* Spring overview
* Dependency Injection
* Spring Beans
* Spring Annotations
* Spring Data Access
* Aspect Oriented Programming (AOP)
* Spring MVC

Loạt bài này sẽ gồm ba phần, trong phần này, chúng ta sẽ lần lượt đi qua Spring overview, Dependency Injection.

Spring overview

**1.Spring là gì?**

Spring là một open source framework dành cho Java Enterprise. Core feature của Spring có thể dùng để xây dựng bất cứ Java application nào, các extensions của Spring có thể được sử dụng cho việc xây dựng web application trên nền tảng Java EE. Spring framework cũng hướng tới mục tiêu làm cho việc phát triển các ứng dụng trên nền tảng Java EE dễ dàng hơn và thúc đẩy việc lập trình tốt hơn bằng model POJO-based.

**2.Lợi ích mà Spring framework mang lại**

Spring framwork mang lại cho chúng ta sự **gọn nhẹ (lightweight)**, sử dụng **Inversion of control (IoC)** để giúp loose coupling, sử dụng **Aspect oriented programming** để tách biệt các thành phần business, có **container** nới quản lý tất cả life cycle và config của các object trong application, phát triển ứng dụng đơn giản với **MVC**, có **transaction management**, cung cấp các API tiện lợi cho việc **handling exception**.

**3.Các module của Spring framework**

Những module căn bản của Spring gồm có: Core module, Bean module, Context module, Expression Language module, JDBC module, ORM module , OXM module, Java Messaging Service (JMS) module, Transaction module, Web module, Web-Servlet module, Web-Struts module, Web-Portlet module.

**4.Core container**

Đây là module cơ bản của Spring, cái mà sẽ cung cấp nhưng tính năng cơ bản của Spring framework. **BeanFactory** chính là trái tim của bất cứ **Spring-based application** nào, Spring framework được xây dụng trên module này và nó cũng chính là Spring container.

**5.BeanFactory**

BeanFactory là implementation của factory pattern dùng để apply inversion of control để tách biệt config và dependencies. Thông thường BeanFactory sẽ được implement bằng XmlBeanFactory.

**6.XmlBeanFactory**

**org.springframework.beans.factory.xml.XmlBeanFactory** sẽ load file XML chưa định nghĩa các beans. Container này sẽ đọc config từ XML file và sử dụng nó để khởi tạo application.

**7.AOP module**

AOP module sử dụng để phát triển application theo hướng khía cạnh. Phần lớn được cung cấp bởi AOP Alliance để chắc chắn đảm bảo khả năng tương tác giữa Spring và các AOP framwork khác.

**8.JDBC abstraction và DAO module**

Với JDBC abstraction và DAO module, chúng ta có thể chắc chắn về việc giữ cho code thao tác với database luôn đơn giản và sạch sẽ, giảm thiểu thao tác manual để maintain connection hay transaction nhằm cũng tránh được các trường hợp fail do close database resources…, ngoài ra nó còn đưa ra một tầng exeption có nghĩa hơn.

**9.Object relational mapping (ORM) module**

Spring hỗ trợ việc sử dụng ORM thông qua JDBC, ngoài ra cũng support các ORM framework như Hibernate, JPA…

**10.Web module**

Spring web module được xây dựng dựa trên application context module, cung cấp web-based context. Module này hỗ trợ web-oriented như handling multipart request, binding request parameter với business object…

**11.Spring MVC module**

Spring MVC module được sinh ra để hỗ trợ xây dựng MVC web application tốt hơn, mặc dù Spring vẫn có thể integrate được với những MVC framework khác. Spring MVC sử dụng IoC để tách biệt logic ra khỏi controller, vào trong business object. Với Spring MVC ta có thể khai báo để bind request parameter với business object một cách dễ dàng.

**12.Spring configuration file**

Spring configuration file là một file XML. File này sẽ chứa các class information và mô tả làm sao chúng được cài đặt và giao tiếp với những class khác.

**13.Spring IoC container**

Spring IoC chịu trách nhiệm cho việc tạo ra các object, quản lý chúng với các dependency (dependency injection (DI)), wiring chúng lại với nhau, cài đặt chúng, cũng như quản lý lifecycle của chúng.

**14.Lợi ích của IoC**

IoC hay dependency injection tối thiểu hóa lương code trong application. Nó giúp cho application dễ test hơn, bởi vì singleton hay JNDI lookup không thể sử dụng trong Unit test. IoC container cũng hỗ trợ eager instantiation và lazy loading.

**15.Implement Application context phổ biến**

**FileSystemXmlApplicationContext,** chúng ta phải truyền đường dẫn đầy đủ của file XML vào constructor.

**ClassPathXmlApplicationContext,**container sẽ tìm file XML từ trong CLASSPATH.

**WebXmlApplicationContext,**container sẽ tìm file XML từ web application.

**16.Điểm khác nhau giữa BeanFactory và Application context**

Application context cung cấp phương tiện để giải quyết các text message bằng cách implement MessageSource, ngoài ra nó cũng đưa ra cách thức generic để load file resource (ví dụ như file hình ảnh).

**17.Cấu trúc của Spring application**

Interface khai báo tất cả các function. Implementation bao gồm các thuộc tính, getter, setter, function… Spring AOP. Spring configuration XML file. Và chương trình sử dụng tất cả các thứ trên

Dependency Injection

**18.Dependency Injection trong Spring**

Dependency injection là một concept chung, có thể mô tả bằng nhiều cách. Concept này nói rằng bạn sẽ không tạo ra object nhưng bạn sẽ phải mô tả làm sao để chúng được tạo ra. Và bạn cũng không gọi kết nối trực tiếp các component cũng như service với nhau trong code nhưng bạn phải mô tả component nào thì cần service nào trong file config. Sau khi khai báo tất cả thì container cụ thể là IoC container sẽ làm nhiệm vụ hook tất cả lại với nhau. Cụ thể hơn về concept các bạn có thể xem bài [AngularJS cho người mới: Dependencies và Services](https://codeaholicguy.wordpress.com/2016/01/14/series-angularjs-cho-nguoi-moi-dependencies-va-services/) hoặc [Dependency Injection và Inversion of Control](https://toidicodedao.wordpress.com/2015/11/03/dependency-injection-va-inversion-of-control-phan-1-dinh-nghia/).

**19.Những kiểu khác nhau của IoC**

* **Constructor-based dependency injection:** Constructor-based DI được thực hiện khi container invoke một class constructor với một số tham số, mỗi tham số sẽ đại diện cho một dependency đến một class khác.
* **Setter-based dependency injection:** Setter-based DI được thực hiện bởi container bằng cách gọi setter method từ bean sau khi invoke no-argument constructor hay no-argument  static factory để khởi tạo bean.

**20.Nên sử dụng Constructor-based hay setter-based DI**

Có thể sử dụng cả hai. Best practice là sử dụng constructor argument những dependency bắt buộc, và setter cho optional dependency.

Spring Beans

**21. Spring beans là gì?**

Spring Beans chính là những Java Object mà từ đó tạo nên khung sườn của một Spring application.Chúng được cài đặt, lắp ráp và quản lý bởi Spring IoC container. Những bean này được tạo ra bởi configuration metadata được cung cấp từ container, ví dụ, trong tag nằm trong file XML.

Các bean được define trong spring framework là singleton bean. Có một thuộc trính trong bean với tên là “singleton” nếu được gán giá trị là true thì bean đó sẽ trở thành singleton, nếu là false thì bean đó sẽ trở thành prototype bean. Mặc định nếu không được định nghĩa giá trị của nó sẽ là true. Vì thế tất cả các bean trong spring framework mặc định sẽ là singleton bean.

**22. Định nghĩa Spring bean gồm những gì?**

Một Spring bean definition chứa tất cả các configuration metadata cái mà cần cung cấp cho container biết làm sao để tạo ra bean, cũng như chi tiết về lifecycle và những depedencies của nó.

**23. Làm sao để cung cấp configuration metadata cho Spring Container?**

Có ba cách để cũng cấp configuration metadata cho Spring container.

* Thông qua XML configuration file
* Thông qua Annotation-based configuration
* Thông qua Java-based configuration

**24. Làm sao để định nghĩa scope của bean?**

Khi định nghĩa một bean trong Spring, chúng ta còn phải định nghĩa scope của bean. Việc định nghĩa scope có thể thực hiện thông qua việc sử dụng thuộc tính tên là “scope” khi định nghĩa. Lấy ví dụ, khi bean phải tạo mới mỗi lần cần sử dụng, thuộc tính scope sẽ là “prototype”. Mặt khác, khi bean luôn luôn trả về một instance giống nhau khi sử dụng, thuộc tính scope sẽ là “singleton”.

**25. Các scope của bean**

Spring framework đưa ra năm scope của bean như sau:

* **singleton**: cho biết bean đó có một instance duy nhất trong Sping IoC container.
* **prototype**: cho biết bean đó được định nghĩa là có nhiều object instances, mỗi lần muốn sử dụng sẽ tạo mới.
* **request**: cho biết bean được định nghĩa với một HTTP request. Scope này chỉ hợp lệ khi chúng ta sử dụng Web Application Context.
* **session**: cho biết bean được định nghĩa với một HTTP session. Scope này cũng chỉ hợp lệ khi chúng ta sử dụng Web Application Context.
* **global-session**: cho biết bean được định nghĩa với một global HTTP session. Scope này cũng chỉ hợp lệ khi chúng ta sử dụng Web Application Context.

Scope mặc định của mọi Spring Bean là singleton.

**26. Singleton Bean có Thread safe trong Spring Framework?**

Không, trong Spring framework, singleton bean không thread safe.

**27. Bean lifecycle trong Spring framework**

* Spring container tìm các bean definition trong file XML và khởi tạo các bean.
* Spring cài đặt tất cả các thuộc tính được định nghĩa trong bean definition (Dependency Injection).
* Nếu bean implement **BeanNameAware** interface, spring sẽ truyền bean id vào trong hàm **setBeanName()**.
* Nếu bean implement **BeanFactoryAware** interface, spring sẽ truyền beanfactory vào hàm **setBeanFactory()**.
* Nếu có bất cứ bean **BeanPostProcessors** nào được liên kết với bean đang khởi tạo, spring sẽ gọi hàm **postProcesserBeforeInitialization()**và **postProcessAfterInitialization()**.
* Nếu bean implement **InitializingBean**, phương thức **afterPropertySet()** sẽ được gọi. Nếu bean đã được khai báo phương thức khởi tạo, thì phương thức đó sẽ được gọi.
* Nếu bean implement **DisposableBean** interface, phương thức **destroy()** sẽ được gọi.

**28. Phương thức nào là quan trọng nhất trong Spring Bean lifecycle**

Có hai phương thức quan trọng nhất trong Spring bean lifecycle. Đầu tiên đó là setup(), phương thức nay được gọi khi bean được load vào container. Phương thức thứ hai đó là teardown(), phương thức này được gọi khi bean được unload khỏi container.

Tag có hai thuộc tính quan trọng là **init-method** và **destroy-method**với thuộc tính này, chúng ta có thể khai báo tùy biến phương thức khởi tạo và destroy cho mỗi bean. Ngoài ra nếu bạn sử dụng Annotation-based thì Spring framework cũng cung cấp hai annotation là @PostConstruct và @PreDestroy

**29. Inner bean trong Spring**

Khi mà bean chỉ được sử dụng như một thuộc tính của một bean khác thì nó được gọi là inner bean. XML-based configuration metadata cung cấp cho chúng ta sử dụng tag bên trong tag hoặc để khai báo gọi inner bean. Inner bean luôn luôn là anonymous và scope của chúng luôn là prototype.

**30. Làm sao để inject Java Collection trong Spring?**

Spring đưa ra cho chúng ta các loại collection như sau:

* dùng để inject list, có thể có các phần tử trùng nhau.
* dùng để inject set, các phần tử không trùng nhau.
* dùng để inject collection dạng key-value, trong đó key và value là loại tùy ý.
* dùng để inject collection dạng key-value, trong đó key và value đều là String.

**31. Bean wiring là gì?**

Wiring, hoặc là Bean wiring là trường hợp mà các bean được kết hợp lại trong Spring container. Khi wiring bean, Spring container cần biết những bean nào cần và làm thế nào để container sử dụng dependency injection nối tất cả chúng lại với nhau.

**32. Bean auto wiring là gì?**

Spring container có khả năng autowire quan hệ giữa các bean có mối quan hệ hợp tác với nhau. Spring sẽ giải quyết các mối quan hệ hợp tác bằng cách xem xét nội dụng của BeanFactory với các tag và .

**33. Các mode auto wiring**

Autowiring có năm mode được sử dụng để hướng dẫn Spring container làm sao autowiring giải quyết dependency injection.

* **no**: đây là default setting, bean tham chiếu sẽ được reference rõ ràng khi khai báo.
* **byName**: khi autowiring byName, Spring container sẽ cố gắng match giá trị được khai báo trong bean với tên tương tự trong configuration file.
* **byType**: khi autowiring bằng data type, Spring container sẽ cố gắng match chính xác với tên của bean trong configuration file. Nếu có nhiều hơn một, fatal exception sẽ được throw ra.
* **constructor**: mode này tương tự với byType, nhưng type sẽ apply lên các constructor argument. Nếu không có kết quả nào phù hợp, thì fata exception cũng được throw ra.
* **autodetect**: Spring sẽ cố gắng wire bằng cách sử dụng constructor, nếu không có nó sẽ fallback xuống sử dụng byType.

**34. Hạn chế của autowiring**

Các hạn chế của việc autowiring:

* **Overriding**: vẫn có thể định nghĩa các dependencies bằng và việc này sẽ luôn luôn override autowiring.
* **Data types**: Không thể autowire những thuộc tính đơn giản như primitive, String và Classes.
* **Confusing**: autowiring thì không tường minh, vì thế sử dụng khai báo tường minh có thể là cách khôn ngoan hơn.

**35. Có thể inject null hoặc empty String trong Spring hay không?**

Có thể.

Spring Annotations

**36. Java-based configuration**

Java-based configuration cho phép chúng ta viết Spring configuration mà không cần sử dụng XML, thay vào đó chúng ta sử dụng các Java-based annotation. Về Annotation trong Java các bạn có thể tìm đọc bài viết của tôi trong bài [Hướng dẫn sử dụng Annotations](https://codeaholicguy.wordpress.com/2015/06/30/getting-deal-with-custom-java-annotations/).

**37. Annotation-based configuration**

Một phương thức thay thế cho việc setup application bằng XML là chúng ta cung cấp các annotation-based configuration dựa trên bytecode metadata cho việc wiring các component. Thay vì sử dụng XML để mô tả một bean, lập trình viên chuyển các configuration vào trong component class bằng cách sử dụng annotation trong class, method hay field của chính class đó.

**38. Làm sao cài đặt autowiring?**

Mặc định Spring container sẽ không enable annotation wiring. Để sử dụng annotation-based wiring chúng ta phải enable nó trong Spring configuration bằng tag .

**39. @Required annotation**

Annotation này đơn giản chỉ ra rằng thuộc tính nào của bean phải được cài đặt tại thời điểm config, thông qua khai báo tường minh hay thông qua autowiring. Nếu những thuộc tính của bean không thể được cài đặt thì container sẽ throw **BeanInitializationException**.

**40. @Autowired annotation**

**@Autowired** annotation cung cấp phương thức wiring cũng như làm thế nào autowiring được hoàn thành. Nó có thể sử dụng autowire bean trong setter method như **@Required** annotation, trong constructor, trong property hoặc trong method nào đói với một hoặc nhiều argument.

**41. @Qualifier annotation**

Khi có nhiều hơn một bean với cùng một loại và chỉ có một bean trong số đó cần được wire với một property nào đó, **@Qualifier** sẽ được sử dụng với **@Autowired**để giảm thiểu sự nhầm lẫn bằng cách định danh chính xác bean nào được wire.

Trong bài cuối cùng chúng ta sẽ tiếp tục nói về Spring Data Access, Aspect Oriented Programming (AOP), Spring MVC.

Trong phần cuối cùng này chúng ta sẽ cùng nói về Spring Data Access, Aspect Oriented Programming (AOP), Spring MVC.

Spring Data Access

**42. Sử dụng JDBC hiệu quả trong Spring framework**

Khi sử dụng Spring JDBC framework gánh nặng về việc quản lý resource và lỗi sẽ giảm bớt khá nhiều. Vì thế developer chỉ cần viết statement và query để set và get data từ database. JDBC có thể sử dụng một cách hiệu quả bằng việc sử dụng các template class được cung cấp bởi Spring framework, hay còn gọi là JdbcTemplate.

**43. JdbcTemplate**

JdbcTemplate clas cung cấp nhiều method tiện lợi cho việc convert database data thành primitive type hay object, execute prepared và callable statement, và cuối cùng nó cũng cung cấp việc handling database error dễ dàng hơn.

**44. Spring DAO support**

Data Access Object (DAO) mục đích hướng đến tạo nên sự dễ dàng để làm việc với các công nghệ để access database như JDBC, Hibernate, JPA tiện lợi hơn. Nó cho phép chúng ta dễ dàng thay đổi giữa các công nghệ database lưu chữ khác nhau một cách dễ dàng mà không phải lo lắng về việc thay đổi code hay catch exception cho từng công nghệ cụ thể.

**45. Sử dụng Hibernate trong Spring**

Có hai cách để tích hợp Hibernate với Spring,

* IoC với Hibernate Template và Callback
* Extend HibernateDAOSupport và appy AOP Interceptor.

**46. Spring hỗ trợ những ORM nào?**

Hibernate, iBatis, JPA (Java Persistence API), TopLink, JDO (Java Data Objects), OJB.

**47. Tích hợp Hibernate với Spring sử dụng HibernateDAOSupport**

Sử dụng Spring SessionFactory gọi LocalSessionFactory. Tích hợp gồm ba bước.

* Config Hibernate SessionFactory
* Extend DAO implementation từ HibernateDAOSupport
* Wire Transaction support với AOP

**48. Spring hỗ trợ những kiểu quản lý transaction nào?**

Spring hỗ trợ hai kiểu quản lý transaction:

* **Programmatic transaction management:** Có nghĩa là bạn có thể quản lý transaction với việc lập trình. Điều này khiến mọi việc trở nên vô cùng tường minh và linh hoạt, tuy nhiên rất khó để maintain.
* **Declarative transaction management:** Có nghĩa là bạn phải tách biệt phần quản lý transaction với business code. Chỉ sử dụng annotaion hoặc XML config để manage transaction.

**49. Lợi ích của việc quản lý transaction bằng Spring Framework**

* Cung cấp một phương thức lập trình thích hợp cho việc sử dụng các transaction API khác nhau như JDBC, Hibernate, JPA.
* Cung cấp API đơn giản hơn.
* Tiện lợi trong việc khai báo và sử dụng với **Declarative transaction management**.
* Tích hợp dễ dàng với các data access abstraction.

**50. Nên sử dụng kiểu quản lý transaction như thế nào?**

Phần lớn những người sử dụng Spring framework đều chọn sử dụng **Declarative transaction management**bởi vì nó giúp giảm thiểu việc tác động tới application code.

Spring Aspect Oriented Programming (AOP)

**51. AOP**

AOP là kỹ thuật cho phép developer module hóa các hành vị cắt ngang những thành phần khác ví dụ như logging hay transaction management.

**52. Khía cạnh (Aspect)**

Cốt lõi để xây dựng lên AOP chính là aspect, các khía cạnh sẽ bao đóng các hành vi ảnh hưởng trên nhiều class thành các module có thể tái sử dụng. Ví dụ logging module có thể gọi là AOP aspect của việc ghi log. Một application có thể có nhiều khía cạnh khác nhau phụ thuộc vào requirement. Trong Spring AOP, aspect được implement như một class thông thường với annotation **@Aspect**.

**53. Sự khác nhau giữa concern và cross-cutting concern trong Spring AOP**

Các mối quan tâm (**concern**) là những hành vi chúng ta muốn có trong module của ứng dụng. Một mối quan tâm có thể được khai báo như một function chúng ta muốn implement.

Những mối quan tâm xuyên suốt (**cross-cutting concern**) là những mối quan tâm apply trên tổng thể ứng dụng và ảnh hướng trên toàn bộ ứng dụng.

**54. Join point**

Là khái niệm mô tả nơi mà hành động được thực hiện sử dụng AOP aspect.

**55. Advice**

Advice là hành động sẽ xảy ra trước hoặc sau khi method được thực thi. Về bản chất đây sẽ là một đoạn code invoke vào quá trình thực thi bởi Spring AOP framework.

Có năm loại advice như sau.

* **before:** chạy trước khi phương thức được thực thi.
* **after:** Chạy sau khi phương thức được thực thi và không quan trọng kết quả của phương thức đó là gì.
* **after-returning:** Chạy sau khi phương thức được thực thi thành công.
* **after-throwing:** Chạy sau khi phương thức được thực thi nhưng throw exception.
* **around:** Chạy cả trước và sau khi phương thức được thực thi.

**56. Pointcut**

Pointcut bao gồm một hoặc nhiều join point. Có thể xác định pointcut bằng cách sử dụng expression hoặc pattern.

**57. Introduction là gì?**

Introduction cho phép chúng ta thêm mới method hoặc attribute vào trong những class đã tồn tại.

**58. Target object là gì?**

Là object được advise bởi một hoặc nhiều aspect. Nó luôn luôn là một proxy object và refer đến một advised object.

**59. Proxy là gì?**

Là một object được tạo ra sau khi apply advise vào một target object.

**60. Các loại AutoProxying**

* BeanNameAutoProxyCreator
* DefaultAdvisorAutoProxyCreator
* Metadata autoproxying

**61. Weaving là gì?**

Weaving là một tiến trình của việc kết nối các aspect với các object trong application để tạo ra advised object. Việc này có thể hoàn tất trong compile time, load time hoặc runtime.

**62. XML Schema-based aspect implementation**

Với cách implement này, aspect sử dụng các class thông thường và được config bởi XML.

**63. Annotation-based (@AspectJ based) aspect implementation**

Cách implement này hướng đến việc khai báo aspect như một Java class với annotation bên trong.

Spring Model View Controller (MVC)

**64. Spring MVC framework là gì?**

Là một framework hoàn chỉnh cho việc xây dụng MVC web application.

**65. DispatcherServlet**

Spring Web MVC framework được thiết kế xoay quanh DispatcherServlet, cái mà sẽ handle tất cả các HTTP request cũng như reponse.

**66. WebApplicationContext**

WebApplicationContext là một phiên bản mở rộng của ApplicationContext, nó đưa ra những feature cần thiết cho web application. Nó khác với ApplicationContext ở chỗ nó có thể giải quyết các vấn đề về template, theme, và biết được servlet liên kết với cái gì.

**67. Controller trong Spring MVC framework là cái gì?**

Controller cung cấp phương thức truy xuất vào hành vi của application mà bạn phải khai báo thông qua service interface. Controller biến user input thành model để hiển thị lên view.

**68. @Controller annotation**

@Controller được gắn với class cụ thể thể biến class đó thành controller. Spring không bắt buộc bạn phải extend bất cứ controller base class hay reference tới Servlet API nào. Chỉ đơn giản là annotate class với @Controller.

**69. @RequestMapping annotation**

@RequestMapping sử dụng để map URL với một class hay phương thức xử lý cụ thể nào đỏ.