# DI

Phụ thuộc: khi các đối tượng gọi nhau.

Ví dụ:

Ta có class Circle:

|  |
| --- |
| public class Circle{} |

Và class Drawing có gọi đến Circle

|  |
| --- |
| public class Drawing{    Circle circle;    public Drawing(){  circle = new Circle();  }  public draw(){  circle.draw();  }  } |

Như vậy Drawing đang phụ thuộc vào Circle.

(1) khi chạy Drawing thì phải tìm và kéo theo 1 đối tượng Circle.

(2) nếu muốn Drawing làm việc với một đối tượng khác ví dụ Squares thì ta phải sửa code Drawing.

Giái quyết số (2) bằng interface:

|  |
| --- |
| public interface Shape |

class Circle implement của Shape, như vậy nếu cần làm việc với đối tượng khác, ta chỉ cần viết thêm 1 class về đối tượng đó và nó là implement của Shape.

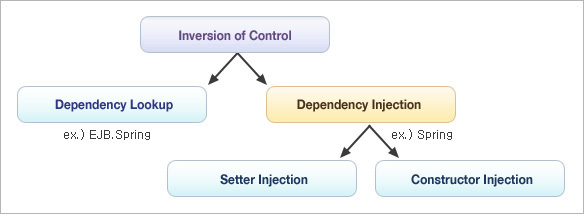
Vấn đề số (1) giải quyết bằng DI: một đối tượng sẽ không phụ thuộc và đối tượng khác và đối tượng khác cũng vậy. Khi cần đối tượng này sẽ gọi tới đối tượng kia và ngược lại.

|  |
| --- |
| //Factory method quy định kiểu dữ liệu của tham chiếu shape là Circle  Shape shape = new Circle();  //Truyền shape qua setter  dr.setShape(shape);  //Từ đây dr như là 1 tham chiếu của Circle, có thể gọi các phương thức trong Circle.  dr.drawDrawing(); |

# IoC

Là khả năng của Spring container trong việc quản lý các thành phần, khi mà thay vì một thành phần phải tự đòi hỏi các tài nguyên cho nó, thì nó sẽ được Spring container cung cấp các tài nguyên dựa vào thông tin trong file cấu hình.

Các container IoC có trách nhiệm khởi tạo, cấu hình và lắp ráp các đối tượng. Các container IoC lấy thông tin từ các tập tin XML và làm việc. Nhiệm vụ chính của container IoC là:



IoC được phân chia thành hai loại khác nhau, đó là:

* Dependency Lookup
* Dependency Injection

***Dependency Lookup*** sẽ tìm kiếm đối tượng phụ thuộc trong khung chứa IoC và sau đó các bạn có thể dùng code để đưa đối tượng phụ thuộc vào trong đối tượng bị phụ thuộc.

***Dependency Injection*** sẽ đưa luôn đối tượng phụ thuộc vào đối tượng bị phụ thuộc.

***Constructor Injection*** việc đưa đối tượng (không bị) phụ thuộc vào đối tượng bị phụ thuộc sẽ thông qua các constructor của đối tượng bị phụ thuộc. Khi đó đối tượng phụ thuộc sẽ là một tham số trong những constructor đó.

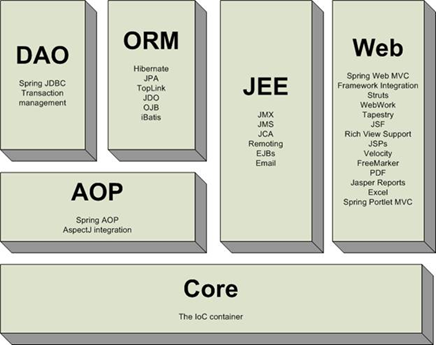
|  |
| --- |
| public class ConstructorInjection {  private Dependency dependency;  public ConstructorInjection(Dependency dependency) {  this.dependency = dependency;  }  } |

Setter Injection thì việc đưa đối tượng phụ thuộc sẽ thông qua một phương thức setter bên trong đối tượng bị phụ thuộc.

|  |
| --- |
| public class SetterInjection {  private Dependency dependency;  public void setDependency(Dependency dependency) {  this.dependency = dependency;  }  } |

## Phân biệt CI và SI

|  |  |  |
| --- | --- | --- |
| **Setter Injection** |  | **Constructor Injection** |
| **1.** Khi sử dụng Setter Injection,nó tiêm một phần phụ thuộc có thể có thể, có nghĩa là nếu chúng ta có 3 thuộc tính như int, string, float, thì sau này muốn sử dụng nó chúng ta không cần thiết phải tiêm (truyền) tất cả các giá trị nếu chúng ta sử dụng setter injection. Nếu bạn không tiêm nó sẽ mất giá trị mặc định cho những đối số nguyên thủy.(ví dụ như int thì mặc định sẽ bằng 0…) |  | **1.** Khi sử dụng constructor injection,tiêm một phần phụ thuộc là không thể, bởi vì để gọi constructor chúng ta phải vượt qua tất cả các đối số, nếu không truyền đầy đủ các đối số thì sẽ nhận được lỗi. |
| **2.** Tiêm Setter sẽ ghi đè giá trị constructor injection, cung cấp nếu chúng ta viết setter injection và nhà xây dựng cho các tài sản cùng [tôi đã nói về việc này, hy vọng bạn nhớ] |  | **2.** Nhưng, constructor injection không thể ghi đè các setter tiêm các giá trị |
| **3.** Sử dụng Injection Setter làm cho đối tượng lớp bean như có thể thay đổi [Chúng ta có thể thay đổi] |  | **3.** Ngược lại,Constructor Injection làm cho đối tượng lớp bean như là bất biến [Chúng ta không thể thay đổi] |
| **4.** Giả sử có 20 bean cần được tiêm phụ thuộc, mỗi bean có 5 thuộc tính nhưng ta cần tiêm 1 thuộc tính trong đó thôi, thì ta chỉ cần viết 20 setter. |  | **4.** Trong khi đó nếu dùng constructor, ta sẽ phải viết đầy đủ 20x5=100 dòng <constructor-arg> cho dù có những thuộc tính để trống |



# Bean

Các đối tượng tạo thành khung của ứng dụng và được quản lý bởi các container Spring IoC được gọi là bean. Một bean là một đối tượng được instantiated - khởi tạo, lắp ráp, và quản lý bởi một container Spring IoC. Các bean này được tạo ra bằng metadata configuration (xml) mà chúng ta cung cấp cho container. Ví dụ, dạng XML <bean /> được dùng để định nghĩa 1 bean.

Các định nghĩa bean chứa trong file cấu hình, cần thiết cho các container xác định:

* Làm thế nào để tạo ra một bean
* Vòng đời của Bean
* Phụ thuộc của Bean.

Các thuộc tính cần quan tâm khi định nghĩa một bean:

* **class**: chỉ rõ class bean được dùng để khởi tạo bean
* **name**: tên đặt cho bean, ta sẽ sử dụng tên này để phân biệt các bean với nhau cũng như gọi ra lúc sử dụng.
* **scope**: phạm vi của bean, thường ta chỉ sử dụng singleton và porotype
* **constructor-arg**: inject dependency qua constructor
* **properties**: inject dependency qua setter
* **autowiring mode**: tự động phát hiện và kết nối class với bean
* **lazy-initialization mode**: ra lệnh cho IoC Container tạo ra một bean instance lúc ra lệnh lần đầu tiên, thường là lúc bắt đầu
* **initialization method**: một call back được gọi lại sau khi các thuộc tính của bean đc tạo ra.
* **destruction method**: tương tự như trên nhưng được dùng lúc một bean bị huỷ.

Có thể cấu hình bean = file cấu hình XML hoặc bằng các đánh dấu Annotation.

## Bean Scope

Có 2 thuộc tính phạm vi bean cần quan tâm là singleton và prototype.

* prototype: để ép Spring tạo ra một bean mới khi cần, và huỷ bean đó khi không dùng nữa.
* singleton: chỉ khởi tạo 1 bean instance và trả về cùng 1 bean đó khi cần thiết, không tạo thêm bean mới. //Đây là cài đặt mặc định. Khi một bean được khởi tạo, nó được lưu vào bộ nhớ cache, các lần gọi tiếp theo sẽ gọi bean này ra.

## Bean Life Cycle

Chu kỳ sống của bean trong Spring rất dễ hiểu. Khi một bean được khởi tạo, nó có thể được yêu cầu để thực hiện một số khởi tạo để có được nó vào một trạng thái sử dụng được. Tương tự như vậy, khi bean không còn cần thiết nữa và được lấy ra khỏi container.

Ta có thể tạo bean = file xml hoặc bằng @annotation (demo)

Các Annotation giúp Spring tự động tạo bean:

– @Component: dùng cho những đối tượng không liên quan đến database, business logic hay presentation layer.

– @Repository: dùng cho những đối tượng liên quan đến database layer.

– @Service: dùng cho những đối tượng liên quan đến business logic layer.

– @Controller: dùng cho những đối tượng liên quan đến presentation layer.

## 

## Bean Autowiring <autowire> và @Autowired

Bản chất của Spring là tạo ra các đối tượng trong container của nó để có thể gọi đến khi cần. Vì thế, Spring hỗ trợ việc tạo ra các bean.

Chúng ta đã biết cách khai báo để đưa đối tượng này vào đối tượng khác, hay có thể nói là đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc, bằng cách sử dụng thuộc tính **ref, idref, list** … Ngoài ra, Spring còn hỗ trợ chúng ta đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc một cách tự động bằng cách sử dụng Bean Autowiring của nó.

Với Bean Autowiring, chúng ta ko cần phải sử dụng các thuộc tính ref, idref hay list, mà chỉ cần viết code tuân thủ theo quy tắc của Spring thì đối tượng phụ thuộc sẽ tự động được đưa vào đối tượng bị phụ thuộc.

Có ba cách để sử dụng Bean Autowiring, đó là: byName, byType, constructor

* byName

mỗi bean trong khung chứa của Spring đều có 1 id của riêng nó. Dựa vào điều này, Spring quy định nếu id của đối tượng phụ thuộc (trong ví dụ của mình là đối tượng Table) giống với tên một biến nào đó trong đối tượng bị phụ thuộc (đối tượng Room) và khi khai báo đối tượng bị phụ thuộc, chúng ta khai báo thêm thuộc tính **autowire=”byName”** thì Spring sẽ tự động đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc qua phương thức setter của đối tượng bị phụ thuộc.

Trong ví dụ của mình, id của đối tượng Table đã hoàn toàn giống với tên biến **table** trong đối tượng Room, do đó chúng ta chỉ cần khai báo thêm cho bean **room** thuộc tính **autowire=”byName”** thì Spring sẽ tự động đưa đối tượng Table vào đối tượng Room cho chúng ta.

|  |
| --- |
| <bean id="room" class="com.huongdanjava.Room" autowire="byType">  </bean> |

* byType

Cách này chúng ta chỉ áp dụng được với những project nhỏ khi mà mỗi đối tượng chỉ có một bean duy nhất trong khung chứa của Spring. Khi đó bằng cách thay đổi thuộc tính autowire=”byType” khi khai báo một bean, Spring sẽ tự động đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc cũng qua phương thức setter của đối tượng bị phụ thuộc.

|  |
| --- |
| <bean id="room" class="com.springexample.Room" autowire="byType">  </bean> |

* constructor

Giống như thuộc tính autowire=”byType”, cách này chúng ta cũng chỉ áp dụng khi đối tượng của chúng ta chỉ có một bean duy nhất trong khung chứa của Spring. Khi đó, Spring sẽ tự động đưa đối tượng phụ thuộc vào đối tượng bị phụ thuộc qua constructor của đối tượng bị phụ thuộc.

|  |
| --- |
| <bean id= "room" class="com.springexample.Room" autowire="constructor">  </bean> |

@**Autowired**

Khi sử dụng annotation này, ta không cần sử dụng các định nghĩa bean phụ thuộc trong file xml.

Để sử dụng **@Autowired** annotation, chúng ta cần đăng ký đối tượng **AutowiredAnnotationBeanPostProcessor** với Spring. Có hai cách để chúng ta có thể làm điều đó:

Sử dụng <context:annotation-config /> trong file cấu hình xml

@Autowired có thể đặt ở Attribute, ở Constructor hoặc ở Setter đều được.

|  |
| --- |
| public class Room {  @Autowired  private Table table;  public Room() {  }  @Autowired  public Room(Table table) {  this.table = table;  }  public Table getTable() {  return table;  }  //@Autowired  public void setTable(Table table) {  this.table = table;  }  } |

## 

## BeanFactory và ApplicationContext

Ta có thể gọi Container của Spring ra bằng 2 cú pháp:

|  |
| --- |
| ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml"); |

hoặc:

|  |
| --- |
| BeanFactory context = new XmlBeanFactory(new ClassPathResource("spring.xml")); |

Interface ApplicationContext được extends từ interface BeanFactory nhưng BeanFactory chỉ cung cấp những tính năng cơ bản cho Dependency Injection trong Spring còn ApplicationContext thì có thêm những chức năng khác nâng cao hơn.

Sự khác biệt đầu tiên của hai interface này là mặc dù cả hai đều cung cấp khả năng lấy đối tượng chúng ta cần trong khung chứa của Spring bằng phương thức getBean() nhưng BeanFactory chỉ tạo ra đối tượng chúng ta cần khi chúng ta gọi phương thức getBean() còn ApplicationContext sẽ tạo hết tất cả các đối tượng chúng ta cần ngay khi bạn gọi đến khung chứa của Spring.

Sự khác biệt thứ hai giữa hai interface này là ở đối tượng hiện thực của chúng. Đối với BeanFactory chúng ta thường dùng đối tượng hiện thực nó là XmlBeanFactory còn ApplicationContext là ClassPathXmlApplicationContext.

Sự khác biệt thứ ba giữa hai interface này là ở khả năng hỗ trợ đa ngôn ngữ. Ví dụ như bạn có một câu được dịch ra bằng ba ngôn ngữ khác nhau: một Tiếng Việt, một Tiếng Anh, một là tiếng Pháp. ApplicationContext cho chúng ta khả năng lấy được nội dung của câu này ở ba ngôn ngữ khác nhau bằng phương thức getMessage(String code, Object[] args, Locale local) trong khi đó BeanFactory không hỗ trợ cho chúng ta làm điều này.

Tóm lại, chúng ta nên sử dụng ApplicationContext.