|  |
| --- |
| **Công ty Cổ phần Giải pháp Thiên Hoàng** |
| **Bộ phận RnD** |
| **LOGO** |
| **BÁO CÁO** |
| **Đề tài:** |
| **Tìm hiểu về Spring Framework** |
| **Người thực hiện:** |
| **Nguyễn Hữu Hoàng** |
| **Hà Nội, ngày tháng năm 2020** |
|  |

# Mục lục

[Mục lục 2](#_Toc34084477)

[Danh sách hình minh hoạ 3](#_Toc34084478)

[Danh sách bảng biểu 4](#_Toc34084479)

[I. Tổng quan về Spring Framework 5](#_Toc34084480)

[1. Giới thiệu về Spring Framework 5](#_Toc34084481)

[1.1. Dependency Injection và Inversion of Control 5](#_Toc34084482)

[1.2. Các module trong Framework 7](#_Toc34084483)

[1.2.1. Core Container (Nhóm module lõi) 8](#_Toc34084484)

[1.2.2. AOP 8](#_Toc34084485)

[1.2.3. Messaging 8](#_Toc34084486)

[1.2.4. Data Access/Intergration 8](#_Toc34084487)

[1.2.5. Web 9](#_Toc34084488)

[1.2.6. Test 9](#_Toc34084489)

[1.3. Tình huống sử dụng 10](#_Toc34084490)

[1.3.1. Quản lý phụ thuộc (dependency management) và cách đặt tên 11](#_Toc34084491)

[1.3.1.1. Spring Dependencies và phụ thuộc vào Spring 14](#_Toc34084492)

[1.3.1.2. Maven Dependency Management – Quản lý phụ thuộc bằng Maven 14](#_Toc34084493)

[1.3.1.3. Maven "Bill Of Materials" Dependency 14](#_Toc34084494)

[1.3.2. Ghi log 15](#_Toc34084495)

[II. Part 2 16](#_Toc34084496)

[III. Part 3 16](#_Toc34084497)

[IV. Part 4 16](#_Toc34084498)

[V. Part 5 16](#_Toc34084499)

[VI. Part 6 16](#_Toc34084500)

[VII. Part 7 16](#_Toc34084501)

# Danh sách hình minh hoạ

[Figure I‑1Tổng quan Spring Framework 7](#_Toc34069492)

[Figure I‑2 Một ứng dụng web điển hình sử dụng Spring. 10](#_Toc34069493)

[Figure I‑3 Tích hợp với web framework bên ngoài 11](#_Toc34069494)

# Danh sách bảng biểu

[Table I‑1 Danh sách artifact của Spring 12](#_Toc34069497)

# Tổng quan về Spring Framework

## Giới thiệu về Spring Framework

Spring Framework là một nền tảng Java mã nguồn mở cung cấp cơ sở hạ tầng bao quát nhiều mảng trong quá trình phát triển ứng dụng Java. Spring sẽ điều khiển cơ sở hạ tầng (phần nền móng của ứng dụng) để giúp ta tập trung vào việc phát triển logic chương trình.

Spring giúp ta phát triển ứng dụng bằng các “plain old Java Objects” (POJOs – các đối tượng thuần Java) và áp dụng các dịch vụ cấp doanh nghiệp vào các POJO một cách an toàn. Ta có thể áp dụng khả năng này vào việc phát triển Java SE lẫn EE.

Lập trình viên có thể được lợi từ Spring trong các trường hợp sau:

* Viết hàm giao dịch CSDL mà không cần quan tâm nhiều tới các API Transaction
* Tạo HTTP Endpoint mà không cần động tới Servlet API
* Tạo hàm xử lý tin nhắn local mà không cần động tới JMS API (Java Messaging Service)

### Dependency Injection và Inversion of Control

Một chương trình Java – từ những chương trình nhúng nhỏ gọn đến những chương trình nhiều lớp chạy trên các server – được tạo thành những đối tượng (từ nay sẽ gọi là object) tương tác với nhau. Như vậy, những object này *phụ thuộc (dependent)* vào nhau.

Mặc dù Java cung cấp nhiều tính năng, nó thiếu cách sắp xếp những đơn vị tạo nên chương trình thành một thể mạch lạc, khiến lập trình viên phải tự thiết kế chức năng đó. Mặc dù ta có thể dùng các cách tổ chức (design pattern) như Factory, AbstractFactory, Builder, Decorator và Service Locator để sắp xếp các lớp cũng như tạo ra các đối tượng làm nên chương trình, nhưng các design pattern đó chỉ là những lời khuyên được đặt cho một cái tên nào đó. Như vậy, lập trình viên vẫn phải tự mình triển khai các pattern đó trong chương trình.

Thành phần IoC (Inversion of Control – Đảo ngược điều khiển) của Spring Framework giải quyết vấn đề này bằng cách cung cấp các phương thức sắp xếp các mảng riêng biệt thành một chương trình hoàn thiện và sẵn sàng đưa vào sử dụng. Spring Framework biến những design pattern trên thành các đối tượng mà ta có thể tích hợp vào trong chương trình.

Spring Framework triển khai khái niệm IoC thông qua kỹ thuật Dependency Injection, gắn những class phụ thuộc vào nhau với nhau nhưng vẫn giữ chúng độc lập.

Ví dụ: chương trình của bạn có một controller cần có khả năng cập nhật, thêm sửa xoá (CRUD) dữ liệu. Trong class controller đó có một thành phần là service. Class đó sẽ trông như thế này:

|  |
| --- |
| public class ClassController {  ClassService service;   public ClassController(){  this.service = new ClassService();  } } |

Ta đã tạo ra một mối quan hệ phụ thuộc giữa ClassController và ClassService, khi thực hiện kỹ thuật Dependency Injection ta sẽ viết như sau:

|  |
| --- |
| public class ClassController {  ClassService service;   public ClassController(ClassService service){  this.service = service;  } } |

Bây giờ ClassController sẽ không phải lo việc triển khai ClassService nữa, thay vào đó, đối tượng thuộc class đó sẽ được truyền vào khi khởi tạo đối tượng ClassController. Quá trình này sẽ được điều khiển bởi Spring Framework.

Ta đã loại bỏ hoàn toàn sự điều khiển khỏi ClassController và giữ nó ở một vị trí khác (vd: file cấu hình XML, Annotation trong file Java) và đối tượng phụ thuộc (ClassService) được trích vào ClassController thông qua **hàm tạo.** Như vậy, luồng điều khiển đã bị “đảo ngược” bởi DI bởi vì ta đã hoàn toàn nhượng lại quyền kiểm soát các đối tượng phụ thuộc cho một hệ thống bên ngoài.

Ngoài cách thực hiện DI qua hàm tạo, ta còn có thể dùng setter để chích đối tượng ClassService vào ClassController.

|  |
| --- |
| import javax.annotation.Resource; public class ClassController {  ClassService service;   @Resource  public void setService(ClassService service){  this.service = service;  } } |

Ta có thể dùng 1 trong 2 cách trên, tuy nhiên nên tuân theo quy tắc sau:

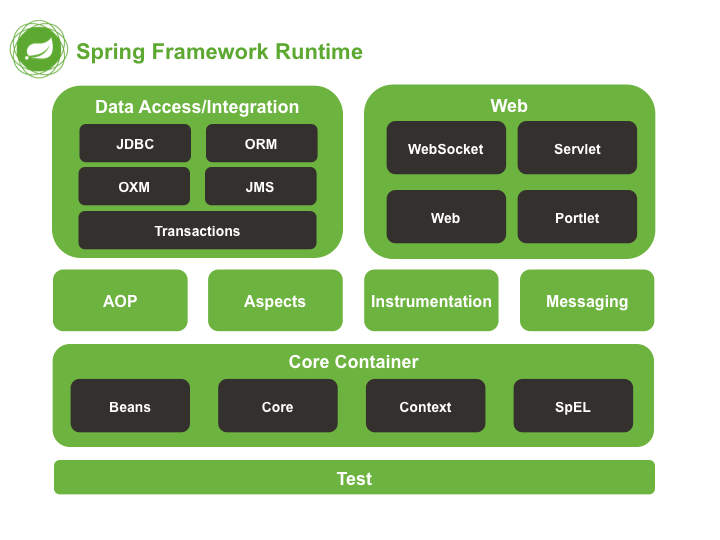
* Dùng **hàm tạo** cho các đối tượng phụ thuộc **bắt buộc**
* Dùng **setter** cho các đối tượng phụ thuộc **không bắt buộc**

Code của ta sẽ gọn gàng hơn khi sử dụng DI và decoupling (hành động tách rời các mảng của một chuwong trình) sẽ hiệu quả hơn.

### Các module trong Framework

Spring Framework được tổ chức theo kiểu các mô đun, mỗi mô đun thực hiện một chức năng khác nhau. Các mô đun này được gộp thành các nhóm: Core Container (Chức năng chính), Data Access/Intergration (Truy cập dữ liệu), Web, AOP (Aspect Oriented Programming – lập trình hướng khía cạnh), Instrumentation, Messaging (liên lạc) và Test (kiểm thử) như hình bên dưới:

Figure I‑1 Tổng quan Spring Framework



Phần sau sẽ liệt kê các module và chức năng của chúng cùng với tên artifact và những khía cạnh chúng bao quát. Tên artifact ứng với mã artifact trong các công cụ quản lý project (vd Maven)

#### Core Container (Nhóm module lõi)

Module lõi bao gồm các module spring-core, spring-beans, spring-context, spring-context-support, và spring-expression (Spring Expression Language – ngôn ngữ biểu diễn Spring).

2 module spring-core, spring-beanscung cấp những phần căn bản của framework, bao gồm những tính năng liên quan đến IoC và Dependency Injection. Class BeanFactoryđược triển khai từ mẫu thiết kế Factory (nhà máy). Nó giúp ta giảm số lượng code viết để cấu hình các đối tượng mà chỉ dùng 1 lần và giúp ta tách riêng logic của ứng dụng khỏi cấu hình và đặc tả của những đối tượng phụ thuộc.

Module context (spring-context) được xây dựng trên nền tảng vững chắc mà module Core và Beans đã tạo ra. Đây là môi trường mà ta có thể tương tác với các đối tượng đã định nghĩa và cấu hình. Interface ApplicationContext là tâm điểm của module này.

Module SpEL (Spring expression language) cung cấp một ngôn ngữ biểu diễn mạnh mẽ dùng để truy vấn và thao túng một đối tượng khi chương trình đang chạy.

#### AOP

Module spring-aopcung cấp giải pháp lập trình hướng khía cạnh cho phép bạn định nghĩa các hàm chặn và những điểm giao để tách các tính năng trong chương trình một cách gọn gàng.

Module spring-aspects cung cấp giải pháp tích hợp với framework AspectJ – một framework AOP nâng cao.

#### Messaging

Kể từ Spring 4, module spring-message được thêm vào Spring Framework. Module này bao gồm các class chính như Message, MessageChannel, MessageHandler và những class khác để cung cấp phần nền cho những ứng dụng dựa trên việc giao tiếp.

#### Data Access/Intergration

Lớp này bao gồm các module: JDBC, ORM, OXM, JMS và Transaction.

Module spring-jdbc cung cấp một lớp JDBC được trừu tượng hoá giúp loại bỏ những dòng code cấu hình lặp đi lặp lại cũng như việc phải xử lý những lỗi riêng của những hệ quản trị CSDL khác nhau.

Module spring-tx hỗ trợ quản lý transaction (giao dịch) bằng code hoặc bằng cấu hình với các class POJO (Plain Old Java Objects)

Module spring-orm cung cấp giải pháp tích hợp với các công cụ ánh xạ đối tượng – quan hệ (ORM – Object-Relational Mapping) như JPA, JDO và Hibernate. Sử dụng module này ta có thể thấy sự tương tác giữa các framework orm này với tất cả các tính năng khác của Spring, ví dụ như cách quản lý transaction bằng cấu hình đã nói ở trên.

Module spring-oxm cung cấp một tầng trừu tượng hoá hỗ trợ ánh xạ đối tượng-XML (Object-XML mapping) và các framework, thư viện liên quan như JAXB, Castor, XMLBeans, v.v

Module spring-jms (Java Messaging Service – dịch vụ truyền tin Java) cung cấp các tính năng sản xuất và xử lý các gói tin. Từ Spring 4.1, module này tích hợp với module spring-messaging.

#### Web

Lớp này bao gồm các module: spring-web, spring-webmvc, spring-websocket, and spring-webmvc-portlet.

Module spring-web cung cấp những tính năng web phổ biến như upload file nhiều part và khởi tạo IoC Container thông qua Servlet listeners và một ApplicationContext hướng web. Nó cũng bao gồm client HTTP và những phần liên quan đến web của tính năng làm việc từ xa của Spring.

Module spring-webmvc (a.k.a module Web-Servlet) chứa giải pháp MVC (Model – View – Controller) và tích hợp dịch vụ REST của Spring

Module spring-mvc-portlet cung cấp giải pháp MVC trong môi trường Portlet, module này có các tính năng tương tự spring-webmvc.

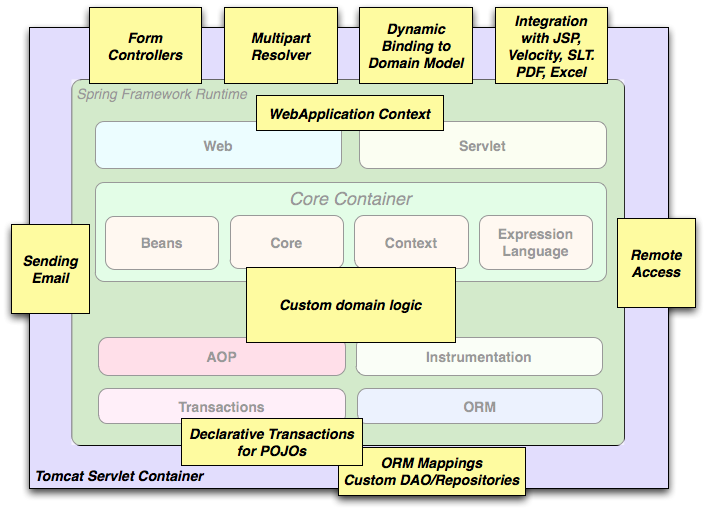
#### Test

Module spring-test hỗ trợ unit testing và intergration testing của các thành phần trong chương trình Spring với Junit hoặc TestNG.

### Tình huống sử dụng

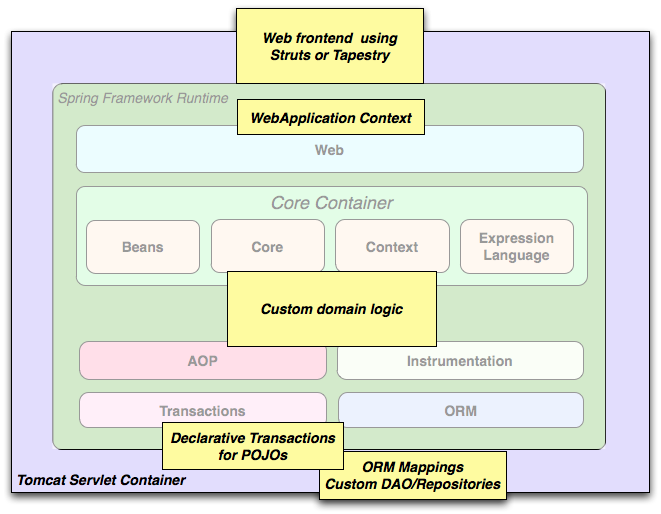
Những module kể trên giúp Spring trở thành giải pháp đúng đắn trong nhiều trường hợp, từ những chương trình nhúng nhỏ gọn đến các giải pháp phần mềm hoàn chỉnh sử dụng tính năng quản lý giao dịch và web framework của Spring.

Figure I‑2 Một ứng dụng web điển hình sử dụng Spring.



Tính năng quản lý giao dịch qua cấu hình (declarative transaction management) của Spring khiến chương trình web của ta hoạt động hoàn toàn qua các giao dịch (tức các lệnh sẽ được gộp vào một nhóm gọi là giao dịch để dễ quản lý). Ta có thể viết logic chương trình bằng các POJO và được quản lý với Spring IoC Container. Spring sẽ tích hợp với các công cụ ORM, ví dụ nếu ta dùng Hibernate thì ta không cần viết lại các file cấu hình ánh xạ và vẫn sẽ dùng SessionFactory như bình thường.

Figure I‑3 Tích hợp với web framework bên ngoài



Đôi khi tình thế không cho phép ta sử dụng một framework khác hoàn toàn. Spring không ép ta phải sử dụng tất cả những module của nó. Front-end đã được dựng bằng Struts, Tapestry hay JSF có thể được tích hợp với Spring để lợi dụng tính năng giao dịch của nó. Bạn chỉ cần viết logic chuwong trình bằng ApplicationContext và dùng WebApplicationContext để tích hợp vào tầng web của bạn.

#### Quản lý phụ thuộc (dependency management) và cách đặt tên

Dependency Management và dependency injection là hai thứ hoàn toàn khác nhau. Để tích hợp tất cả những tính năng hữu dụng của Spring vào chương trình của bạn (ví dụ như DI chẳng hạn), bạn cần tập hợp tất cả các thư viện cần thiết (dưới dạng file .jar) và đảm bảo rằng những file jar đó nằm trong classpath khi chạy chương trình, hay khi compile nữa. Những phụ thuộc này KHÔNG PHẢI những đối tượng được chích vào khi chạy chương trình, mà là những file tách biệt khỏi chương trình và được lưu trữ trong máy tính.

Quá trình quản lý phụ thuộc bao gồm việc định vị những file đó, lưu chúng lại và thêm chúng vào classpath. Các thư viện có thể phụ thuộc một cách trực tiếp hoặc gián tiếp, ví dụ chương trình của ta phụ thuộc trực tiếp vào Spring lúc runtime, hay phụ thuộc vào commons-dbcp (thư viện này lại phụ thuộc vào commons-pool -> chương trình của ta phụ thuộc gián tiếp vào commons-pool). Những thư viện phụ thuộc gián tiếp này được gọi là “transitive” (biến động) và rất khó để định vị và quản lý chúng.

Nếu bạn muốn sử dụng Spring, bạn cần một bản copy của tất cả những thư viện tạo nên Spring mà bạn cần. Để khiến điều này dễ dàng hơn, Spring được gói thành các gói (package) chứa những module liên quan đến nhau. Ví dụ, nếu bạn không lập trình web thì bạn không cần module spring-web. Trong tài liệu này tôi sẽ dùng spring-\* hoặc spring-\*.jar để gọi các thư viện của Spring (vd spring-core, spring-webmvc, spring-jms, vv.). Khi lập trình thì file bạn dùng sẽ có tên = tên module + phiên bản (vd spring-core-4.3.26.RELEASE.jar).

Mỗi bản phát hành của Spring Framework sẽ đăng các artifacts (file .jar) lên Maven Central - kho chứa mặc định mà Maven sẽ tham chiếu đến. Những thư viện khác mà Spring phụ thuộc vào cũng có ở đây, vả lại, phần lớn cộng đồng sử dụng Spring cũng dùng Maven nên rất tiện. Tên các file jar ở đây sẽ theo dạng spring-\*-<phiên bản>.jar và tên group của Spring là org.springframework .

Vậy điều đầu tiên ta cần học là cách quản lý các thư viện phụ thuộc. Tôi khuyên dùng những công cụ tự động như Maven, Gradle hay Ivy, nhưng bạn cũng có thể tự quản lý bằng cách tự tải những file .jar cần thiết và đặt chúng vào đúng chỗ.

Bên dưới là danh sách những artifact của Spring:

Table I‑1 Danh sách artifact của Spring

| **GroupId** | **ArtifactId** | **Description** |
| --- | --- | --- |
| org.springframework | spring-aop | Proxy-based AOP support |
| org.springframework | spring-aspects | AspectJ based aspects |
| org.springframework | spring-beans | Beans support, including Groovy |
| org.springframework | spring-context | Application context runtime, including scheduling and remoting abstractions |
| org.springframework | spring-context-support | Support classes for integrating common third-party libraries into a Spring application context |
| org.springframework | spring-core | Core utilities, used by many other Spring modules |
| org.springframework | spring-expression | Spring Expression Language (SpEL) |
| org.springframework | spring-instrument | Instrumentation agent for JVM bootstrapping |
| org.springframework | spring-instrument-tomcat | Instrumentation agent for Tomcat |
| org.springframework | spring-jdbc | JDBC support package, including DataSource setup and JDBC access support |
| org.springframework | spring-jms | JMS support package, including helper classes to send/receive JMS messages |
| org.springframework | spring-messaging | Support for messaging architectures and protocols |
| org.springframework | spring-orm | Object/Relational Mapping, including JPA and Hibernate support |
| org.springframework | spring-oxm | Object/XML Mapping |
| org.springframework | spring-test | Support for unit testing and integration testing Spring components |
| org.springframework | spring-tx | Transaction infrastructure, including DAO support and JCA integration |
| org.springframework | spring-web | Foundational web support, including web client and web-based remoting |
| org.springframework | spring-webmvc | HTTP-based Model-View-Controller and REST endpoints for Servlet stacks |
| org.springframework | spring-webmvc-portlet | MVC implementation to be used in a Portlet environment |
| org.springframework | spring-websocket | WebSocket and SockJS infrastructure, including STOMP messaging support |

##### Spring Dependencies và phụ thuộc vào Spring

Mặc dù Spring cung cấp tích hợp và hỗ trợ cho rất nhiều công cụ bên thứ 3, các nhà phát triển muốn ta giữ số lượng dependency bắt buộc là tối thiểu. Bạn không cần và không nên tải tất cả các thư viện của Spring để phát triển một ứng dụng đơn giản. Với tính năng dependency injection đơn giản, ta chỉ cần một thư viện bắt buộc cho việc ghi log (sẽ nói thêm ở bên dưới)

Tiếp theo ta sẽ nói về các bước thiết lập một chương trình sử dụng Spring bằng Maven.

##### Maven Dependency Management – Quản lý phụ thuộc bằng Maven

Nếu dùng Maven để quản lý phụ thuộc, ta không cần chỉ định rõ thư viện ghi log. Ví dụ, để tạo ApplicationContext và dùng DI ta chỉ cần liệt kê dependency như sau:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.3.26.RELEASE</version>

<scope>runtime</scope>

</dependency>

</dependencies>

##### Maven "Bill Of Materials" Dependency

Khi sử dụng Maven, rất có khả năng sẽ xảy ra trường hợp ta dùng nhiều phiên bản Spring khác nhau. Để giải quyết việc này, Maven đưa ra giải pháp Bill of Materials (BOM) giúp ta đồng bộ những phiên bản các artifact của một framework, chỉ cần thêm đoạn config sau vào pom.xml:

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-framework-bom</artifactId>

<version>4.3.26.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

Như vậy, ta sẽ không cần phải chỉ định phiên bản của các artifact Spring khác, thẻ dependencies sẽ trông như sau:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

</dependency>

<dependencies>

#### Ghi log

Ghi log (Logging) là một thư viện phụ thuộc rất quan trọng với Spring vì 3 lý do: nó là phụ thuộc bắt buộc duy nhất, ai cũng muốn xem output chương trình (lí do quá rõ ràng, duh), Spring tích hợp nhiều công cụ khác cũng có Logging là phụ thuộc bắt buộc. Mục tiêu của lập trình viên là ghi log vào 1 vị trí cụ thể, tập trung, điều này không dễ vì có rất nhiều framework hỗ trợ việc ghi log.

Thư viện logging bắt buộc của Spring là Jakarta Commons Logging API (JCL) với giải pháp triển khai là commons-logging đước tích hợp trong module spring-core .

Điều tuyệt vời với commons-logging là bạn không cần cài đặt hay cấu hình cái gì khác để ghi log cho chương trình. Khi chạy, nó sẽ tìm các framework logging khác và sử dụng cái nào hợp lý. Nếu không có cái nào thì nó sẽ dùng java.util.logging để output ra console

# Các công nghệ chính

## IoC Container

### Giới thiệu về Spring IoC Container và Beans

Chương này sẽ nói về cách mà Spring triển khai khái niệm IoC (Inversion of Control – Đảo ngược điều khiển). Spring thực hiện một kỹ thuật gọi là Dependency Injection (DI – chích đối tượng phụ thuộc), đây là quá trình mà đối tượng (gọi là A) định nghĩa ra những đối tượng khác (gọi là B) mà nó phụ thuộc vào bằng cách truyền tham số vào hàm tạo (constructor) hoặc bằng các phương thức setter. Sau đó container sẽ chích (inject) B đó vào A khi container tạo ra B. Đây chính là sự đảo ngược mà ta đang nói đến, thay vì chính bản thân A có những dòng code để định nghĩa ra B thì việc này được container làm.

2 package org.springframework.beans và org.springframework.context là tâm điểm của IoC trong Spring. Interface BeanFactory cung cấp những phương thức có thể dùng để xử lý tất cả các loại đối tượng. Interface ApplicationContext mở rộng BeanFactory, có thêm những phương thức để phối hợp với tính năng AOP của Spring, truyền các sự kiện trong chương trình, cùng với các context chuyên dụng như WebApplicationContext để dùng trong các ứng dụng mạng.

Như vậy, BeanFactory cung cấp framework để cấu hình và những tính năng cơ bản, còn ApplicationContext thêm những tính năng cao cấp hơn. Trong những ví dụ, chúng ta sẽ dùng ApplicationContext.

Trong Spring, những đối tượng tạo ra khung làm việc cho chương trình được gọi là bean. Bean là một đối tượng được khởi tạo, sắp xếp, và quản lý bởi IoC Container. Ngoài việc đó thì bean cũng chỉ là một đối tượng bình thường như bao đối tượng bình thường khác (POJO). Beans và các đối tượng phụ thuộc vào chúng được định nghĩa/chỉ định trong thông tin cấu hình được sử dụng bởi Container.

### Tổng quan về IoC Container

Interface org.springframework.context.ApplicationContext đại diện cho IoC Container và có nhiệm vụ khởi tạo, cấu hình và tập hợp các bean đã nói ở trên. Container lấy thông tin cấu hình (configuration metadata) và thực hiện những việc trên theo đúng như chỉ dẫn mà lập trình viên đã viết. Thông tin này được biểu diễn bằng file XML, chú thích trong code (Annotation) hoặc qua code Java. Thông tin này cho phép bạn định nghĩa các đối tượng cần thiết cùng những quan hệ phụ thuộc lẫn nhau giữa chúng.

Spring có sẵn nhiều class triển khai interface ApplicationContext. Với những chương trình standalone (không phụ thuộc, liên quan tới chương trình khác) ta thường dùng class ClassPathXmlApplicationContext or FileSystemXmlApplicationContext. XML là cách cấu hình truyền thống, nhưng bạn có thể sử dụng cách chú thích code hoặc cấu hình qua code. Tuy nhiên bạn vẫn sẽ phải thêm 1 dòng cấu hình XML để sử dụng các cách mới này.

Trong hầu hết các chương trình, lập trình viên không cần viết nhiều code để tạo ra IoC Container. Ví dụ trong một chưong trình web ta chỉ cần ~8 dòng câu hình XML theo mẫu có sẵn trong file web.xml (tham khảo phần XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX). Nếu dùng Spring Tool Suite (phần mềm phát triển ứng dụng Spring), những dòng này bạn còn không phải copy và dán vào, chỉ vài click là xong.

Hình sau mô tả nguyên lý hoạt động của Spring ở mức cao. Những class của bạn sẽ kết hợp với thông tin cấu hình để khi ApplicationContext được tạo ra, bạn sẽ có một chương trình hoàn thiện và sẵn sàng đưa vào sử dụng.

Figure ‑ Spring IoC Container



#### Thông tin cấu hình (Configuration metadata)

Như hình trên đã mô tả, IoC Container cần config data để hoạt động. Thông tin này biểu diễn cách mà lập trình viên chỉ định các hoạt động của chương trình Spring (khởi tạo beans, v.v).

Config metadata thường được biểu diễn dưới dạng XML đơn giản và dễ hiểu, trong phần này ta sẽ dùng chủ yếu dạng này.

Chú thích: cấu hình qua XML không phải cách cấu hình duy nhất, ngày nay nhiều lập trình viên dựa vào Java để viết cấu hình. Chi tiết hơn về các cách cấu hình khác:

* Cấu hình qua chú thích (Annotation): XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
* Cấu hình qua code Java: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX, tham khảo các mục chú thích @Configuration, @Bean, @Import và @DependsOn.

Cấu hình Spring thường sẽ bao gồm 1 hoặc nhiều định nghĩa bean, thông tin để cấu hình các bean này nằm trong phần tử XML <bean/>, phần tử này lại nằm trong phần tử cấp 1 <beans/>. Nếu cấu hình qua Java thì sẽ chú thích một phương thức nào đó với @Bean trong 1 class được chú thích @Configuration.

Những định nghĩa này sẽ giúp tạo ra những đối tượng bạn dùng trong chương trình. Thường những đối tượng này là đối tượng thuộc lớp dịch vụ (service layer objects), đối tượng truy cập dữ liệu (Data Access Objects – DAO), các đối tượng nền móng như SessionFactory của Hibernate, v.v. Thường thì ta không tạo những đối tượng sẽ được lưu trong cơ sở dữ liệu (vd nhân viên 1, nhân viên 2), việc đó là của các DAO, chúng sẽ lấy những đối tượng đó từ CSDL. Nhưng bạn có thể dùng AspectJ để cấu hình những đối tượng ngoài sự quản lý của IoC Container. See [Using AspectJ to dependency-inject domain objects with Spring](https://docs.spring.io/spring/docs/4.3.26.RELEASE/spring-framework-reference/htmlsingle/#aop-atconfigurable). XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Ví dụ sau cho thấy cấu trúc cơ bản của 1 file cấu hình XML:

<?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

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

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

*<!—cấu hình cho bean này ở đây-->*

</bean>

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

*<!—cấu hình cho bean này ở đây-->*

</bean>

*<!—các <bean> khác ở đây-->*

</beans>

Thuộc tính id là 1 String ta có thể dùng để phân biệt giữa các bean có cùng class. Thuộc tính class chỉ định loại đối tượng, dùng tên các lớp Java.

#### Khởi tạo Container

Việc khởi tạo 1 container không hề rắc rối. Ta chỉ cần cung cấp đường dẫn đến file cấu hình XML trong hàm tạo của ApplicationContext là được. Spring có thể đọc cấu hình từ nhiều nguồn khác nhau, vd như file trong máy, Java CLASSPATH, v.v. Bên dưới là cách khởi tạo Container từ file cấu hình XML.

ApplicationContext context =

**new** ClassPathXmlApplicationContext(

"services.xml",

"daos.xml");

Ví dụ service.xml:

<?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

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

*<!-- services -->*

<bean id="subjectService" class="hoang.SubjectServiceImpl">

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

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

*<!—các thuộc tính khác viết sau đây-->*

</bean>

*<!—thêm các service khác ở đây-->*

</beans>

Ví dụ sau đây là về các bean trong dao.xml:

*<?*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 "> *<!-- Enables the Spring MVC @Controller programming model -->* <annotation-driven />  
  
 <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"  
 destroy-method="close">  
 <property name="driverClassName" value="com.mysql.jdbc.Driver" />  
 <property name="url"  
 value="jdbc:mysql://localhost:3306/test" />  
 <property name="username" value="test" />  
 <property name="password" value="test" />  
 </bean>  
  
 *<!-- Hibernate 4 SessionFactory Bean definition -->* <bean id="annotatedSessionFactory"  
 class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">  
 <property name="dataSource" ref="dataSource" />  
 <property name="annotatedClasses">  
 <list>  
 <value>hoang.spring.model.Teacher</value>  
 <value> hoang.spring.model.Assistant</value>  
 </list>  
 </property>  
 <property name="hibernateProperties">  
 <props>  
 <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect  
 </prop>  
 </props>  
 </property>  
 </bean>

<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">  
 <property name="sessionFactory" ref="annotatedSessionFactory" />  
 </bean>  
   
 <bean id="teacherDao" class=" hoang.dao.TeacherDaoImpl">  
 <property name="sessionFactory" ref="annotatedSessionFactory" />  
 </bean>

<bean id="assistantDao" class=" hoang.dao.assistantDAOImpl">  
 <property name="sessionFactory" ref="annotatedSessionFactory"/>  
 </bean>   
</beans>

Trong ví dụ trên, lớp service bao gồm SubjectServiceImpl, 2 đối tượng DAO là teacherDao và assistantDao (dựa trên Hibernate ORM). Phần tử <property name tương ứng với tên thuộc tính của class đó. Phần tử ref (reference) tương ứng với id của 1 bean khác.

##### Viết cấu hình XML

Việc viết cấu hình ở nhiều file XML khá hữu dụng, thường thì mỗi file này sẽ ứng với một tầng trong cấu trúc chương trình.

Bạn có thể truyền vào hàm tạo của ApplicationContext nhiều file XML, nhưng ở đây ta sẽ nói đến cách khác, dùng phần tử <import/> trong file XML như sau:

<beans>

<import resource="services.xml"/>

<import resource="dao.xml"/>

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

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

</beans>

## Resources

## Data Validation, Data Binding, Type Conversion

## Spring Expression Language

## AOP with Spring

## AOP API in Spring

# Part 3

# Part 4

# Part 5

# Part 6

# Part 7