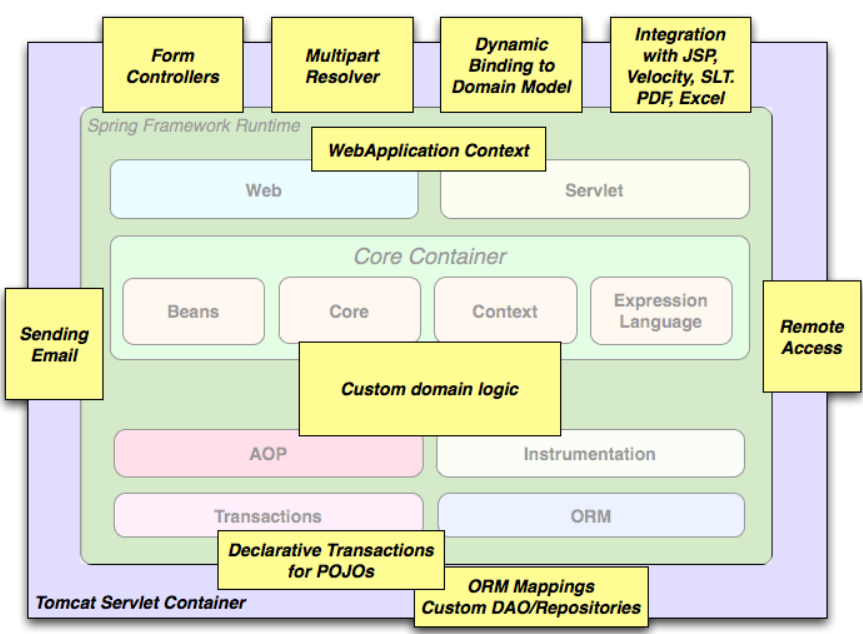
ماژول Instrumentation پشتیبانی از ابزار دقیق کلاس و پیاده سازی کلاس لودر را برای استفاده در سرورهای برنامه خاص فراهم می کند

#### Test

ماژول Test از تست اجزای Spring با JUnit یا TestNG پشتیبانی می کند. بارگیری مداوم Spring ApplicationContexts و ذخیره سازی آن conext ها را فراهم می کند. همچنین اشیاء ساختگی را ارائه می دهد که می توانید از آنها برای آزمایش کد خود به صورت مجزا استفاده کنید.

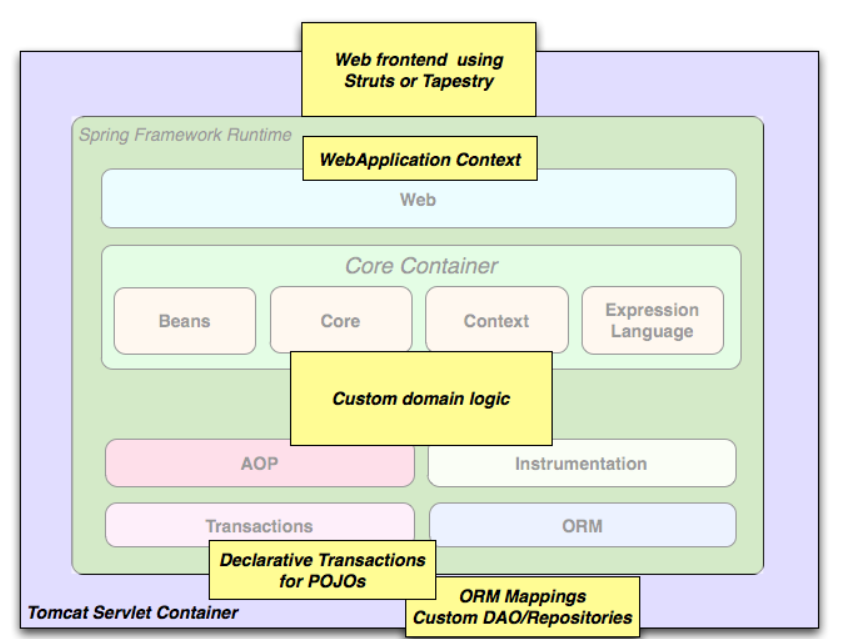
## Usage scenarios

بلوک‌های ساختمانی که قبلاً توضیح داده شد، Spring را در بسیاری از سناریوها، از اپلت‌ها گرفته تا برنامه‌های کاربردی تمام عیار سازمانی که از عملکرد مدیریت تراکنش Spring و یکپارچه‌سازی چارچوب وب استفاده می‌کنند، به یک انتخاب منطقی تبدیل می‌کند.



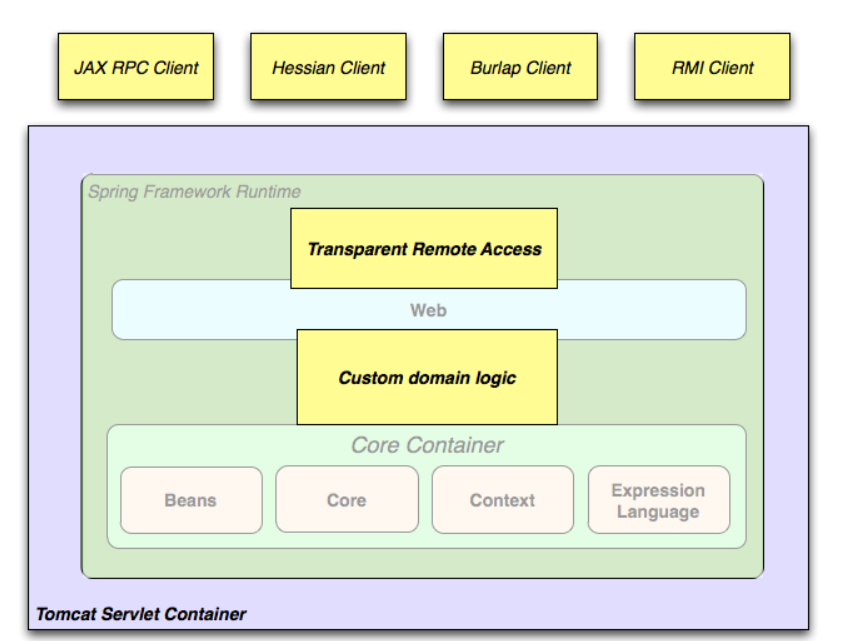
Typical full-fledged Spring web application

ویژگی‌های مدیریت تراکنش‌های declarative Spring، برنامه وب را کاملاً تراکنشی می‌کند، درست همانطور که اگر از تراکنش‌های مدیریت شده توسط EJB استفاده می‌کردید. تمام منطق کسب و کار سفارشی شما را می توان با POJO های ساده پیاده سازی کرد و توسط کانتینر Spring's IoC مدیریت شد. خدمات اضافی شامل پشتیبانی از ارسال ایمیل و تأیید اعتبار مستقل از لایه وب است که به شما امکان می دهد مکان اجرای قوانین اعتبار سنجی را انتخاب کنید. پشتیبانی ORM Spring با JPA، Hibernate، JDO و iBatis یکپارچه شده است. به عنوان مثال، هنگام استفاده از Hibernate، می توانید به استفاده از فایل های نقشه برداری موجود و پیکربندی استاندارد Hibernate SessionFactory ادامه دهید. کنترل‌کننده‌های فرم به‌طور یکپارچه لایه وب را با مدل دامنه ادغام می‌کنند و نیاز به ActionForms یا کلاس‌های دیگر را که پارامترهای HTTP را به مقادیر مدل دامنه شما تبدیل می‌کنند، از بین می‌برند.



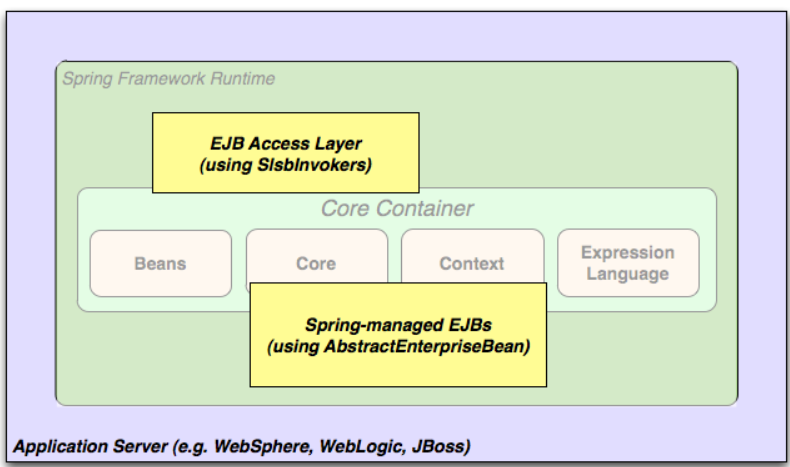
Spring middle-tier using a third-party web framework

گاهی اوقات شرایط به شما اجازه نمی دهد که به طور کامل به یک چارچوب متفاوت تغییر دهید. چارچوب Spring شما را مجبور به استفاده از همه چیز در آن نمی کند. این یک راه حل همه یا هیچ نیست. فریم‌اندهای موجود که با WebWork، Struts، Tapestry یا سایر چارچوب‌های رابط کاربری ساخته شده‌اند، می‌توانند با یک لایه میانی مبتنی بر Spring ادغام شوند، که به شما امکان می‌دهد از ویژگی‌های تراکنش Spring استفاده کنید. شما به سادگی باید منطق کسب و کار خود را با استفاده از ApplicationContext متصل کنید و از WebApplicationContext برای ادغام لایه وب خود استفاده کنید.



Remoting usage scenario

هنگامی که نیاز به دسترسی به کدهای موجود از طریق سرویس های وب دارید، می توانید از کلاس های Hessian-، Burlap-، Rmi- یا JaxRpcProxyFactory Spring's استفاده کنید. فعال کردن دسترسی از راه دور به برنامه های موجود دشوار نیست.



EJBs - Wrapping existing POJOs

Spring Framework همچنین یک لایه دسترسی و انتزاع برای Enterprise JavaBeans فراهم می کند که به شما امکان می دهد از POJO های موجود خود استفاده مجدد کنید و آنها را در Session Beans برای استفاده در برنامه های کاربردی وب مقیاس پذیر و ایمن که ممکن است به امنیت declarative نیاز داشته باشند، بپیچید.

### Dependency Management and Naming Conventions

مدیریت وابستگی و تزریق وابستگی چیزهای متفاوتی هستند. برای دریافت آن ویژگی های خوب Spring در برنامه خود (مانند تزریق وابستگی) باید تمام کتابخانه های مورد نیاز (فایل های jar) را جمع آوری کنید و آنها را در زمان اجرا و احتمالاً در زمان کامپایل در مسیر کلاس خود قرار دهید. این وابستگی ها اجزای مجازی نیستند که تزریق می شوند، بلکه منابع فیزیکی در یک سیستم فایل (معمولا) هستند. فرآیند مدیریت وابستگی شامل مکان یابی آن منابع، ذخیره آنها و افزودن آنها به مسیرهای کلاس است. وابستگی ها می توانند مستقیم باشند (مثلاً برنامه من در زمان اجرا به Spring بستگی دارد) یا غیر مستقیم (مثلاً برنامه من به commons-dbcp بستگی دارد که به commons-pool بستگی دارد). وابستگی‌های غیرمستقیم به عنوان «گذرا» نیز شناخته می‌شوند و این وابستگی‌ها هستند که شناسایی و مدیریت آن‌ها سخت‌تر است.

اگر می‌خواهید از Spring استفاده کنید، باید یک کپی از کتابخانه‌های jar که شامل تکه‌های Spring مورد نیاز شما هستند، تهیه کنید. برای آسان‌تر کردن این کار، Spring به‌عنوان مجموعه‌ای از ماژول‌ها بسته‌بندی می‌شود که وابستگی‌ها را تا حد امکان از هم جدا می‌کنند، بنابراین برای مثال اگر نمی‌خواهید یک برنامه وب بنویسید، به ماژول‌های Spring-Web نیازی ندارید. برای رجوع به ماژول های کتابخانه Spring در این راهنما، از یک قرارداد نامگذاری مختصر Spring-\* یا Spring-\*.jar استفاده می کنیم، که در آن "\*" نشان دهنده نام کوتاه ماژول است (به عنوان مثال Spring-core، Spring-webmvc، Spring-jms، و غیره.). نام واقعی فایل jar که استفاده می‌کنید ممکن است به این شکل باشد (به زیر مراجعه کنید) یا ممکن است نباشد، و معمولاً یک شماره نسخه در نام فایل نیز دارد (به عنوان مثال Spring-core-3.0.0.RELEASE.jar).

به طور کلی، بهار مصنوعات خود را در چهار مکان مختلف منتشر می کند:

در سایت دانلود انجمن http://www.springsource.org/download/community. در اینجا تمام jar‌های Spring را در یک فایل فشرده برای دانلود آسان پیدا می‌کنید. نام jars در اینجا از نسخه 3.0 به شکل org.springframework.\*-<version>.jar است.

Maven Central، که مخزن پیش فرضی است که Maven پرس و جو می کند و برای استفاده نیازی به پیکربندی خاصی ندارد. بسیاری از کتابخانه های رایج که Spring به آنها وابسته است نیز از Maven Central در دسترس هستند و بخش بزرگی از جامعه Spring از Maven برای مدیریت وابستگی استفاده می کنند، بنابراین این برای آنها راحت است. نام jar ها در اینجا به صورت Spring-\*-<version>.jar و Maven groupId org.springframework است.

Enterprise Bundle Repository (EBR) که توسط SpringSource اجرا می شود و همچنین میزبان تمام کتابخانه هایی است که با Spring یکپارچه می شوند. مخازن Maven و Ivy هر دو در اینجا برای تمام JAR‌های Spring و وابستگی‌های آن‌ها در دسترس هستند، به علاوه تعداد زیادی از کتابخانه‌های رایج دیگر که مردم در برنامه‌های کاربردی با Spring از آنها استفاده می‌کنند. هر دو نسخه کامل و همچنین نقاط عطف و تصاویر فوری توسعه در اینجا مستقر شده اند. نام فایل‌های jar به همان شکل دانلود انجمن است (org.springframework.\*-<version>.jar)، و وابستگی‌ها نیز به این شکل طولانی هستند، با کتابخانه‌های خارجی (نه از SpringSource) پیشوند com.springsource. برای اطلاعات بیشتر به سوالات متداول مراجعه کنید. در یک مخزن عمومی Maven که در آمازون S3 میزبانی می‌شود، برای عکس‌های فوری توسعه و انتشارات نقطه عطف (یک کپی از نسخه‌های نهایی نیز در اینجا نگهداری می‌شود). نام فایل‌های jar به شکل Maven Central هستند، بنابراین این مکان مفیدی برای دریافت نسخه‌های توسعه‌دهنده Spring برای استفاده با سایر کتابخانه‌های مستقر در Maven Central است.

بنابراین اولین چیزی که باید تصمیم بگیرید این است که چگونه وابستگی های خود را مدیریت کنید: اکثر مردم از یک سیستم خودکار مانند Maven یا Ivy استفاده می کنند، اما شما می توانید این کار را به صورت دستی با دانلود همه JAR ها انجام دهید. هنگام به دست آوردن اسپرینگ با Maven یا Ivy باید تصمیم بگیرید که از کدام مکان آن را تهیه کنید. به طور کلی، اگر به OSGi اهمیت می دهید، از EBR استفاده کنید، زیرا مصنوعات سازگار با OSGi را برای همه وابستگی های Spring مانند Hibernate و Freemarker در خود جای داده است. اگر OSGi برای شما اهمیتی ندارد، هر کدام از این مکان‌ها کار می‌کنند، هرچند مزایا و معایبی بین آنها وجود دارد. به طور کلی، یک مکان یا مکان دیگر را برای پروژه خود انتخاب کنید. آنها را مخلوط نکنید این امر به ویژه مهم است زیرا مصنوعات EBR لزوماً از نامگذاری متفاوتی نسبت به مصنوعات Maven Central استفاده می کنند.

**Table 1.1. Comparison of Maven Central and SpringSource EBR Repositories**

| **Feature** | **Maven Central** | **EBR** |
| --- | --- | --- |
| OSGi Compatible | Not explicit | Yes |
| Number of Artifacts | Tens of thousands; all kinds | Hundreds; those that Spring integrates with |
| Consistent Naming Conventions | No | Yes |
| Naming Convention: GroupId | Varies. Newer artifacts often use domain name, e.g. org.slf4j. Older ones often just use the artifact name, e.g. log4j. | Domain name of origin or main package root,  e.g. org.springframework |
| Naming Convention: ArtifactId | Varies. Generally the project or module name, using a hyphen "-" separator, e.g. spring-core, logj4. | Bundle Symbolic Name, derived from the main package root,  e.g. org.springframework.beans. If the jar had to be patched to  ensure OSGi compliance then com.springsource is appended,  e.g. com.springsource.org.apache.log4j |
| Naming Convention: Version | Varies. Many new artifacts use m.m.m or m.m.m.X (with m=digit, X=text). Older ones use m.m. Some neither. Ordering is defined but not often relied on, so not strictly reliable. | OSGi version number m.m.m.X, e.g. 3.0.0.RC3. The  text qualifier imposes alphabetic ordering on versions with  the same numeric values. |
| Publishing | Usually automatic via rsync or source control updates. Project authors can upload individual jars to JIRA. | Manual (JIRA processed by SpringSource) |
| Quality Assurance | By policy. Accuracy is responsibility of authors. | Extensive for OSGi manifest, Maven POM and Ivy metadata.  QA performed by Spring team. |
| Hosting | Contegix. Funded by Sonatype with several mirrors. | S3 funded by SpringSource. |
| Search Utilities | Various | <http://www.springsource.com/repository> |
| Integration with SpringSource Tools | Integration through STS with Maven dependency management | Extensive integration through STS with Maven, Roo,  CloudFoundry |

#### Spring Dependencies and Depending on Spring

اگرچه Spring یکپارچه سازی و پشتیبانی را برای طیف عظیمی از سازمانی و سایر ابزارهای خارجی فراهم می کند، اما عمداً وابستگی های اجباری خود را به حداقل می رساند: برای اینکه بتوانید تعداد زیادی کتابخانه jar را پیدا و دانلود کنید (حتی به طور خودکار) لازم نیست. از Spring برای موارد استفاده ساده استفاده کنید. برای تزریق وابستگی اولیه تنها یک وابستگی خارجی اجباری وجود دارد، و آن برای logging است (برای توضیحات بیشتر در مورد گزینه‌های logging به زیر مراجعه کنید).

در ادامه مراحل اولیه مورد نیاز برای پیکربندی برنامه‌ای که به Spring بستگی دارد، ابتدا با Maven و سپس با Ivy شرح می‌دهیم. در همه موارد، اگر چیزی نامشخص است، به مستندات سیستم مدیریت وابستگی خود مراجعه کنید، یا به برخی از کدهای نمونه نگاه کنید - اسپرینگ خودش از Ivy برای مدیریت وابستگی ها هنگام ساخت استفاده می کند و نمونه های ما بیشتر از Maven استفاده می کنند.

#### Maven Dependency Management

اگر از Maven برای مدیریت وابستگی استفاده می کنید، حتی نیازی به ارائه وابستگی logging ندارید. به عنوان مثال، برای ایجاد یک زمینه برنامه و استفاده از تزریق وابستگی برای پیکربندی یک برنامه، وابستگی های Maven شما به شکل زیر خواهد بود:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>3.0.0.RELEASE</version>

<scope>runtime</scope>

</dependency>

</dependencies>

خودشه. توجه داشته باشید که اگر نیازی به کامپایل در برابر Spring API ندارید، می‌توان دامنه را به‌عنوان زمان اجرا اعلام کرد، که معمولاً برای موارد استفاده از تزریق وابستگی اولیه صادق است.

ما از قراردادهای نامگذاری Maven Central در مثال بالا استفاده کردیم، به طوری که با Maven Central یا مخزن SpringSource S3 Maven کار می کند. برای استفاده از مخزن S3 Maven (به عنوان مثال برای نقاط عطف یا عکس‌های فوری برنامه‌نویس)، باید مکان مخزن را در پیکربندی Maven خود مشخص کنید. برای نسخه های کامل:

<repositories>

<repository>

<id>com.springsource.repository.maven.release</id>

<url>http://repo.springsource.org/release/</url>

<snapshots><enabled>false</enabled></snapshots>

</repository>

</repositories>

برای milestone:

<repositories>

<repository>

<id>com.springsource.repository.maven.milestone</id>

<url>http://repo.springsource.org/milestone/</url>

<snapshots><enabled>false</enabled></snapshots>

</repository>

</repositories>

برای snapshots:

<repositories>

<repository>

<id>com.springsource.repository.maven.snapshot</id>

<url>http://repo.springsource.org/snapshot/</url>

<snapshots><enabled>true</enabled></snapshots>

</repository>

</repositories>

برای استفاده از SpringSource EBR باید از یک قرارداد نامگذاری متفاوت برای وابستگی ها استفاده کنید. حدس زدن نام ها معمولا آسان است، به عنوان مثال. در این مورد این است:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>org.springframework.context</artifactId>

<version>3.0.0.RELEASE</version>

<scope>runtime</scope>

</dependency>

</dependencies>

همچنین باید مکان مخزن را به صراحت اعلام کنید (فقط URL مهم است):

<repositories>

<repository>

<id>com.springsource.repository.bundles.release</id>

<url>http://repository.springsource.com/maven/bundles/release/</url>

</repository>

</repositories>

اگر وابستگی‌های خود را به صورت دستی مدیریت می‌کنید، URL موجود در اعلامیه مخزن بالا قابل مرور نیست، اما یک رابط کاربری در http://www.springsource.com/repository وجود دارد که می‌توان از آن برای جستجو و دانلود وابستگی‌ها استفاده کرد. همچنین دارای قطعات مفیدی از پیکربندی Maven و Ivy است که می توانید در صورت استفاده از آن ابزارها را کپی و جایگذاری کنید.

#### Ivy Dependency Management

اگر ترجیح می دهید از Ivy برای مدیریت وابستگی ها استفاده کنید، نام ها و گزینه های پیکربندی مشابهی وجود دارد.

برای پیکربندی Ivy برای اشاره به SpringSource EBR حل کننده های زیر را به ivysettings.xml خود اضافه کنید:

<resolvers>

<url name="com.springsource.repository.bundles.release">

<ivy pattern="http://repository.springsource.com/ivy/bundles/release/

[organisation]/[module]/[revision]/[artifact]-[revision].[ext]" />

<artifact pattern="http://repository.springsource.com/ivy/bundles/release/

[organisation]/[module]/[revision]/[artifact]-[revision].[ext]" />

</url>

<url name="com.springsource.repository.bundles.external">

<ivy pattern="http://repository.springsource.com/ivy/bundles/external/

[organisation]/[module]/[revision]/[artifact]-[revision].[ext]" />

<artifact pattern="http://repository.springsource.com/ivy/bundles/external/

[organisation]/[module]/[revision]/[artifact]-[revision].[ext]" />

</url>

</resolvers>

XML بالا معتبر نیست زیرا خطوط بیش از حد طولانی هستند - اگر کپی-پیست کنید، انتهای خطوط اضافی را در وسط الگوهای آدرس اینترنتی حذف کنید.

هنگامی که Ivy پیکربندی شد تا در EBR جستجو کند، افزودن یک وابستگی آسان است. به سادگی صفحه جزئیات بسته مورد نظر را در مرورگر مخزن بالا بکشید و یک قطعه پیچک آماده برای اضافه کردن در بخش وابستگی‌ها خواهید یافت. به عنوان مثال (در ivy.xml):

<dependency org="org.springframework"

name="org.springframework.core" rev="3.0.0.RELEASE" conf="compile->runtime"/>

### Logging

Logging یک وابستگی بسیار مهم برای Spring است زیرا الف) تنها وابستگی خارجی اجباری است، ب) همه دوست دارند برخی از خروجی ها را از ابزارهایی که استفاده می کنند ببینند، و ج) Spring با بسیاری از ابزارهای دیگر ادغام می شود که همه آنها نیز ساخته شده اند. انتخاب وابستگی logging. یکی از اهداف یک توسعه‌دهنده برنامه اغلب این است که گزارش‌گیری یکپارچه را در یک مکان مرکزی برای کل برنامه، از جمله تمام اجزای خارجی، پیکربندی کند. این دشوارتر از آن چیزی است که ممکن است باشد، زیرا انتخاب های زیادی برای logging framework وجود دارد.

وابستگی اجباری logging در بهار، Jakarta Commons Logging API (JCL) است. ما در برابر JCL کامپایل می کنیم و همچنین اشیاء JCL Log را برای کلاس هایی که چارچوب Spring را گسترش می دهند قابل مشاهده می کنیم. برای کاربران مهم است که همه نسخه‌های Spring از یک کتابخانه گزارش‌گیری استفاده کنند: مهاجرت آسان است زیرا سازگاری به عقب حتی با برنامه‌هایی که Spring را گسترش می‌دهند حفظ می‌شود. روشی که ما این کار را انجام می‌دهیم این است که یکی از ماژول‌ها را در Spring به طور صریح به Commons-logging (پیاده‌سازی متعارف JCL) وابسته کنیم و سپس همه ماژول‌های دیگر را در زمان کامپایل به آن وابسته کنیم. اگر برای مثال از Maven استفاده می‌کنید، و نمی‌دانید که وابستگی به logging مشترک را از کجا دریافت کرده‌اید، پس این وابستگی از Spring و به طور خاص از ماژول مرکزی به نام Spring-core است.

نکته خوب در مورد Commons-logging این است که برای کارکرد برنامه خود به هیچ چیز دیگری نیاز ندارید. این یک الگوریتم کشف زمان اجرا دارد که به دنبال سایر چارچوب‌های گزارش‌گیری در مکان‌های شناخته شده در مسیر کلاس می‌گردد و از یکی که فکر می‌کند مناسب است استفاده می‌کند (یا می‌توانید در صورت نیاز به آن بگویید کدام یک). اگر هیچ چیز دیگری در دسترس نباشد، فقط از JDK (java.util.logging یا به اختصار JUL) لاگ‌های زیبا به‌نظر می‌رسد. باید متوجه شوید که برنامه Spring شما در اکثر مواقع به خوبی کار می کند و به راحتی به کنسول وارد می شود و این مهم است.

**Not Using Commons Logging**

متأسفانه، الگوریتم کشف زمان اجرا commons-logging، در حالی که برای کاربر نهایی مناسب است، مشکل ساز است. اگر می‌توانستیم ساعت را به عقب برگردانیم و Spring را اکنون به عنوان یک پروژه جدید شروع کنیم، از وابستگی گزارش متفاوتی استفاده می‌کردیم. اولین انتخاب احتمالاً Simple Logging Facade برای جاوا (SLF4J) است که توسط بسیاری از ابزارهای دیگر که افراد با Spring در داخل برنامه های خود استفاده می کنند نیز استفاده می شود.

خاموش کردن Commons-logging آسان است: فقط مطمئن شوید که در زمان اجرا در مسیر کلاس نباشد. در اصطلاح Maven شما وابستگی را حذف می کنید و به دلیل نحوه اعلان وابستگی های Spring، فقط یک بار باید این کار را انجام دهید.

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>3.0.0.RELEASE</version>

<scope>runtime</scope>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

اکنون این برنامه احتمالاً خراب است زیرا هیچ پیاده سازی API JCL در مسیر کلاس وجود ندارد، بنابراین برای رفع آن باید یک برنامه جدید ارائه شود. در بخش بعدی به شما نشان می دهیم که چگونه می توانید یک پیاده سازی جایگزین از JCL با استفاده از SLF4J به عنوان مثال ارائه دهید.

#### Using SLF4J

SLF4J یک وابستگی تمیزتر و در زمان اجرا کارآمدتر از Commons-logging است زیرا از اتصالات زمان کامپایل به جای کشف زمان اجرا سایر چارچوب‌های گزارش‌گیری که ادغام می‌کند استفاده می‌کند. این همچنین به این معنی است که شما باید در مورد آنچه می خواهید در زمان اجرا اتفاق بیفتد واضح تر باشید و آن را اعلام کنید یا بر اساس آن پیکربندی کنید. SLF4J اتصال به بسیاری از چارچوب‌های گزارش‌گیری رایج را فراهم می‌کند، بنابراین معمولاً می‌توانید یکی را که قبلاً استفاده می‌کنید انتخاب کنید و برای پیکربندی و مدیریت به آن متصل شوید.

SLF4J اتصال به بسیاری از چارچوب‌های گزارش‌گیری رایج، از جمله JCL را فراهم می‌کند، و همچنین برعکس را انجام می‌دهد: پل‌هایی بین سایر چارچوب‌های گزارش‌گیری و خودش. بنابراین برای استفاده از SLF4J با Spring، باید وابستگی Commons-logging را با پل SLF4J-JCL جایگزین کنید. هنگامی که این کار را انجام دادید، تماس‌های ثبت‌نام از داخل Spring به تماس‌های ورود به سیستم API SLF4J ترجمه می‌شوند، بنابراین اگر کتابخانه‌های دیگر در برنامه شما از آن API استفاده می‌کنند، شما یک مکان واحد برای پیکربندی و مدیریت logging خواهید داشت.

یک انتخاب رایج ممکن است این باشد که Spring را به SLF4J متصل کنید، و سپس اتصال صریح از SLF4J به Log4J را ارائه دهید. شما باید 4 وابستگی را تامین کنید (و از اشتراک‌گذاری‌های موجود حذف کنید): پل، API SLF4J، اتصال به Log4J و خود پیاده‌سازی Log4J. در Maven شما این کار را به این صورت انجام می دهید

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>3.0.0.RELEASE</version>

<scope>runtime</scope>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>1.5.8</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.5.8</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.5.8</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.14</version>

<scope>runtime</scope>

</dependency>

</dependencies>

این ممکن است وابستگی های زیادی به نظر برسد فقط برای گرفتن گزارش. خوب است، اما اختیاری است، و باید بهتر از ثبت نام وانیلیcommons-logging با توجه به مشکلات کلاس لودر رفتار کند، به ویژه اگر در یک ظرف سخت مانند یک پلت فرم OSGi هستید. ظاهراً یک مزیت عملکرد نیز وجود دارد زیرا اتصالات در زمان کامپایل هستند و نه در زمان اجرا.

انتخاب رایج‌تر در میان کاربران SLF4J، که از مراحل کمتری استفاده می‌کند و وابستگی‌های کمتری ایجاد می‌کند، اتصال مستقیم به Logback است. این مرحله اتصال اضافی را حذف می‌کند زیرا Logback مستقیماً SLF4J را پیاده‌سازی می‌کند، بنابراین شما فقط باید به دو کتابخانه وابسته باشید نه چهار (jcl-over-slf4j و logback). اگر این کار را انجام دهید، ممکن است لازم باشد وابستگی slf4j-api را از سایر وابستگی‌های خارجی (نه Spring) حذف کنید، زیرا فقط یک نسخه از آن API را در مسیر کلاس می‌خواهید.

**Using Log4J**

بسیاری از افراد از Log4j به عنوان یک logging framework برای اهداف پیکربندی و مدیریت استفاده می کنند. این کارآمد و به خوبی تثبیت شده است، و در واقع همان چیزی است که در زمان اجرا هنگام ساخت و آزمایش Spring از آن استفاده می کنیم. Spring همچنین برخی از ابزارهای کاربردی را برای پیکربندی و مقداردهی اولیه Log4j فراهم می کند، بنابراین در برخی ماژول ها وابستگی زمان کامپایل اختیاری به Log4j دارد.

برای اینکه Log4j با وابستگی پیش فرض JCL (commons-logging) کار کند، تنها کاری که باید انجام دهید این است که Log4j را در مسیر کلاس قرار دهید و یک فایل پیکربندی (log4j.properties یا log4j.xml در ریشه classpath) برای آن ارائه دهید. بنابراین برای کاربران Maven این اعلامیه وابستگی شما است

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>3.0.0.RELEASE</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.14</version>

<scope>runtime</scope>

</dependency>

</dependencies>

و در اینجا یک نمونه log4j.properties برای ورود به کنسول وجود دارد:

log4j.rootCategory=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %t %c{2}:%L - %m%n

log4j.category.org.springframework.beans.factory=DEBUG

##### Runtime Containers with Native JCL3

بسیاری از مردم برنامه‌های Spring خود را در ظرفی اجرا می‌کنند که خود اجرای JCL را ارائه می‌کند. IBM Websphere Application Server (WAS) کهن الگو است. این اغلب باعث مشکلاتی می شود و متأسفانه هیچ راه حل گلوله نقره ای وجود ندارد. صرفاً کنار گذاشتن commons-loggingاز برنامه شما در اکثر مواقع کافی نیست.

برای روشن شدن این موضوع: مشکلات گزارش شده معمولاً به خودی خود مربوط به JCL یا حتی با ثبت نام مشترک نیستند: بلکه مربوط به اتصال ثبت مشترکات به چارچوب دیگری (اغلب Log4J) است. این ممکن است شکست بخورد زیرا ثبت نام مشترک روشی را که در زمان اجرا بین نسخه‌های قدیمی (1.0) موجود در برخی از کانتینرها و نسخه‌های مدرنی که اکثر مردم اکنون استفاده می‌کنند (1.1) را تغییر داده است. Spring از هیچ بخش غیرمعمولی از JCL API استفاده نمی کند، بنابراین هیچ چیز در آنجا خراب نمی شود، اما به محض اینکه Spring یا برنامه شما سعی می کند هر گونه logging را انجام دهد، می توانید متوجه شوید که اتصالات به Log4J کار نمی کنند.

در چنین مواردی با WAS ساده‌ترین کار این است که سلسله مراتب کلاس بارگذار را وارونه کنید (IBM آن را "آخرین والد" می‌نامد) تا برنامه وابستگی JCL را کنترل کند، نه کانتینر. این گزینه همیشه باز نیست، اما بسیاری از پیشنهادات دیگر در حوزه عمومی برای رویکردهای جایگزین وجود دارد، و مسافت پیموده شده شما ممکن است بسته به نسخه دقیق و مجموعه ویژگی‌های ظرف متفاوت باشد.

# 2. New Features and Enhancements in Spring Framework 3.0

اگر مدتی است که از Spring Framework استفاده می‌کنید، می‌دانید که Spring تحت دو بازنگری اصلی قرار گرفته است: Spring 2.0، که در اکتبر 2006 منتشر شد، و Spring 2.5، که در نوامبر 2007 منتشر شد. اکنون زمان برای سومین بازنگری است. در Spring Framework 3.0.

Java SE و Java EE پشتیبانی می کند

Spring Framework اکنون بر اساس جاوا 5 است و جاوا 6 به طور کامل پشتیبانی می شود.

علاوه بر این، Spring با J2EE 1.4 و Java EE 5 سازگار است، در حالی که در همان زمان برخی از پشتیبانی اولیه را برای Java EE 6 ارائه می کند.

## 2.1 Java 5

کل کد چارچوب برای استفاده از ویژگی‌های جاوا 5 مانند ژنریک، varargs و سایر پیشرفت‌های زبانی اصلاح شده است. ما تمام تلاش خود را انجام داده‌ایم تا همچنان کد را با هم سازگار نگه داریم. ما اکنون استفاده مداوم از مجموعه‌ها و نقشه‌های عمومی، استفاده مداوم از FactoryBeans عمومی، و همچنین وضوح ثابت روش‌های bridge در Spring AOP API داریم. Generic ApplicationListeners به ​​طور خودکار فقط انواع خاصی از رویداد را دریافت می کند. همه واسط های تماس برگشتی مانند TransactionCallback و HibernateCallback اکنون یک مقدار نتیجه عمومی را اعلام می کنند. به طور کلی، پایگاه کد هسته Spring اکنون به تازگی برای جاوا 5 اصلاح و بهینه شده است.

چکیده Spring's TaskExecutor برای ادغام نزدیک با امکانات java.util.concurrent جاوا 5 به روز شده است. ما اکنون پشتیبانی درجه یک برای Calables و Futures و همچنین آداپتورهای ExecutorService، ادغام ThreadFactory و غیره ارائه می‌کنیم. علاوه بر این، ما از طریق استفاده از حاشیه نویسی @Async جدید (یا حاشیه نویسی Asynchronous@ EJB 3.1) از فراخوانی روش ناهمزمان پشتیبانی می کنیم.

## 2.2 Improved documentation

مستندات مرجع Spring همچنین بطور قابل توجهی به روز شده است تا همه تغییرات و ویژگی های جدید Spring Framework 3.0 را منعکس کند. در حالی که تمام تلاش‌ها برای اطمینان از عدم وجود خطایی در این مستندات انجام شده است، اما ممکن است برخی از اشتباهات وارد شده باشد. اگر اشتباهات تایپی یا حتی خطاهای جدی‌تری را مشاهده کردید، و می‌توانید در طول ناهار از چند چرخه صرف نظر کنید، لطفاً آن را به همراه داشته باشید. خطایی که با طرح موضوع مورد توجه تیم بهار قرار گرفت.

## 2.3 New articles and tutorials

مقالات و آموزش های بسیار خوبی وجود دارد که نحوه شروع کار با ویژگی های Spring Framework 3 را نشان می دهد. آنها را در صفحه مستندات بهار بخوانید.

نمونه ها برای بهره مندی از ویژگی های جدید در Spring Framework 3 بهبود یافته و به روز شده اند. علاوه بر این، نمونه ها به خارج از درخت منبع به یک مخزن اختصاصی SVN منتقل شده اند که در آدرس زیر موجود است:

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

به این ترتیب، نمونه ها دیگر در کنار Spring Framework 3 توزیع نمی شوند و باید جداگانه از مخزن ذکر شده در بالا دانلود شوند. با این حال، این مستندات همچنان به برخی از نمونه ها (به ویژه Petclinic) برای نشان دادن ویژگی های مختلف اشاره می کند.

## 2.4 New module organization and build system

ماژول های فریم ورک بازنگری شده اند و اکنون به طور جداگانه با یک منبع درخت در هر jar مدول مدیریت می شوند:

* 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

مصنوع Spring.jar که تقریباً کل چارچوب را در بر می گرفت، دیگر ارائه نشده است.

ما اکنون از یک سیستم ساخت Spring جدید به نام Spring Web Flow 2.0 استفاده می کنیم. این به ما می دهد:

* Ivy-based "Spring Build" system
* consistent deployment procedure
* consistent dependency management
* consistent generation of OSGi manifests

## 2.5 Overview of new features

این لیستی از ویژگی های جدید Spring Framework 3.0 است. در ادامه این بخش به جزئیات بیشتر این ویژگی ها خواهیم پرداخت.

* Spring Expression Language
* IoC enhancements/Java based bean metadata
* General-purpose type conversion system and field formatting system
* Object to XML mapping functionality (OXM) moved from Spring Web Services project
* Comprehensive REST support
* @MVC additions
* Declarative model validation
* Early support for Java EE 6
* Embedded database support

### 2.5.1 Core APIs updated for Java 5

رابط BeanFactory نمونه‌هایtype bean را تا آنجا که ممکن است برمی‌گرداند:

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

رابط TaskExecutor Spring اکنون java.util.concurrent.Executor را گسترش می دهد:

توسعه یافته AsyncTaskExecutor از Calables استاندارد با Futures پشتیبانی می کند

API و SPI مبدل جدید مبتنی بر جاوا 5:

ConversionService و Converters بدون تابعیت

جایگزین JDK PropertyEditors استاندارد

تایپ شده ApplicationListener<E>

### 2.5.2 Spring Expression Language

Spring یک زبان عبارتی را معرفی می کند که از نظر نحو شبیه به Unified EL است اما ویژگی های قابل توجهی بیشتری را ارائه می دهد. زبان عبارت را می توان در هنگام تعریف XML و تعاریف Bean مبتنی بر Annotation استفاده کرد و همچنین به عنوان پایه ای برای پشتیبانی از زبان بیان در سراسر مجموعه Spring عمل می کند. جزئیات این قابلیت جدید را می توان در فصل Spring Expression Language (SpEL) یافت.

زبان بیان بهار برای ارائه یک زبان بیانی واحد با پشتیبانی خوب برای جامعه Spring ایجاد شده است که می تواند در همه محصولات موجود در مجموعه Spring استفاده شود. ویژگی‌های زبان آن بر اساس الزامات پروژه‌های مجموعه Spring، از جمله الزامات ابزار برای پشتیبانی از تکمیل کد در مجموعه ابزار SpringSource مبتنی بر Eclipse، هدایت می‌شود.

در زیر مثالی از نحوه استفاده از Expression Language برای پیکربندی برخی از ویژگی های تنظیمات پایگاه داده آورده شده است

<bean class="mycompany.RewardsTestDatabase">

<property name="databaseName"

value="#{systemProperties.databaseName}"/>

<property name="keyGenerator"

value="#{strategyBean.databaseKeyGenerator}"/>

</bean>

اگر ترجیح می دهید اجزای خود را با استفاده از حاشیه نویسی پیکربندی کنید، این عملکرد نیز در دسترس است:

*@Repository*

**public** **class** RewardsTestDatabase {

*@Value("#{systemProperties.databaseName}")*

**public** **void** setDatabaseName(String dbName) { … }

*@Value("#{strategyBean.databaseKeyGenerator}")*

**public** **void** setKeyGenerator(KeyGenerator kg) { … }

}

### 2.5.3 The Inversion of Control (IoC) container

#### Java based bean metadata

برخی از ویژگی های اصلی پروژه JavaConfig اکنون به Spring Framework اضافه شده است. این بدان معنی است که حاشیه نویسی های زیر اکنون مستقیماً پشتیبانی می شوند:

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

در اینجا نمونه ای از کلاس جاوا است که با استفاده از ویژگی های JavaConfig جدید، پیکربندی اولیه را ارائه می دهد:

**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);

}

}

برای انجام این کار، باید ورودی اسکن مؤلفه زیر را در فایل XML application contextحداقل برنامه خود اضافه کنید.

<context:component-scan base-package="org.example.config"/>

<util:properties id="jdbcProperties" location="classpath:org/example/config/jdbc.properties"/>

یا می توانید یک کلاس Configuration@ را مستقیماً با استفاده از AnnotationConfigApplicationContext بوت استرپ کنید:

**public** **static** **void** main(String[] args) {

ApplicationContext ctx = **new** AnnotationConfigApplicationContext(AppConfig.**class**);

FooService fooService = ctx.getBean(FooService.**class**);

fooService.doStuff();

}

برای اطلاعات کامل در مورد AnnotationConfigApplicationContext به بخش 5.12.2، "نمونه سازی ظرف Spring با استفاده از AnnotationConfigApplicationContext" مراجعه کنید.

#### Defining bean metadata within components

روش‌های حاشیه‌نویسی @Bean نیز در اجزای Spring پشتیبانی می‌شوند. آنها تعریف factory beanرا به ظرف ارائه می دهند. برای اطلاعات بیشتر به تعریف فراداده bean در اجزاء مراجعه کنید

### 2.5.4 General purpose type conversion system and field formatting system

یک سیستم تبدیل نوع عمومی معرفی شده است. این سیستم در حال حاضر توسط SpEL برای تبدیل نوع استفاده می‌شود، و همچنین ممکن است توسط یک Spring Container و DataBinder هنگام اتصال مقادیر ویژگی bean استفاده شود.

علاوه بر این، یک قالب‌ساز SPI برای قالب‌بندی مقادیر فیلد معرفی شده است. این SPI جایگزین ساده‌تر و قوی‌تری برای JavaBean PropertyEditors برای استفاده در محیط‌های کلاینت مانند Spring MVC ارائه می‌کند.

**2.5.5 The Data Tier**

شی به عملکرد نقشه برداری XML (OXM) از پروژه خدمات وب Spring اکنون به چارچوب اصلی Spring منتقل شده است. عملکرد در بسته org.springframework.oxm یافت می شود. اطلاعات بیشتر در مورد استفاده از ماژول OXM را می توان در فصل Marshalling XML با استفاده از O/X Mappers یافت.

**2.5.6 The Web Tier**

هیجان انگیزترین ویژگی جدید برای Web Tier پشتیبانی از ساخت سرویس های وب و برنامه های وب RESTful است. همچنین تعدادی حاشیه نویسی جدید وجود دارد که می تواند در هر برنامه وب استفاده شود.

**Comprehensive REST support**

پشتیبانی از سمت سرور برای ساخت برنامه های RESTful به عنوان یک توسعه از چارچوب وب MVC مبتنی بر حاشیه نویسی موجود ارائه شده است. پشتیبانی از سمت کلاینت توسط کلاس RestTemplate با روح کلاس های قالب دیگر مانند JdbcTemplate و JmsTemplate ارائه می شود. هم عملکرد REST سمت سرور و هم سرویس گیرنده از HttpConverters برای تسهیل تبدیل بین اشیا و نمایش آنها در درخواست ها و پاسخ های HTTP استفاده می کنند.

MarshallingHttpMessageConverter از عملکرد نگاشت Object به XML که قبلا ذکر شد استفاده می کند.

برای اطلاعات بیشتر به بخش های MVC و RestTemplate مراجعه کنید.

**@MVC additions**

فضای نام mvc معرفی شده است که پیکربندی Spring MVC را بسیار ساده می کند.

حاشیه نویسی های اضافی مانند @CookieValue و @RequestHeaders اضافه شده است. برای اطلاعات بیشتر به نگاشت مقادیر کوکی با حاشیه‌نویسی @CookieValue و ویژگی‌های سرصفحه درخواست Mapping با حاشیه‌نویسی @RequestHeader مراجعه کنید.

**2.5.7 Declarative model validation**

چندین پیشرفت اعتبارسنجی، از جمله پشتیبانی JSR 303 که از Hibernate Validator به عنوان ارائه‌دهنده پیش‌فرض استفاده می‌کند.

**2.5.8 Early support for Java EE 6**

ما از طریق استفاده از حاشیه نویسی @Async جدید (یا حاشیه نویسی Asynchronous@ EJB 3.1) از فراخوانی روش ناهمزمان پشتیبانی می کنیم.

JSR 303، JSF 2.0، JPA 2.0 و غیره

**2.5.9 Support for embedded databases**

اکنون پشتیبانی راحت از موتورهای پایگاه داده جاوا جاسازی شده از جمله HSQL، H2 و Derby ارائه شده است.

# 3. New Features and Enhancements in Spring Framework 3.1

این لیستی از ویژگی های جدید Spring Framework 3.1 است. تعدادی از ویژگی‌ها مستندات مرجع اختصاصی ندارند، اما دارای Javadoc کامل هستند. در چنین مواردی، نام کلاس های کاملاً واجد شرایط ذکر می شود. همچنین به پیوست C، مهاجرت به چارچوب بهار 3.1 مراجعه کنید

## 3.1 Cache Abstraction

* [Chapter 29, *Cache Abstraction*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/cache.html)
* [Cache Abstraction](http://blog.springsource.com/2011/02/23/spring-3-1-m1-caching/) (SpringSource team blog)

## 3.2 Bean Definition Profiles

* [XML profiles](http://blog.springsource.com/2011/02/11/spring-framework-3-1-m1-released/) (SpringSource Team Blog)
* [Introducing @Profile](http://blog.springsource.com/2011/02/14/spring-3-1-m1-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](http://blog.springsource.com/2011/02/11/spring-framework-3-1-m1-released/) (SpringSource Team Blog)
* See org.springframework.core.env.Environment Javadoc

## 3.4 PropertySource Abstraction

* [Unified Property Management](http://blog.springsource.com/2011/02/15/spring-3-1-m1-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

برای کلاس‌های موجود در بسته جدید org.springframework.orm.hibernate4 به Javadoc مراجعه کنید.

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

حاشیه‌نویسی @ContextConfiguration اکنون از ارائه کلاس‌های Configuration @ برای پیکربندی Spring TestContext پشتیبانی می‌کند. علاوه بر این، یک حاشیه نویسی @ActiveProfiles برای پشتیبانی از پیکربندی declarative پروفایل های تعریف فعال bean در تست های یکپارچه سازی ApplicationContext معرفی شده است.

* [Spring 3.1 M2: Testing with @Configuration Classes and Profiles](http://blog.springsource.com/2011/06/21/spring-3-1-m2-testing-with-configuration-classes-and-profiles/) (SpringSource Team Blog)
* See [Section 11.3.5, “Spring TestContext Framework”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#testcontext-framework)
* See [the section called “Context configuration with annotated classes”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#testcontext-ctx-management-javaconfig) 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”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-c-namespace)

## 3.9 Support for injection against non-standard JavaBeans setters

قبل از Spring Framework 3.1، برای تزریق در برابر یک متد ویژگی، باید کاملاً با قوانین امضای ویژگی JavaBeans مطابقت می‌کرد، یعنی هر روش «setter» باید باطل باشد. اکنون در Spring XML می توان متدهای تنظیم کننده را که هر نوع شی را برمی گرداند، مشخص کرد. این هنگام در نظر گرفتن طراحی APIها برای زنجیره روش، که در آن متدهای تنظیم کننده ارجاع به "this" را برمی گرداند، مفید است.

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

WebApplicationInitializer جدید بر روی پشتیبانی ServletContainerInitializer Servlet 3.0 ایجاد می کند تا جایگزینی برنامه ای برای web.xml سنتی ارائه دهد.

* See org.springframework.web.WebApplicationInitializer Javadoc
* [Diff from Spring's Greenhouse reference application](https://bit.ly/lrDHja) 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

در JPA استاندارد، واحدهای پایداری از طریق فایل‌های META-INF/persistence.xml در فایل‌های jar خاص تعریف می‌شوند که به نوبه خود برای کلاس‌های Entity@ جستجو می‌شوند. در بسیاری از موارد، persistence.xml بیش از یک نام واحد را شامل نمی شود و برای همه نگرانی های دیگر (مانند منبع داده برای استفاده، و غیره) به تنظیمات پیش فرض و/یا تنظیمات خارجی متکی است. به همین دلیل، Spring Framework 3.1 یک جایگزین ارائه می‌کند: LocalContainerEntityManagerFactoryBean یک ویژگی «packagesToScan» را می‌پذیرد و بسته‌های پایه را برای اسکن برای کلاس‌های Entity@ مشخص می‌کند. این شبیه به ویژگی AnnotationSessionFactoryBean با همین نام برای تنظیمات اصلی Hibernate، و همچنین به ویژگی اسکن کامپوننت Spring برای دانه‌های Spring معمولی است. به طور موثر، این اجازه می دهد تا با صرف هزینه تعیین یک بسته پایه برای اسکن موجودیت، راه اندازی JPA بدون XML را انجام دهید: یک تطابق بسیار خوب برای برنامه های کاربردی Spring که به اسکن مؤلفه برای Spring bean نیز متکی هستند، حتی احتمالاً با استفاده از یک Servlet مبتنی بر کد بوت استرپ شده اند. اولیه ساز 3.0

## 3.13 New HandlerMethod-based Support Classes For Annotated Controller Processing

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

کلاس‌های جدید در پاسخ به بسیاری از درخواست‌ها برای سفارشی‌سازی و باز کردن کلاس‌های پشتیبانی از کنترل‌کننده حاشیه‌نویسی ایجاد شدند. در حالی که قبلاً می‌توانستید یک حل‌کننده آرگومان روش کنترل‌کننده مشروح سفارشی را پیکربندی کنید، با کلاس‌های پشتیبانی جدید می‌توانید پردازش را برای هر آرگومان متد پشتیبانی‌شده یا نوع مقدار بازگشتی سفارشی کنید.

به org.springframework.web.method.support.HandlerMethodArgumentResolver Javadoc مراجعه کنید

به org.springframework.web.method.support.HandlerMethodReturnValueHandler Javadoc مراجعه کنید

دومین تفاوت قابل توجه، معرفی یک انتزاع HandlerMethod برای نشان دادن یک روش @RequestMapping است. این انتزاع در سراسر کلاس های پشتیبانی جدید به عنوان نمونه کنترل کننده استفاده می شود. به عنوان مثال یک HandlerInterceptor می تواند کنترل کننده را از Object به HandlerMethod ارسال کند و به روش کنترل کننده هدف، حاشیه نویسی های آن و غیره دسترسی پیدا کند.

کلاس های جدید به طور پیش فرض توسط فضای نام MVC و پیکربندی مبتنی بر جاوا از طریق @EnableWebMvc فعال می شوند. کلاس های موجود همچنان در دسترس خواهند بود، اما استفاده از کلاس های جدید در آینده توصیه می شود.

See [the section called “New Support Classes for @RequestMapping methods in Spring MVC 3.1”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-requestmapping-31-vs-30) for additional details and a list of features not available with the new support classes

## 3.14 "consumes" and "produces" conditions in @RequestMapping

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”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-requestmapping-consumes) and [the section called “Producible Media Types”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-requestmapping-produces)

## 3.15 Flash Attributes and RedirectAttributes

اکنون می توان ویژگی های فلش را در یک FlashMap ذخیره کرد و در جلسه HTTP ذخیره کرد تا از تغییر مسیر نجات پیدا کند. برای یک نمای کلی از پشتیبانی کلی از ویژگی های فلش در Spring MVC به بخش 17.6، "استفاده از ویژگی های فلش" مراجعه کنید.در کنترل‌کننده‌های حاشیه‌نویسی، یک روش @RequestMapping می‌تواند ویژگی‌های فلش را با اعلام آرگومان متد از نوع RedirectAttributes اضافه کند. این آرگومان متد اکنون می‌تواند برای کنترل دقیق ویژگی‌های مورد استفاده در یک سناریوی تغییر مسیر نیز استفاده شود. برای جزئیات بیشتر به بخش "تعیین تغییر مسیر و ویژگی های فلش" مراجعه کنید.

## 3.16 URI Template Variable Enhancements

متغیرهای قالب URI از درخواست فعلی در مکان‌های بیشتری استفاده می‌شوند:

متغیرهای قالب URI علاوه بر پارامترهای درخواست هنگام اتصال درخواست به آرگومان های متد @ModelAttribute استفاده می شوند.

مقادیر آرگومان روش @PathVariable قبل از رندر در مدل ادغام می‌شوند، مگر در نماهایی که محتوا را به صورت خودکار تولید می‌کنند، مانند سریال‌سازی JSON یا مارشال‌سازی XML.

یک رشته تغییر مسیر می‌تواند حاوی مکان‌نماهایی برای متغیرهای URI باشد (مثلاً "redirect:/blog/{year}/{month}"). هنگام گسترش مکان‌ها، متغیرهای قالب URI از درخواست فعلی به طور خودکار در نظر گرفته می‌شوند.

یک آرگومان متد @ModelAttribute می‌تواند از یک متغیر الگوی URI نمونه‌سازی شود، مشروط بر اینکه یک Converter یا PropertyEditor ثبت شده برای تبدیل از یک رشته به نوع شی هدف وجود داشته باشد.

## 3.17 @Valid On @RequestBody Controller Method Arguments

یک آرگومان متد @RequestBody را می‌توان با @Valid حاشیه‌نویسی کرد تا اعتبارسنجی خودکار مشابه پشتیبانی از آرگومان‌های متد ModelAttribute@ را احضار کند. یک MethodArgumentNotValidException ایجاد شده در DefaultHandlerExceptionResolver مدیریت می شود و منجر به یک کد پاسخ 400 می شود.

## 3.18 @RequestPart Annotation On Controller Method Arguments

This new annotation provides access to the content of a "multipart/form-data" request part. See [Section 17.10.5, “Handling a file upload request from programmatic clients”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-multipart-forms-non-browsers) and [Section 17.10, “Spring's multipart (file upload) support”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-multipart).

## 3.19 UriComponentsBuilder and UriComponents

یک کلاس UriComponents جدید اضافه شده است که یک محفظه غیرقابل تغییر از اجزای URI است که دسترسی به تمام اجزای URI موجود را فراهم می کند. یک کلاس UriComponentsBuilder جدید نیز برای کمک به ایجاد نمونه های UriComponents ارائه شده است. این دو کلاس با هم کنترل دقیقی بر تمام جنبه‌های آماده‌سازی یک URI از جمله ساخت، گسترش از متغیرهای قالب URI و رمزگذاری می‌دهند.

در بیشتر موارد، کلاس‌های جدید را می‌توان به‌عنوان جایگزین انعطاف‌پذیرتری برای UriTemplate موجود استفاده کرد، به‌خصوص که UriTemplate به صورت داخلی به همان کلاس‌ها متکی است.

یک زیر کلاس ServletUriComponentsBuilder متدهای کارخانه ای استاتیک را برای کپی اطلاعات از درخواست Servlet فراهم می کند. بخش 17.7 "ساخت URI" را ببینید.

# 4. New Features and Enhancements in Spring Framework 3.2

این بخش به موارد جدید در Spring Framework 3.2 می پردازد. همچنین به پیوست D، مهاجرت به چارچوب بهار 3.2 مراجعه کنید

## 4.1 Support for Servlet 3 based asynchronous request processing

مدل برنامه نویسی Spring MVC اکنون پشتیبانی صریح Servlet 3 async را ارائه می دهد. روش‌های @RequestMapping می‌توانند یکی از موارد زیر را برگردانند:

* 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 [Section 17.3.4, “Asynchronous Request Processing”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-async).

## 4.2 Spring MVC Test framework

پشتیبانی درجه یک برای آزمایش برنامه های Spring MVC با یک API روان و بدون کانتینر Servlet. تست های سمت سرور شامل استفاده از DispatcherServlet هستند در حالی که تست های REST سمت سرویس گیرنده به RestTemplate متکی هستند. See [Section 11.3.6, “Spring MVC Test Framework”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#spring-mvc-test-framework)..

## 4.3 Content negotiation improvements

اکنون یک ContentNegotiationStrategy برای حل و فصل انواع رسانه درخواستی از یک درخواست ورودی در دسترس است. پیاده سازی های موجود بر اساس پسوند فایل، پارامتر پرس و جو، هدر «پذیرش» یا یک نوع محتوای ثابت است. گزینه‌های معادل قبلاً فقط در ContentNegotiatingViewResolver موجود بود اما اکنون در سراسر آن موجود است.

ContentNegotiationManager کلاس مرکزی برای استفاده در هنگام پیکربندی گزینه های مذاکره محتوا است. For more details see [Section 17.15.4, “Configuring Content Negotiation”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-config-content-negotiation).

معرفی ContentNegotiationManger همچنین تطبیق الگوی پسوند انتخابی را برای درخواست های دریافتی امکان پذیر می کند. For more details, see the Javadoc of [RequestMappingHandlerMapping.setUseRegisteredSuffixPatternMatch](http://static.springsource.org/spring-framework/docs/3.2.0.BUILD-SNAPSHOT/api/org/springframework/web/servlet/mvc/method/annotation/RequestMappingHandlerMapping.html#setUseRegisteredSuffixPatternMatch(boolean)).

## 4.4 @ControllerAdvice annotation

کلاس های حاشیه نویسی شده با @ControllerAdvice می توانند شامل متدهای ExceptionHandler، @InitBinder، و @ModelAttribute باشند و این روش ها بر روی روش های RequestMapping@ در سلسله مراتب کنترلر اعمال می شوند، برخلاف سلسله مراتب کنترلری که در آن اعلام شده اند. @ControllerAdvice یک حاشیه نویسی مؤلفه است که به کلاس های پیاده سازی اجازه می دهد تا از طریق اسکن مسیر کلاس به طور خودکار شناسایی شوند.

## 4.5 Matrix variables

حاشیه نویسی @MatrixVariable جدید برای استخراج متغیرهای ماتریس از URI درخواست پشتیبانی می کند. For more details see [the section called “Matrix Variables”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-matrix-variables).

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

یک پیاده‌سازی کلاس پایه انتزاعی از رابط WebApplicationInitializer برای ساده‌سازی ثبت نام مبتنی بر کد یک DispatcherServlet و فیلترهای نگاشت شده به آن ارائه شده است. کلاس جدید AbstractDispatcherServletInitializer نام دارد و زیر کلاس آن AbstractAnnotationConfigDispatcherServletInitializer می تواند با پیکربندی Spring مبتنی بر جاوا استفاده شود. For more details see [Section 17.14, “Code-based Servlet container initialization”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-container-config).

## 4.7 ResponseEntityExceptionHandler class

یک کلاس پایه مناسب با یک روش ExceptionHandler@ که استثناهای استاندارد Spring MVC را مدیریت می‌کند و ResponseEntity را برمی‌گرداند که اجازه سفارشی‌سازی و نوشتن پاسخ با مبدل‌های پیام HTTP را می‌دهد. این به عنوان جایگزینی برای DefaultHandlerExceptionResolver عمل می کند، که همین کار را می کند اما به جای آن یک ModelAndView برمی گرداند.

See the revised [Section 17.11, “Handling exceptions”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-exceptionhandlers) including information on customizing the default Servlet container error page.

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

RestTemplate اکنون می تواند یک پاسخ HTTP را به یک نوع عمومی بخواند (به عنوان مثال List<Account>). سه متد Exchange() جدید وجود دارد که ParameterizedTypeReference را می پذیرد، کلاس جدیدی که امکان گرفتن و ارسال اطلاعات نوع عمومی را فراهم می کند.

برای پشتیبانی از این ویژگی، HttpMessageConverter توسط GenericHttpMessageConverter با افزودن روشی برای خواندن محتوا با یک نوع پارامتر مشخص شده گسترش می‌یابد. رابط جدید توسط MappingJacksonHttpMessageConverter و همچنین توسط Jaxb2CollectionHttpMessageConverter جدید پیاده سازی شده است که می تواند یک مجموعه عمومی را بخواند که در آن نوع عمومی یک نوع JAXB است که با @XmlRootElement یا @XmlType حاشیه نویسی شده است.

## 4.9 Jackson JSON 2 and related improvements

کتابخانه جکسون JSON 2 اکنون پشتیبانی می شود. به دلیل تغییرات بسته بندی در کتابخانه جکسون، کلاس های جداگانه ای در Spring MVC نیز وجود دارد. اینها MappingJackson2HttpMessageConverter و MappingJackson2JsonView هستند. سایر بهبودهای پیکربندی مرتبط شامل پشتیبانی از چاپ زیبا و همچنین JacksonObjectMapperFactoryBean برای سفارشی سازی راحت یک ObjectMapper در پیکربندی XML است.

## 4.10 Tiles 3

اکنون Tiles 3 علاوه بر Tiles 2.x پشتیبانی می شود. پیکربندی آن باید بسیار شبیه به پیکربندی Tiles 2 باشد، یعنی ترکیبی از TilesConfigurer، TilesViewResolver و TilesView به جز استفاده از tiles3 به جای بسته tiles2.

همچنین توجه داشته باشید که علاوه بر تغییر شماره نسخه، وابستگی tiles ها نیز تغییر کرده است. شما باید یک زیر مجموعه یا همه tiles-request-api, tiles-api, tiles-core, tiles-servlet, tiles-jsp, tiles-el داشته باشید.

## 4.11 @RequestBody improvements

اکنون یک آرگومان @RequestBody یا یک آرگومان @RequestPart می‌تواند توسط یک آرگومان Errors دنبال شود که امکان رسیدگی به خطاهای اعتبارسنجی (در نتیجه یک حاشیه‌نویسی Valid@) را به صورت محلی در روش @RequestMapping می‌کند. @RequestBody اکنون از یک پرچم مورد نیاز نیز پشتیبانی می کند.

## 4.12 HTTP PATCH method

روش درخواست HTTP PATCH اکنون می‌تواند در روش‌های RequestMapping@ و همچنین در RestTemplate در ارتباط با Apache HttpComponents HttpClient نسخه 4.2 یا بالاتر استفاده شود. اتصال JDK HttpURLC از روش PATCH پشتیبانی نمی کند.

## 4.13 Excluded patterns in mapped interceptors

Mapped interceptorsاکنون از الگوهای URL پشتیبانی می‌کنند تا حذف شوند. فضای نام MVC و MVC JavaConfig هر دو این گزینه ها را نشان می دهند.

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

از 3.2، Spring به @Autowired و @Value اجازه می دهد تا به عنوان متا حاشیه نویسی استفاده شوند، به عنوان مثال. برای ساخت حاشیه نویسی تزریق سفارشی در ترکیب با واجد شرایط خاص. به طور مشابه، می توانید حاشیه نویسی های سفارشی تعریف @Bean را برای کلاس های پیکربندی @ بسازید، به عنوان مثال. در ترکیب با واجد شرایط خاص، @Lazy، @Primary، و غیره.

## 4.15 Initial support for JCache 0.5

Spring یک آداپتور CacheManager برای JCache فراهم می کند که در برابر نسخه پیش نمایش JCache 0.5 ساخته می شود. پشتیبانی کامل JCache در سال آینده همراه با Java EE 7 نهایی ارائه می شود.

## 4.16 Support for @DateTimeFormat without Joda Time

حاشیه‌نویسی @DateTimeFormat اکنون بدون نیاز به وابستگی به کتابخانه Joda Time قابل استفاده است. اگر Joda Time وجود نداشته باشد، از JDK SimpleDateFormat برای تجزیه و چاپ الگوهای تاریخ استفاده می شود. هنگامی که Joda Time وجود دارد، همچنان در اولویت SimpleDateFormat استفاده می شود.

## 4.17 Global date & time formatting

اکنون می‌توان قالب‌های جهانی را تعریف کرد که هنگام تجزیه و چاپ انواع تاریخ و زمان استفاده می‌شوند. See [Section 7.7, “Configuring a global date & time format”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/validation.html#format-configuring-formatting-globaldatetimeformat) for details.

## 4.18 New Testing Features

علاوه بر گنجاندن فوق‌الذکر  [Spring MVC Test Framework](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#spring-mvc-test-framework) در ماژول تست فنری، چارچوب Spring TestContext با پشتیبانی از برنامه‌های کاربردی وب تست یکپارچه‌سازی و همچنین پیکربندی زمینه‌های برنامه با مقداردهی اولیه‌ی متن بازبینی شده است. برای جزئیات بیشتر به موارد زیر مراجعه کنید.

* Configuring and [loading a WebApplicationContext](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#testcontext-ctx-management-web) in integration tests
* Configuring [context hierarchies](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#testcontext-ctx-management-ctx-hierarchies) in integration tests
* Testing [request and session scoped beans](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#testcontext-web-scoped-beans)
* Improvements to [Servlet API mocks](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html#mock-objects-servlet)
* Configuring test application contexts with [ApplicationContextInitializers](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html" \l "testcontext-ctx-management-initializers" \o "Context configuration with context initializers)

## 4.19 Concurrency refinements across the framework

Spring Framework 3.2 شامل تنظیم دقیق ساختارهای داده همزمان در بسیاری از بخش‌های چارچوب، به حداقل رساندن قفل‌ها و به طور کلی بهبود ترتیبات برای ایجاد بسیار همزمان از scoped/prototype bean است.

## 4.20 New Gradle-based build and move to GitHub

ساختن و مشارکت در چارچوب با حرکت ما به سیستم ساخت مبتنی بر Gradle و کنترل منبع در GitHub هرگز ساده‌تر نبوده است. برای جزئیات کامل، بخش ساختمان از منبع README و دستورالعمل‌های مشارکت‌کننده را ببینید.

## 4.21 Refined Java SE 7 / OpenJDK 7 support

آخرین اما نه کم‌اهمیت، Spring Framework 3.2 با پشتیبانی اصلاح‌شده جاوا 7 در چارچوب و همچنین از طریق وابستگی‌های شخص ثالث ارتقا یافته ارائه می‌شود: به طور خاص، CGLIB 3.0، ASM 4.0 (که هر دو به عنوان وابستگی‌های داخلی با Spring now عرضه می‌شوند) و پشتیبانی AspectJ 1.7 (در کنار پشتیبانی موجود AspectJ 1.6).

**Part III. Core Technologies**

این بخش از مستندات مرجع، تمام آن فناوری‌هایی را پوشش می‌دهد که کاملاً در چارچوب Spring Framework هستند.

مهمترین آنها کانتینر Inversion of Control (IoC) Spring Framework است. بررسی کامل محفظه IoC Spring Framework با پوشش جامع فناوری‌های برنامه‌نویسی جنبه‌گرا (AOP) Spring دنبال می‌شود. Spring Framework چارچوب AOP خود را دارد که از نظر مفهومی قابل درک است و با موفقیت 80% نیازهای AOP را در برنامه نویسی سازمانی جاوا برطرف می کند.

پوشش ادغام Spring با AspectJ (در حال حاضر غنی ترین - از نظر ویژگی ها - و مطمئناً بالغ ترین پیاده سازی AOP در فضای سازمانی جاوا) نیز ارائه شده است.

در نهایت، اتخاذ رویکرد توسعه مبتنی بر آزمایش (TDD) برای توسعه نرم‌افزار قطعاً توسط تیم Spring حمایت می‌شود، و بنابراین پوشش پشتیبانی Spring برای تست یکپارچه‌سازی پوشش داده می‌شود (در کنار بهترین روش‌ها برای آزمایش واحد). تیم Spring دریافته است که استفاده صحیح از IoC مطمئناً آزمایش واحد و ادغام را آسان‌تر می‌کند (به این دلیل که وجود متدهای تنظیم‌کننده و سازنده‌های مناسب در کلاس‌ها باعث می‌شود تا بدون نیاز به راه‌اندازی رجیستری مکان یاب سرویس، در یک آزمایش به هم متصل شوند. و مانند آن)... فصلی که صرفاً به تست اختصاص داده شده است، امیدواریم شما را نیز در این مورد متقاعد کند.

* [Chapter 5, *The IoC container*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html)
* [Chapter 6, *Resources*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/resources.html)
* [Chapter 7, *Validation, Data Binding, and Type Conversion*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/validation.html)
* [Chapter 8, *Spring Expression Language (SpEL)*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/expressions.html)
* [Chapter 9, *Aspect Oriented Programming with Spring*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/aop.html)
* [Chapter 10, *Spring AOP APIs*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/aop-api.html)
* [Chapter 11, *Testing*](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/testing.html)

# 5. The IoC container

## 5.1 Introduction to the Spring IoC container and beans

این فصل اجرای چارچوب Spring از اصل وارونگی کنترل (IoC) [1] را پوشش می دهد. IoC همچنین به عنوان تزریق وابستگی (DI) شناخته می شود. این فرآیندی است که طی آن اشیاء وابستگی های خود را تعریف می کنند، یعنی اشیایی که با آنها کار می کنند، فقط از طریق آرگومان های سازنده، آرگومان های یک متد کارخانه، یا ویژگی هایی که روی نمونه شی پس از ساخته شدن یا بازگرداندن آن از یک متد کارخانه ای تنظیم می شوند. . سپس container زمانی که لوبیا را ایجاد می کند آن وابستگی ها را تزریق می کند. این فرآیند اساساً معکوس است، از این رو Inversion of Control (IoC) نامیده می شود، که خود bean با استفاده از ساخت مستقیم کلاس ها یا مکانیزمی مانند الگوی Service Locator، نمونه یا مکان وابستگی های خود را کنترل می کند.

بسته های org.springframework.beans و org.springframework.context مبنایی برای کانتینر IoC Spring Framework هستند. رابط BeanFactory مکانیزم پیکربندی پیشرفته ای را ارائه می دهد که قادر به مدیریت هر نوع شی است. ApplicationContext یک رابط فرعی BeanFactory است. ادغام آسان تر با ویژگی های AOP Spring را اضافه می کند. مدیریت منابع پیام (برای استفاده در بین المللی)، انتشار رویداد. و زمینه های خاص لایه برنامه مانند WebApplicationContext برای استفاده در برنامه های کاربردی وب.

به طور خلاصه، BeanFactory چارچوب پیکربندی و عملکردهای اساسی را فراهم می کند، و ApplicationContext عملکردهای ویژه سازمانی بیشتری را اضافه می کند. ApplicationContext یک سوپرمجموعه کامل از BeanFactory است و منحصراً در این فصل در توضیحات کانتینر IoC Spring استفاده می شود. برای اطلاعات بیشتر در مورد استفاده از BeanFactory به جای ApplicationContext، به بخش 5.15، "The BeanFactory" مراجعه کنید.

در Spring، اشیایی که ستون فقرات برنامه شما را تشکیل می دهند و توسط کانتینر Spring IoC مدیریت می شوند، beans نامیده می شوند. لوبیا شیئی است که نمونه سازی شده، مونتاژ می شود، و در غیر این صورت توسط یک ظرف Spring IoC مدیریت می شود. در غیر این صورت، یک لوبیا به سادگی یکی از بسیاری از اشیاء در برنامه شما است. لوبیاها و وابستگی‌های موجود در میان آن‌ها، در فراداده‌های پیکربندی استفاده‌شده توسط یک ظرف منعکس می‌شوند.

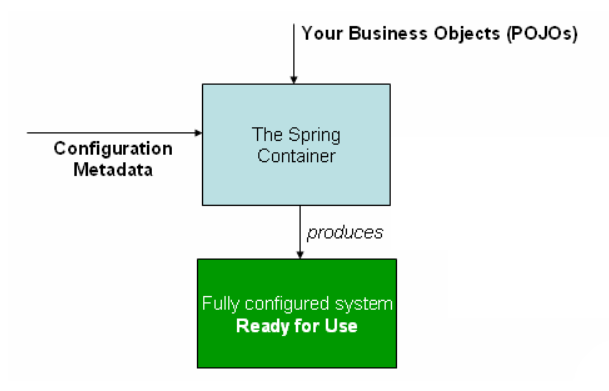
## 5.2 Container overview

رابط org.springframework.context.ApplicationContext نشان دهنده ظرف Spring IoC است و مسئول نمونه سازی، پیکربندی و مونتاژ دانه های فوق الذکر است. ظرف با خواندن فراداده های پیکربندی دستورالعمل های خود را در مورد مواردی که باید نمونه سازی، پیکربندی و مونتاژ شود، دریافت می کند. فراداده پیکربندی در XML، حاشیه نویسی جاوا یا کد جاوا نشان داده می شود. این به شما امکان می دهد اشیایی که برنامه شما را تشکیل می دهند و وابستگی های متقابل غنی بین چنین اشیایی را بیان کنید.

چندین پیاده سازی از رابط ApplicationContext به صورت خارج از جعبه با Spring عرضه می شود. در برنامه های مستقل ایجاد یک نمونه از ClassPathXmlApplicationContext یا FileSystemXmlApplicationContext معمول است. در حالی که XML فرمت سنتی برای تعریف فراداده پیکربندی بوده است، می‌توانید با ارائه مقدار کمی از پیکربندی XML برای فعال کردن پشتیبانی از این فرمت‌های فراداده اضافی، به ظرف دستور دهید که از حاشیه‌نویسی یا کد جاوا به عنوان قالب فراداده استفاده کند.

در اکثر سناریوهای برنامه کاربردی، برای نمونه سازی یک یا چند نمونه از یک ظرف Spring IoC، به کد کاربر صریح نیازی نیست. به عنوان مثال، در یک سناریوی برنامه وب، یک هشت خط ساده (یا بیشتر) از XML توصیف‌گر وب boilerplate J2EE در فایل web.xml برنامه معمولاً کافی است (به بخش 5.14.4، «نمونه سازی مناسب ApplicationContext برای برنامه‌های کاربردی وب مراجعه کنید» ). اگر از محیط توسعه مجهز به SpringSource Tool Suite Eclipse یا Spring Roo استفاده می کنید، این پیکربندی دیگ بخار را می توان به راحتی با چند کلیک ماوس یا فشار دادن کلید ایجاد کرد.

نمودار زیر نمای سطح بالایی از نحوه عملکرد Spring است. کلاس های برنامه شما با ابرداده های پیکربندی ترکیب می شوند تا پس از ایجاد و تنظیم اولیه ApplicationContext، یک سیستم یا برنامه کاملاً پیکربندی شده و قابل اجرا داشته باشید.



**5.2.1 Configuration metadata**

همانطور که نمودار قبل نشان می دهد، محفظه Spring IoC شکلی از فراداده پیکربندی را مصرف می کند. این فراداده پیکربندی نشان می‌دهد که چگونه شما به‌عنوان یک توسعه‌دهنده برنامه به کانتینر Spring می‌گویید که اشیاء را در برنامه شما نمونه‌سازی، پیکربندی و مونتاژ کند.

فراداده پیکربندی به طور سنتی در قالب XML ساده و شهودی ارائه می‌شود، که بیشتر این فصل از آن برای انتقال مفاهیم و ویژگی‌های کلیدی کانتینر Spring IoC استفاده می‌کند.

[توجه داشته باشید]

فراداده مبتنی بر XML تنها فرم مجاز فراداده پیکربندی نیست. خود محفظه Spring IoC کاملاً از قالبی که این فراداده پیکربندی واقعاً در آن نوشته شده است جدا شده است.

برای کسب اطلاعات در مورد استفاده از سایر اشکال فراداده با ظرف Spring، نگاه کنید به:

پیکربندی مبتنی بر حاشیه نویسی: Spring 2.5 پشتیبانی از ابرداده پیکربندی مبتنی بر حاشیه نویسی را معرفی کرد.

پیکربندی مبتنی بر جاوا: با شروع Spring 3.0، بسیاری از ویژگی های ارائه شده توسط پروژه Spring JavaConfig بخشی از چارچوب اصلی Spring Framework شدند. بنابراین شما می توانید با استفاده از جاوا به جای فایل های XML، beans را برای کلاس های برنامه خود تعریف کنید. برای استفاده از این ویژگی‌های جدید، به حاشیه‌نویسی‌های @Configuration، @Bean، @Import و @DependsOn مراجعه کنید.

پیکربندی فنری شامل حداقل یک تعریف و معمولاً بیش از یک تعریف است که ظرف باید آن را مدیریت کند. فراداده پیکربندی مبتنی بر XML این دانه‌ها را به‌عنوان عناصر <bean/> در یک عنصر سطح بالای <beans/> پیکربندی می‌کند.

این تعاریف bean با اشیاء واقعی که برنامه شما را تشکیل می دهند مطابقت دارد. معمولاً اشیاء لایه سرویس، اشیاء دسترسی به داده (DAO)، اشیاء ارائه مانند نمونه‌های Struts Action، اشیاء زیرساخت مانند Hibernate SessionFactories، صف‌های JMS و غیره را تعریف می‌کنید. به طور معمول، یکی اشیاء دامنه ریز را در کانتینر پیکربندی نمی کند، زیرا معمولاً مسئولیت DAO ها و منطق تجاری ایجاد و بارگذاری اشیاء دامنه است. با این حال، می توانید از ادغام Spring با AspectJ برای پیکربندی اشیایی که خارج از کنترل یک ظرف IoC ایجاد شده اند استفاده کنید. استفاده از AspectJ برای وابستگی تزریق اشیاء دامنه با Spring را ببینید.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="..." class="...">

*<!-- collaborators and configuration for this bean go here -->*

</bean>

<bean id="..." class="...">

*<!-- collaborators and configuration for this bean go here -->*

</bean>

*<!-- more bean definitions go here -->*

</beans>

خصیصه id رشته ای است که از آن برای شناسایی تعریف منحصر به فرد bean استفاده می کنید. ویژگی class نوع bean را تعریف می کند و از نام کلاس کاملاً واجد شرایط استفاده می کند. مقدار مشخصه id به اشیاء همکار اشاره دارد. XML برای ارجاع به اشیاء همکار در این مثال نشان داده نشده است. برای اطلاعات بیشتر به Dependencies مراجعه کنید.

**5.2.2 Instantiating a container**

نمونه سازی یک کانتینر Spring IoC ساده است. مسیر یا مسیرهای مکان ارائه شده به سازنده ApplicationContext در واقع رشته های منبعی هستند که به کانتینر اجازه می دهند ابرداده های پیکربندی را از انواع منابع خارجی مانند سیستم فایل محلی، از Java CLASSPATH و غیره بارگیری کند.

ApplicationContext context =

**new** ClassPathXmlApplicationContext(**new** String[] {"services.xml", "daos.xml"});

[توجه داشته باشید]

پس از آشنایی با کانتینر IoC Spring، ممکن است بخواهید بیشتر در مورد انتزاع منابع Spring بدانید، همانطور که در فصل 6، منابع توضیح داده شده است، که مکانیزمی مناسب برای خواندن یک جریان ورودی از مکان های تعریف شده در نحو URI ارائه می دهد. به طور خاص، مسیرهای منبع برای ساختن زمینه برنامه‌ها همانطور که در بخش 6.7، «زمینه‌های برنامه و مسیرهای منبع» توضیح داده شد، استفاده می‌شوند.

مثال زیر فایل پیکربندی اشیاء لایه سرویس (services.xml) را نشان می دهد:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

*<!-- services -->*

<bean id="petStore"

class="org.springframework.samples.jpetstore.services.PetStoreServiceImpl">

<property name="accountDao" ref="accountDao"/>

<property name="itemDao" ref="itemDao"/>

*<!-- additional collaborators and configuration for this bean go here -->*

</bean>

*<!-- more bean definitions for services go here -->*

</beans>

مثال زیر فایل daos.xml اشیاء دسترسی به داده را نشان می دهد:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="accountDao"

class="org.springframework.samples.jpetstore.dao.ibatis.SqlMapAccountDao">

*<!-- additional collaborators and configuration for this bean go here -->*

</bean>

<bean id="itemDao" class="org.springframework.samples.jpetstore.dao.ibatis.SqlMapItemDao">

*<!-- additional collaborators and configuration for this bean go here -->*

</bean>

*<!-- more bean definitions for data access objects go here -->*

</beans>

در مثال قبل، لایه سرویس از کلاس PetStoreServiceImpl تشکیل شده است و دو شیء دسترسی به داده از نوع SqlMapAccountDao و SqlMapItemDao بر اساس چارچوب iBatis Object/Relational نگاشت هستند. عنصر نام ویژگی به نام ویژگی JavaBean و عنصر ref به نام تعریف دیگری از bean اشاره دارد. این پیوند بین عناصر id و ref وابستگی بین اشیاء همکار را بیان می کند. برای جزئیات پیکربندی وابستگی‌های یک شی، به Dependencies مراجعه کنید.

**Composing XML-based configuration metadata**

این می تواند مفید باشد که تعاریف bean چندین فایل XML را در بر گیرند. اغلب هر فایل پیکربندی XML فردی یک لایه یا ماژول منطقی در معماری شما را نشان می دهد.

شما می توانید از سازنده زمینه برنامه برای بارگذاری تعاریف bean از تمام این قطعات XML استفاده کنید. همانطور که در بخش قبل نشان داده شد، این سازنده چندین مکان منبع را می گیرد. از طرف دیگر، از یک یا چند مورد از عنصر <import/> برای بارگیری تعاریف bean از یک فایل یا فایل دیگر استفاده کنید. مثلا:

<beans>

<import resource="services.xml"/>

<import resource="resources/messageSource.xml"/>

<import resource="/resources/themeSource.xml"/>

<bean id="bean1" class="..."/>

<bean id="bean2" class="..."/>

</beans>

در مثال قبلی، تعاریف خارجی bean از سه فایل services.xml، messageSource.xml و themeSource.xml بارگیری می‌شوند. همه مسیرهای مکان نسبت به فایل تعریفی هستند که وارد کردن را انجام می دهد، بنابراین services.xml باید در همان دایرکتوری یا مکان کلاس مسیر با فایلی که وارد کردن را انجام می دهد، قرار داشته باشد، در حالی که messageSource.xml و themeSource.xml باید در یک مکان منابع زیر مکان قرار داشته باشند. از فایل وارد کننده همانطور که می بینید، یک اسلش پیشرو نادیده گرفته می شود، اما با توجه به نسبی بودن این مسیرها، بهتر است که به هیچ وجه از اسلش استفاده نکنید. محتویات فایل هایی که وارد می شوند، از جمله عنصر سطح بالای <beans/>، باید طبق طرحواره Spring یا DTD دارای تعاریف معتبر XML bean باشند.

ارجاع فایل ها در فهرست های والد با استفاده از مسیر نسبی "../" امکان پذیر است، اما توصیه نمی شود. انجام این کار باعث ایجاد وابستگی به فایلی می شود که خارج از برنامه فعلی است. به طور خاص، این مرجع برای URL های "classpath:" توصیه نمی شود (به عنوان مثال، "classpath:../services.xml")، که در آن فرآیند حل زمان اجرا "نزدیک ترین" ریشه classpath را انتخاب می کند و سپس به فهرست اصلی آن نگاه می کند. تغییرات پیکربندی Classpath ممکن است منجر به انتخاب دایرکتوری متفاوت و نادرستی شود.

همیشه می‌توانید از مکان‌های منابع کاملاً واجد شرایط به جای مسیرهای نسبی استفاده کنید: به عنوان مثال، "file:C:/config/services.xml" یا "classpath:/config/services.xml". با این حال، توجه داشته باشید که پیکربندی برنامه خود را به مکان‌های مطلق خاصی متصل می‌کنید. به طور کلی ترجیح داده می‌شود که برای چنین مکان‌های مطلق، یک مسیر غیرمستقیم حفظ شود، به‌عنوان مثال، از طریق متغیرهای "${...}" که در زمان اجرا بر اساس ویژگی‌های سیستم JVM حل می‌شوند.

### 5.2.3 Using the container

ApplicationContext رابطی برای یک کارخانه پیشرفته است که قادر به حفظ رجیستری از دانه های مختلف و وابستگی های آنها است. با استفاده از روش T getBean(String name, Class<T> requiredType)می توانید نمونه هایی از bean های خود را بازیابی کنید.

ApplicationContext شما را قادر می سازد تا تعاریف bean را بخوانید و به صورت زیر به آنها دسترسی داشته باشید:

*// create and configure beans*

ApplicationContext context =

**new** ClassPathXmlApplicationContext(**new** String[] {"services.xml", "daos.xml"});

*// retrieve configured instance*

PetStoreServiceImpl service = context.getBean("petStore", PetStoreServiceImpl.**class**);

*// use configured instance*

List userList = service.getUsernameList();

شما از getBean() برای بازیابی نمونه هایی از bean خود استفاده می کنید. رابط ApplicationContext چند روش دیگر برای بازیابی لوبیا دارد، اما در حالت ایده آل کد برنامه شما هرگز نباید از آنها استفاده کند. در واقع، کد برنامه شما نباید به هیچ وجه به متد ()getBean فراخوانی کند، و بنابراین اصلاً وابستگی به API های Spring نداشته باشد. برای مثال، ادغام Spring با چارچوب‌های وب، تزریق وابستگی را برای کلاس‌های چارچوب وب مختلف مانند کنترل‌کننده‌ها و دانه‌های مدیریت‌شده با JSF فراهم می‌کند.

## 5.3 Bean overview

ظرف Spring IoC یک یا چند لوبیا را مدیریت می کند. این دانه‌ها با metadata پیکربندی که به کانتینر ارائه می‌دهید، ایجاد می‌شوند، به عنوان مثال، در قالب تعاریف XML <bean/>.

در خود container، این تعاریف bean به عنوان اشیاء BeanDefinition نشان داده می‌شوند که شامل (در میان اطلاعات دیگر) فوق داده‌های زیر است:

نام کلاس واجد شرایط بسته: معمولاً کلاس پیاده سازی واقعی bean تعریف می شود.

عناصر پیکربندی رفتاری Bean، که بیان می‌کند که bean چگونه باید در کانتینر رفتار کند (محدوده، تماس‌های چرخه حیات، و غیره).

ارجاع به دانه های دیگری که برای انجام کار لوبیا مورد نیاز است. به این مراجع همکار یا وابستگی نیز می گویند.

سایر تنظیمات پیکربندی برای تنظیم در شی جدید ایجاد شده، به عنوان مثال، تعداد اتصالات برای استفاده در یک Bean که یک مخزن اتصال را مدیریت می کند، یا محدودیت اندازه استخر.

این metadata به مجموعه ای از ویژگی ها ترجمه می شود که هر تعریف bean را تشکیل می دهد.

**Table 5.1. The bean definition**

| **Property** | **Explained in...** |
| --- | --- |
| class | [Section 5.3.2, “Instantiating beans”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-class) |
| name | [Section 5.3.1, “Naming beans”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-beanname) |
| scope | [Section 5.5, “Bean scopes”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-scopes) |
| constructor arguments | [Section 5.4.1, “Dependency injection”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-collaborators) |
| properties | [Section 5.4.1, “Dependency injection”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-collaborators) |
| autowiring mode | [Section 5.4.5, “Autowiring collaborators”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-autowire) |
| lazy-initialization mode | [Section 5.4.4, “Lazy-initialized beans”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-lazy-init) |
| initialization method | [the section called “Initialization callbacks”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-lifecycle-initializingbean) |
| destruction method | [the section called “Destruction callbacks”](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans-factory-lifecycle-disposablebean) |

علاوه بر تعاریف bean که حاوی اطلاعاتی در مورد نحوه ایجاد یک bean خاص است، پیاده سازی های ApplicationContext همچنین اجازه ثبت اشیاء موجود را که خارج از کانتینر توسط کاربران ایجاد می شوند را می دهد. این کار با دسترسی به BeanFactory ApplicationContext از طریق متد getBeanFactory() انجام می شود که اجرای BeanFactory DefaultListableBeanFactory را برمی گرداند. DefaultListableBeanFactory این ثبت را از طریق متدهای registerSingleton(..) و registerBeanDefinition(..) پشتیبانی می کند. با این حال، برنامه‌های کاربردی معمولی تنها با لوبیاهایی که از طریق تعاریف متادیتا bean تعریف شده‌اند، کار می‌کنند.

### 5.3.1 Naming beans

هر bean یک یا چند شناسه دارد. این شناسه ها باید در ظرفی که bean را میزبانی می کند منحصر به فرد باشد. یک bean معمولاً فقط یک شناسه دارد، اما اگر به بیش از یک شناسه نیاز داشته باشد، موارد اضافی را می توان نام مستعار در نظر گرفت.

در metaData پیکربندی مبتنی بر XML، از ویژگی‌های id و/یا نام برای تعیین شناسه(های) bean استفاده می‌کنید. ویژگی id به شما امکان می دهد دقیقا یک شناسه را مشخص کنید. معمولاً این نام‌ها حروف عددی هستند ("myBean"، "fooService"، و غیره)، اما ممکن است کاراکترهای خاصی نیز داشته باشند. اگر می‌خواهید نام‌های مستعار دیگری را به bean معرفی کنید، می‌توانید آن‌ها را در ویژگی name که با کاما (،)، نقطه ویرگول (;) یا فاصله سفید از هم جدا شده‌اند، مشخص کنید. به عنوان یک یادداشت تاریخی، در نسخه های قبل از Spring 3.1، ویژگی id به صورت xsd:ID تایپ می شد که کاراکترهای ممکن را محدود می کرد. از 3.1، اکنون xsd:string است. توجه داشته باشید که منحصر به فرد بودن شناسه bean هنوز توسط کانتینر اعمال می شود، البته دیگر توسط تجزیه کننده های XML اعمال نمی شود.

شما نیازی به ارائه نام یا شناسه برای bean ندارید. اگر هیچ نام یا شناسه ای به صراحت ارائه نشده باشد، ظرف یک نام منحصر به فرد برای آن bean ایجاد می کند. با این حال، اگر می‌خواهید به آن bean با نام اشاره کنید، از طریق استفاده از عنصر ref یا جستجوی سبک Service Locator، باید یک نام ارائه کنید. انگیزه های عدم ارائه نام مربوط به استفاده از ایننر لوبیا و همکاران سیم کشی خودکار است.

Bean naming conventions

قرارداد استفاده از قرارداد استاندارد جاوا برای مثال نام فیلدها هنگام نامگذاری دانه ها است. یعنی نام لوبیا با حروف کوچک شروع می شود و از آن به بعد شتری است. نمونه هایی از این نام ها (بدون نقل قول) 'accountManager'، 'accountService'، 'userDao'، 'loginController' و غیره خواهد بود.

نامگذاری لوبیا به طور مداوم پیکربندی شما را برای خواندن و درک آسان‌تر می‌کند، و اگر از Spring AOP استفاده می‌کنید، هنگام اعمال توصیه‌هایی برای مجموعه‌ای از دانه‌های مرتبط با نام کمک زیادی می‌کند.

**Aliasing a bean outside the bean definition**

در خود تعریف bean، با استفاده از ترکیبی از حداکثر یک نام مشخص شده توسط ویژگی id، و هر تعداد نام دیگر در ویژگی name، می توانید بیش از یک نام برای bean ارائه دهید. این نام‌ها می‌توانند نام مستعار معادل همان bean باشند، و برای برخی موقعیت‌ها مفید هستند، مانند اجازه دادن به هر جزء در یک برنامه کاربردی برای ارجاع به یک وابستگی مشترک با استفاده از نام bean که مختص خود آن جزء است.

با این حال، مشخص کردن همه نام‌های مستعار که در آن bean واقعاً تعریف شده است، همیشه کافی نیست. گاهی اوقات توصیه می شود نام مستعار برای یک لوبیا معرفی شود که در جای دیگری تعریف شده است. معمولاً در سیستم‌های بزرگ که پیکربندی بین هر زیرسیستم تقسیم می‌شود، هر زیرسیستم مجموعه‌ای از تعاریف شی خاص خود را دارد. در فراداده پیکربندی مبتنی بر XML، می‌توانید از عنصر <alias/> برای انجام این کار استفاده کنید.

<alias name="fromName" alias="toName"/>

در این مورد، یک لوبیا در همان ظرف که ازName نامگذاری شده است، ممکن است پس از استفاده از این تعریف مستعار، به عنوان toName نیز شناخته شود.

برای مثال، فراداده پیکربندی برای زیرسیستم A ممکن است به یک DataSource از طریق نام 'subsystemA-dataSource اشاره کند. فراداده پیکربندی برای زیرسیستم B ممکن است به یک DataSource از طریق نام 'subsystemB-dataSource' اشاره کند. هنگام نوشتن برنامه اصلی که از هر دو زیرسیستم استفاده می کند، برنامه اصلی از طریق نام "myApp-dataSource" به DataSource اشاره می کند. برای اینکه هر سه نام را به یک شی ارجاع دهید، تعاریف نام مستعار زیر را به ابرداده پیکربندی MyApp اضافه کنید:

<alias name="subsystemA-dataSource" alias="subsystemB-dataSource"/>

<alias name="subsystemA-dataSource" alias="myApp-dataSource" />

اکنون هر کامپوننت و برنامه اصلی می توانند از طریق نامی به dataSource مراجعه کنند که منحصر به فرد است و تضمین شده است که با هیچ تعریف دیگری تداخل نداشته باشد (در واقع ایجاد فضای نام)، اما آنها به همان bean اشاره می کنند.

### 5.3.2 Instantiating beans

تعریف لوبیا اساساً دستور العملی برای ایجاد یک یا چند شی است. container زمانی که از شما خواسته می‌شود به دستور العمل یک bean با نام نگاه می‌کند و از metadata‌های پیکربندی محصور شده توسط تعریف bean برای ایجاد (یا به دست آوردن) یک شی واقعی استفاده می‌کند.

اگر از metadata پیکربندی مبتنی بر XML استفاده می‌کنید، نوع (یا کلاس) شی را که قرار است در ویژگی کلاس عنصر <bean/> نمونه‌سازی شود، مشخص می‌کنید. این ویژگی کلاس، که در داخل یک ویژگی کلاس در یک نمونه BeanDefinition است، معمولاً اجباری است. (برای موارد استثنا، به بخش «نمونه سازی با استفاده از روش کارخانه نمونه» و بخش 5.7، «ارث بری تعریف لوبیا» مراجعه کنید.) شما از ویژگی Class به یکی از دو روش استفاده می کنید:

به طور معمول، برای مشخص کردن کلاس bean که باید در موردی ساخته شود که ظرف خود مستقیماً با فراخوانی سازنده آن به صورت بازتابی، bean را ایجاد کند، که تا حدودی معادل کد جاوا با استفاده از عملگر جدید است.

برای تعیین کلاس واقعی حاوی متد کارخانه ایستا که برای ایجاد شیء فراخوانی می شود، در حالتی که کانتینر یک متد استاتیک و کارخانه ای را روی یک کلاس برای ایجاد bean فراخوانی می کند. نوع شیء برگشتی از فراخوانی روش کارخانه ایستا ممکن است همان کلاس یا کلاس دیگری باشد.

Inner class names

اگر می خواهید تعریف bean را برای یک کلاس تو در تو ثابت پیکربندی کنید، باید از نام باینری کلاس داخلی استفاده کنید.

به عنوان مثال، اگر کلاسی به نام Foo در بسته com.example داشته باشید، و این کلاس Foo دارای یک کلاس داخلی ثابت به نام Bar باشد، مقدار ویژگی 'class' در تعریف bean خواهد بود...

com.example.Foo$Bar

به استفاده از کاراکتر $ در نام برای جدا کردن نام کلاس داخلی از نام کلاس خارجی توجه کنید.

#### Instantiation with a constructor

وقتی یک bean با رویکرد سازنده ایجاد می‌کنید، همه کلاس‌های عادی توسط Spring قابل استفاده و سازگار هستند. یعنی کلاسی که توسعه می‌یابد نیازی به پیاده‌سازی رابط‌های خاص یا کدگذاری به روش خاصی ندارد. صرفاً مشخص کردن کلاس bean کافی است. با این حال، بسته به نوع IoC که برای آن Bean خاص استفاده می‌کنید، ممکن است به یک سازنده پیش‌فرض (خالی) نیاز داشته باشید.

کانتینر Spring IoC می تواند تقریباً هر کلاسی را که می خواهید مدیریت کند، مدیریت کند. فقط به مدیریت JavaBeans واقعی محدود نمی شود. اکثر کاربران Spring JavaBeans واقعی را تنها با سازنده پیش‌فرض (بدون آرگومان) و تنظیم‌کننده‌ها و دریافت‌کننده‌های مناسب که بر اساس ویژگی‌های موجود در ظرف مدل‌سازی شده‌اند، ترجیح می‌دهند. همچنین می‌توانید کلاس‌های عجیب‌تری به سبک غیر لوبیا در ظرف خود داشته باشید. برای مثال، اگر نیاز به استفاده از یک استخر اتصال قدیمی دارید که مطلقاً با مشخصات JavaBean مطابقت ندارد، Spring می‌تواند آن را نیز مدیریت کند.

با فراداده پیکربندی مبتنی بر XML می توانید کلاس bean خود را به صورت زیر مشخص کنید:

<bean id="exampleBean" class="examples.ExampleBean"/>

<bean name="anotherExample" class="examples.ExampleBeanTwo"/>

برای جزئیات در مورد مکانیسم ارائه آرگومان‌ها به سازنده (در صورت نیاز) و تنظیم ویژگی‌های نمونه شی پس از ساخته شدن شی، به تزریق وابستگی‌ها مراجعه کنید.

#### Instantiation with a static factory method

هنگام تعریف bean که با روش کارخانه ایستا ایجاد می کنید، از ویژگی class برای تعیین کلاس حاوی متد static factory و از ویژگی با نام factory-method برای تعیین نام خود متد کارخانه استفاده می کنید. شما باید بتوانید این متد را فراخوانی کنید (با آرگومان های اختیاری همانطور که در ادامه توضیح داده شد) و یک شی زنده را برگردانید، که متعاقباً طوری رفتار می شود که گویی از طریق سازنده ایجاد شده است. یکی از کاربردهای چنین تعریفی، فراخوانی کارخانه‌های استاتیک در کدهای قدیمی است.

تعریف bean زیر مشخص می کند که bean با فراخوانی یک روش کارخانه ایجاد می شود. تعریف نوع (کلاس) شیء برگشتی را مشخص نمی کند، فقط کلاس حاوی متد کارخانه را مشخص می کند. در این مثال، متد ()createInstance باید یک متد ثابت باشد

<bean id="clientService"

class="examples.ClientService"

factory-method="createInstance"/>

**public** **class** ClientService {

**private** **static** ClientService clientService = **new** ClientService();

**private** ClientService() {}

**public** **static** ClientService createInstance() {

**return** clientService;

}

}

برای جزئیات در مورد مکانیسم ارائه آرگومان‌های (اختیاری) به روش کارخانه و تنظیم ویژگی‌های نمونه شی پس از بازگرداندن شی از کارخانه، به جزئیات وابستگی‌ها و پیکربندی مراجعه کنید.

#### **Instantiation using an instance factory method**

مشابه نمونه سازی از طریق روش کارخانه ایستا، نمونه سازی با روش کارخانه نمونه از یک روش غیر استاتیک از یک دانه موجود از ظرف برای ایجاد یک دانه جدید فراخوانی می کند. برای استفاده از این مکانیسم، صفت کلاس را خالی بگذارید، و در ویژگی factory-bean، نام bean را در ظرف فعلی (یا والد/اجداد) که حاوی متد نمونه‌ای است که قرار است برای ایجاد شیء فراخوانی شود، مشخص کنید. نام خود روش کارخانه را با ویژگی factory-method تنظیم کنید.

*<!-- the factory bean, which contains a method called createInstance() -->*

<bean id="serviceLocator" class="examples.DefaultServiceLocator">

*<!-- inject any dependencies required by this locator bean -->*

</bean>

*<!-- the bean to be created via the factory bean -->*

<bean id="clientService"

factory-bean="serviceLocator"

factory-method="createClientServiceInstance"/>

**public** **class** DefaultServiceLocator {

**private** **static** ClientService clientService = **new** ClientServiceImpl();

**private** DefaultServiceLocator() {}

**public** ClientService createClientServiceInstance() {

**return** clientService;

}

}

همانطور که در اینجا نشان داده شده است، یک کلاس کارخانه می تواند بیش از یک روش کارخانه را نیز در خود جای دهد:

<bean id="serviceLocator" class="examples.DefaultServiceLocator">

*<!-- inject any dependencies required by this locator bean -->*

</bean>

<bean id="clientService"

factory-bean="serviceLocator"

factory-method="createClientServiceInstance"/>

<bean id="accountService"

factory-bean="serviceLocator"

factory-method="createAccountServiceInstance"/>

**public** **class** DefaultServiceLocator {

**private** **static** ClientService clientService = **new** ClientServiceImpl();

**private** **static** AccountService accountService = **new** AccountServiceImpl();

**private** DefaultServiceLocator() {}

**public** ClientService createClientServiceInstance() {

**return** clientService;

}

**public** AccountService createAccountServiceInstance() {

**return** accountService;

}

}

این رویکرد نشان می دهد که خود کارخانه لوبیا را می توان از طریق تزریق وابستگی (DI) مدیریت و پیکربندی کرد. به جزئیات وابستگی ها و پیکربندی ها مراجعه کنید.

در مستندات Spring، Factory Bean به Bean اطلاق می‌شود که در ظرف Spring پیکربندی شده است و از طریق یک نمونه یا روش کارخانه ایستا، اشیاء را ایجاد می‌کند. در مقابل، FactoryBean (به حروف بزرگ توجه کنید) به FactoryBean مخصوص Spring اشاره دارد.

## 5.4 Dependencies

یک برنامه کاربردی معمولی سازمانی از یک شی (یا در اصطلاح بهار) تشکیل نشده است. حتی ساده ترین برنامه دارای چند شی است که با هم کار می کنند تا آنچه را که کاربر نهایی به عنوان یک برنامه منسجم می بیند ارائه دهد. این بخش بعدی توضیح می‌دهد که چگونه از تعریف تعدادی از تعاریف bean که به تنهایی می‌مانند، به یک کاربرد کاملاً تحقق‌یافته که در آن اشیاء برای رسیدن به یک هدف با یکدیگر همکاری می‌کنند، می‌روید.

### 5.4.1 Dependency injection

تزریق وابستگی (DI) فرآیندی است که طی آن اشیاء وابستگی‌های خود، یعنی سایر اشیاء را که با آنها کار می‌کنند، تنها از طریق آرگومان‌های سازنده، آرگومان‌های یک متد کارخانه یا ویژگی‌هایی که بر روی نمونه شی پس از ساخته شدن یا برگرداندن آن تنظیم می‌شوند، تعریف می‌کنند. از روش کارخانه ای سپس ظرف زمانی که لوبیا را ایجاد می کند آن وابستگی ها را تزریق می کند. این فرآیند اساساً معکوس است، از این رو Inversion of Control (IoC) نامیده می شود، که خود bean نمونه یا مکان وابستگی های خود را به تنهایی با استفاده از ساخت مستقیم کلاس ها یا الگوی Service Locator کنترل می کند.

کد با اصل DI تمیزتر است و جداسازی زمانی موثرتر است که اشیا با وابستگی های خود ارائه شوند. شیء وابستگی های خود را جستجو نمی کند و مکان یا کلاس وابستگی ها را نمی داند. به این ترتیب، تست کلاس‌های شما آسان‌تر می‌شود، به‌ویژه زمانی که وابستگی‌ها بر روی رابط‌ها یا کلاس‌های پایه انتزاعی هستند، که اجازه می‌دهد از پیاده‌سازی‌های خرد یا ساختگی در تست‌های واحد استفاده شود.

DI در دو نوع اصلی وجود دارد، تزریق وابستگی مبتنی بر سازنده و تزریق وابستگی مبتنی بر Setter.

#### Constructor-based dependency injection

DI مبتنی بر سازنده با فراخوانی یک سازنده با تعدادی آرگومان که هر کدام یک وابستگی را نشان می‌دهند، انجام می‌شود. فراخوانی یک روش کارخانه ایستا با آرگومان های خاص برای ساخت bean تقریباً معادل است، و این بحث آرگومان ها را برای سازنده و روش کارخانه ایستا به طور مشابه مورد بررسی قرار می دهد. مثال زیر کلاسی را نشان می‌دهد که فقط با تزریق سازنده می‌تواند وابستگی تزریق شود. توجه داشته باشید که هیچ چیز خاصی در مورد این کلاس وجود ندارد، این یک POJO است که هیچ وابستگی به رابط های خاص کانتینر، کلاس های پایه یا حاشیه نویسی ندارد.

**public** **class** SimpleMovieLister {

*// the SimpleMovieLister has a dependency on a MovieFinder*

**private** MovieFinder movieFinder;

*// a constructor so that the Spring container can 'inject' a MovieFinder*

**public** SimpleMovieLister(MovieFinder movieFinder) {

**this**.movieFinder = movieFinder;

}

*// business logic that actually 'uses' the injected MovieFinder is omitted...*

}

##### Constructor argument resolution

تطبیق وضوح آرگومان سازنده با استفاده از نوع آرگومان انجام می شود. اگر هیچ ابهامی بالقوه در آرگومان‌های سازنده تعریف bean وجود نداشته باشد، ترتیبی که آرگومان‌های سازنده در تعریف bean تعریف می‌شوند، ترتیبی است که در آن آرگومان‌ها به سازنده مناسب زمانی که bean در حال نمونه‌سازی است، ارائه می‌شود. کلاس زیر را در نظر بگیرید:

**package** x.y;

**public** **class** Foo {

**public** Foo(Bar bar, Baz baz) {

*// ...*

}

}

هیچ ابهامی بالقوه وجود ندارد، با این فرض که کلاس های Bar و Baz با وراثت به هم مرتبط نیستند. بنابراین پیکربندی زیر به خوبی کار می‌کند، و نیازی نیست که شاخص‌ها و/یا انواع آرگومان سازنده را به طور صریح در عنصر <constructor-arg/> مشخص کنید.

<beans>

<bean id="foo" class="x.y.Foo">

<constructor-arg ref="bar"/>

<constructor-arg ref="baz"/>

</bean>

<bean id="bar" class="x.y.Bar"/>

<bean id="baz" class="x.y.Baz"/>

</beans>

هنگامی که به bean دیگری ارجاع داده می شود، نوع آن مشخص می شود و تطبیق می تواند رخ دهد (همانطور که در مثال قبل رخ داد). وقتی از یک نوع ساده مانند <value>true<value> استفاده می‌شود، Spring نمی‌تواند نوع مقدار را تعیین کند و بنابراین نمی‌تواند بدون کمک با نوع آن مطابقت داشته باشد. کلاس زیر را در نظر بگیرید:

**package** examples;

**public** **class** ExampleBean {

*// No. of years to the calculate the Ultimate Answer*

**private** **int** years;

*// The Answer to Life, the Universe, and Everything*

**private** String ultimateAnswer;

**public** ExampleBean(**int** years, String ultimateAnswer) {

**this**.years = years;

**this**.ultimateAnswer = ultimateAnswer;

}

}

###### Constructor argument type matching

در سناریوی قبل، اگر به صراحت نوع آرگومان سازنده را با استفاده از ویژگی type مشخص کنید، کانتینر می تواند از تطبیق نوع با انواع ساده استفاده کند. مثلا:

<bean id="exampleBean" class="examples.ExampleBean">

<constructor-arg type="int" value="7500000"/>

<constructor-arg type="java.lang.String" value="42"/>

</bean>

###### Constructor argument index

از ویژگی index برای مشخص کردن صریح شاخص آرگومان های سازنده استفاده کنید. مثلا:

<bean id="exampleBean" class="examples.ExampleBean">

<constructor-arg index="0" value="7500000"/>

<constructor-arg index="1" value="42"/>

</bean>

علاوه بر حل ابهام چندین مقدار ساده، مشخص کردن یک شاخص ابهام را در جایی که سازنده دو آرگومان از یک نوع داشته باشد برطرف می کند. توجه داشته باشید که شاخص بر اساس 0 است.

###### Constructor argument name

از Spring 3.0 می‌توانید از نام پارامتر سازنده برای ابهام‌زدایی مقدار نیز استفاده کنید:

<bean id="exampleBean" class="examples.ExampleBean">

<constructor-arg name="years" value="7500000"/>

<constructor-arg name="ultimateanswer" value="42"/>

</bean>

به خاطر داشته باشید که برای انجام این کار، کد شما باید با فعال بودن پرچم اشکال زدایی کامپایل شود تا Spring بتواند نام پارامتر را از سازنده جستجو کند. اگر نمی‌توانید کد خود را با پرچم اشکال‌زدایی کامپایل کنید (یا نمی‌خواهید)، می‌توانید از حاشیه‌نویسی @ConstructorProperties JDK برای نام‌گذاری صریح آرگومان‌های سازنده خود استفاده کنید. سپس کلاس نمونه باید به صورت زیر باشد:

**package** examples;

**public** **class** ExampleBean {

*//* Fields omitted

*@ConstructorProperties({"years", "ultimateAnswer"})*

**public** ExampleBean(**int** years, String ultimateAnswer) {

**this**.years = years;

**this**.ultimateAnswer = ultimateAnswer;

}

}

#### Setter-based dependency injection

DI مبتنی بر تنظیم توسط ظرفی که متدهای تنظیم کننده را روی دانه های شما فراخوانی می کند، پس از فراخوانی یک سازنده بدون آرگومان یا روش کارخانه ایستا بدون آرگومان برای نمونه سازی bean شما انجام می شود.

مثال زیر کلاسی را نشان می‌دهد که فقط با استفاده از تزریق ستتر خالص می‌توان به آن وابستگی تزریق کرد. این کلاس جاوای معمولی است. این یک POJO است که هیچ وابستگی به رابط های خاص کانتینر، کلاس های پایه یا حاشیه نویسی ندارد.

**public** **class** SimpleMovieLister {

*// the SimpleMovieLister has a dependency on the MovieFinder*

**private** MovieFinder movieFinder;

*// a setter method so that the Spring container can 'inject' a MovieFinder*

**public** **void** setMovieFinder(MovieFinder movieFinder) {

**this**.movieFinder = movieFinder;

}

*// business logic that actually 'uses' the injected MovieFinder is omitted...*

}

ApplicationContext از DI مبتنی بر سازنده و تنظیم کننده برای دانه هایی که مدیریت می کند پشتیبانی می کند. همچنین پس از تزریق برخی وابستگی ها از طریق رویکرد سازنده، از DI مبتنی بر تنظیم کننده پشتیبانی می کند. شما وابستگی ها را در قالب BeanDefinition پیکربندی می کنید، که با نمونه های PropertyEditor برای تبدیل ویژگی ها از یک فرمت به فرمت دیگر استفاده می کنید. با این حال، اکثر کاربران Spring مستقیماً با این کلاس‌ها (به صورت برنامه‌ای) کار نمی‌کنند، بلکه با یک فایل تعریف XML کار می‌کنند که سپس به صورت داخلی به نمونه‌هایی از این کلاس‌ها تبدیل می‌شود و برای بارگیری کل نمونه کانتینر Spring IoC استفاده می‌شود.

**Constructor-based or setter-based DI?**

از آنجایی که می‌توانید هر دو، DI مبتنی بر سازنده و تنظیم‌کننده را با هم ترکیب کنید، استفاده از آرگومان‌های سازنده برای وابستگی‌های اجباری و تنظیم‌کننده‌ها برای وابستگی‌های اختیاری یک قانون خوب است. توجه داشته باشید که استفاده از حاشیه نویسی @Required در یک تنظیم کننده می تواند برای وابستگی های مورد نیاز تنظیم کننده ها استفاده شود.

تیم Spring عموماً از تزریق تنظیم کننده حمایت می کند، زیرا تعداد زیادی از آرگومان های سازنده می توانند ناکارآمد باشند، به خصوص زمانی که ویژگی ها اختیاری هستند. متدهای Setter نیز اشیاء آن کلاس را قابل پیکربندی مجدد یا تزریق مجدد در آینده می کنند. مدیریت از طریق JMX MBeans یک مورد استفاده قانع کننده است.

برخی از ناب کاران از تزریق مبتنی بر سازنده حمایت می کنند. تامین تمام وابستگی‌های شی به این معنی است که شی همیشه به کد مشتری (در حال فراخوانی) در یک حالت کاملاً اولیه بازگردانده می‌شود. نقطه ضعف آن این است که جسم کمتر در معرض پیکربندی مجدد و تزریق مجدد قرار می گیرد.

از DI استفاده کنید که بیشترین معنا را برای یک کلاس خاص دارد. گاهی اوقات، هنگام برخورد با کلاس های شخص ثالثی که منبع آنها را ندارید، انتخاب با شما انجام می شود. یک کلاس قدیمی ممکن است هیچ روش تنظیم کننده را نشان ندهد، بنابراین تزریق سازنده تنها DI موجود است.

#### Dependency resolution process

ظرف تفکیک وابستگی لوبیا را به صورت زیر انجام می دهد:

ApplicationContext با فراداده های پیکربندی که همه دانه ها را توصیف می کند ایجاد و مقداردهی اولیه می شود. فراداده پیکربندی را می توان از طریق XML، کد جاوا یا حاشیه نویسی مشخص کرد.

برای هر bean، وابستگی‌های آن در قالب ویژگی‌ها، آرگومان‌های سازنده یا آرگومان‌هایی به روش static-factory بیان می‌شود، اگر از آن به‌جای یک سازنده معمولی استفاده می‌کنید. این وابستگی ها در زمانی که لوبیا واقعاً ایجاد می شود به Bean ارائه می شود.

هر ویژگی یا آرگومان سازنده یک تعریف واقعی از مقداری است که باید تنظیم شود، یا ارجاع به bean دیگری در ظرف است.

هر ویژگی یا آرگومان سازنده که یک مقدار است از قالب مشخص شده خود به نوع واقعی آن ویژگی یا آرگومان سازنده تبدیل می شود. به‌طور پیش‌فرض Spring می‌تواند یک مقدار ارائه‌شده در قالب رشته را به همه انواع داخلی مانند int، long، string، boolean و غیره تبدیل کند.

ظرف Spring پیکربندی هر دانه را در حین ایجاد ظرف تأیید می کند، از جمله تأیید اینکه آیا ویژگی های مرجع bean به دانه های معتبر اشاره دارد یا خیر. با این حال، خود خواص لوبیا تا زمانی که لوبیا واقعاً ایجاد نشود تنظیم نمی شود. لوبیاهایی که به صورت تک‌تنه‌ای هستند و به‌صورت پیش‌فرض تنظیم شده‌اند، هنگام ایجاد ظرف ایجاد می‌شوند. محدوده‌ها در بخش 5.5، "حوزه‌های Bean" تعریف شده‌اند، در غیر این صورت، bean تنها زمانی ایجاد می‌شود که درخواست شود. ایجاد یک bean به طور بالقوه باعث می شود که یک نمودار از bean ایجاد شود، زیرا وابستگی های bean و وابستگی های آن (و غیره) ایجاد و اختصاص داده می شوند.

**Circular dependencies**

اگر عمدتاً از تزریق سازنده استفاده می کنید، ممکن است یک سناریوی وابستگی دایره ای حل نشدنی ایجاد کنید.

به عنوان مثال: کلاس A به یک نمونه از کلاس B از طریق تزریق سازنده نیاز دارد، و کلاس B به نمونه ای از کلاس A از طریق تزریق سازنده نیاز دارد. اگر دانه‌ها را برای کلاس‌های A و B پیکربندی کنید تا به یکدیگر تزریق شوند، ظرف Spring IoC این مرجع دایره‌ای را در زمان اجرا شناسایی می‌کند و BeanCurrentlyInCreationException را پرتاب می‌کند.

یک راه حل ممکن این است که کد منبع برخی از کلاس ها را ویرایش کنید تا توسط تنظیم کننده ها به جای سازنده ها پیکربندی شوند. روش دیگر، از تزریق سازنده خودداری کنید و فقط از تزریق ستر استفاده کنید. به عبارت دیگر، اگرچه توصیه نمی شود، اما می توانید وابستگی های دایره ای را با تزریق ستر پیکربندی کنید.

برخلاف حالت معمولی (بدون وابستگی دایره‌ای)، وابستگی دایره‌ای بین لوبیا A و لوبیا B باعث می‌شود یکی از لوبیاها قبل از اینکه کاملاً مقداردهی شود به دیگری تزریق شود (یک سناریوی کلاسیک مرغ/تخم مرغ).

به طور کلی می توانید به بهار اعتماد کنید تا کار درست را انجام دهد. مشکلات پیکربندی، مانند ارجاع به دانه‌های موجود و وابستگی‌های دایره‌ای را در زمان بارگیری ظرف شناسایی می‌کند. اسپرینگ ویژگی ها را تنظیم می کند و وابستگی ها را تا جایی که ممکن است دیرتر برطرف می کند، زمانی که لوبیا واقعاً ایجاد می شود. این بدان معنی است که یک کانتینر Spring که به درستی بارگیری شده است، می‌تواند بعداً در صورت درخواست یک شی، در صورت وجود مشکل در ایجاد آن شی یا یکی از وابستگی‌های آن، استثنا ایجاد کند. به عنوان مثال، bean یک استثنا را در نتیجه یک ویژگی مفقود یا نامعتبر پرتاب می کند. این مشاهده بالقوه تأخیر در برخی از مسائل پیکربندی به همین دلیل است که پیاده سازی های ApplicationContext به طور پیش فرض از پیش نمونه های singleton bean می کنند. با صرف مقداری زمان و حافظه اولیه برای ایجاد این دانه‌ها قبل از نیاز واقعی، مشکلات پیکربندی را زمانی که ApplicationContext ایجاد می‌شود، کشف می‌کنید، نه بعداً. همچنان می‌توانید این رفتار پیش‌فرض را نادیده بگیرید تا لوبیاهای سینگلتون به جای اینکه از قبل آماده شوند، تنبلی اولیه شوند.

اگر هیچ وابستگی دایره‌ای وجود نداشته باشد، هنگامی که یک یا چند دانه همکار به یک دانه وابسته تزریق می‌شود، هر یک از دانه‌های همکار قبل از تزریق به دانه وابسته کاملاً پیکربندی می‌شود. این بدان معناست که اگر bean A وابستگی به bean B داشته باشد، محفظه Spring IoC قبل از فراخوانی روش تنظیم کننده در Bean A، به طور کامل bean B را پیکربندی می کند. وابستگی ها تنظیم می شوند و روش های چرخه حیات مربوطه (مانند روش init پیکربندی شده یا روش InitializingBean callback) فراخوانی می شوند.

#### Examples of dependency injection