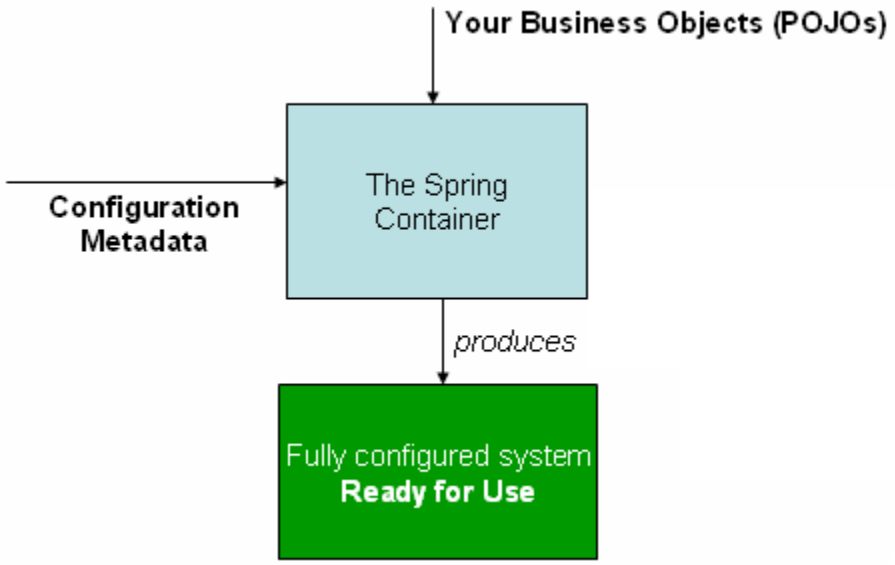
Spring Core

1. IoC container

- **IoC (Inversion of Control) container** được xem là cốt lõi của Spring Framework, giúp lập trình viên quản lý quá trình thực thi **DI (Dependency Injection)** trong ứng dụng một cách tự động.
- Spring IoC Container sẽ tạo các đối tượng, liên kết chúng lại với nhau, cấu hình các đối tượng và quản lý vòng đời của chúng từ lúc tạo ra cho đến lúc bị hủy. Các đối tượng tạo nên xương sống của ứng dụng và được quản lý bởi Spring IoC Container được gọi là các **bean**.
- Hai package **org.springframework.beans** và **org.springframework.context** là những nền tảng của Spring IoC container
    - Interface **BeanFactory** của **package org.springframework.beans** cung cấp framework cấu hình và chức năng cơ bản.
    - Interface **ApplicationContext** của package **org.springframework.context:** ApplicationContext được xây dựng từ BeanFactory nhưng nó có them một số chức năng mở rộng như tích hợp với Spring AOP, xử lý message, context cho web application. Chúng ta có thể xem Application Context giống như là một sub interface bổ trợ cho Bean Factory. Do đó trong thực tế, chúng ta sẽ ưu tiên sử dụng ApplicationContext hơn là BeanFactory.



Tổng quan luồng hoạt động của Spring IoC Container. Spring IoC Container (mà đại diện là ApplicationContext) sau khi nạp các POJO, sẽ kết hợp với configuration metadata (được biểu diễn bằng XML hoặc Java code) để khởi tạo, cấu hình và lắp ráp các bean.Lúc này, chúng ta sẽ có một ứng dụng được cấu hình đầy đủ và có thể thực thi được.

2. Dependency Injection (DI)

- Dependency Injection (có thể dịch tiêm các thành phần phụ thuộc) là một sức mạnh nổi bật của Spring Framework.
- Dependency Injection là một design pattern implement theo nguyên lý Inversion of Control (IoC). DI truyền vào sự phụ thuộc giữa các đối tượng.Thay vì để đối tượng A trực tiếp khởi tạo các đối tượng B, C, D, ... thì ta sẽ tiêm (inject) các đối tượng phụ thuộc của B, C, D vào A
- Ta sẽ có một ví dụ đơn giản về Dependency Injection như sau: Bạn có một web controller có nhiệm vụ lưu thông tin gửi từ form của người dùng. Theo nguyên lý Đơn trách nhiệm nói trên, bạn không muốn lớp controller tương tác với cơ sở dữ liệu. Thay vào đó, bạn sẽ sử dụng một lớp service để làm công việc này. Như vậy, controller của bạn sẽ chỉ phải xử lý dữ liệu của form (get form data, validate data, ...) rồi gọi một phương thức của lớp service được inject để lưu dữ liệu. Controller không cần phải quan tâm hay lo lắng về kết nối cơ sở dữ liệu, pooling hay bảng nào sẽ được update. Cũng như service không cần phải biết request có những thông tin gì.
- Spring Framework hỗ trợ 2 cách thực hiện Dependency Injection (DI):
  + DI thông qua constructor
  + DI thông qua setter

- **Chúng ta nên thực hiện DI thông qua constructor hay setter?**

  Chúng ta hoàn toàn có thể sử dụng kết hợp cả 2 cách.Nhưng theo khuyến nghị của Spring, chúng ta nên:

  - DI thông qua constructor với các dependency bắt buộc phải có.
  - DI thông qua setter với các dependency không bắt buộc phải có.
  - Điều này có thể giúp DI trở nên linh động hơn. Tuy nhiên, có một số điều nên tránh cũng như hạn chế khi thực hiện DI là:

DI thông qua constructor: một constructor có thể có quá nhiều tham số => code smell =>cần phải refactor code.

DI thông qua setter: dependency instance có thể bị null =>cần phải check null trước khi thực hiện DI.

3. Autowired

- Spring cho phép chúng ta thực hiện DI một cách tự động thông qua cơ chế autowiring. Chúng ta chỉ cần sử dụng annotation **org.springframework.beans.factory.annotation.Autowired**

- Có 4 autowiring mode là:
    - No (mặc định): autowiring bị off.
    - By Type
    - By Name
    - Constructor

3.1 Autowiring by Type

- Spring IoC container tìm kiếm bean theo class type đã được autowired:
    - Nếu trong container chỉ có duy nhất một bean phù hợp, autowiring sẽ được thực hiện.
    - Nếu trong container có nhiều hơn một bean phù hợp, một fatal error sẽ được ném ra và autowiring không được thực hiện.
    - Nếu trong container không có bean nào phù hợp, không có error nào được ném ra và autowiring cũng không được thực hiện.

    - @**Autowired** được đặt ở trên thuộc tính của class.

```java
@RestController
public class GreetingController {

    // Autowired by type
    @Autowired
    private GreetingService greetingService;

    @GetMapping("/greeting")
    public String greeting() {

        return greetingService.hello() + greetingService.goodbye();
    }
}
```

3.2 Autowiring by Name

- Spring IoC container tìm kiếm bean có cùng tên với thuộc tính.
- @Autowired được đặt ở trên setter.

```java
@RestController
public class GoodbyeController {

    private GoodbyeService goodbyeService;

    /**
     * DI by setter
     */
    @Autowired
    public void setGoodbyeService(GoodbyeService goodbyeService) {
        this.goodbyeService = goodbyeService;
    }

    @GetMapping("/goodbye")
    public String goodbye() {

        return goodbyeService.goodbye();
    }
}
```

3.3 Autowiring by Constructor

- Spring IoC container tìm kiếm bean theo class type của các đối số trong constructor đã được autowired. Với mỗi đối số, nếu trong container không có bean nào hoặc có nhiều hơn một bean phù hợp thì một fatal error sẽ được ném ra. Ngược lại, autowiring sẽ được thực hiện.
- @Autowired được đặt ở trên constructor. Các thuộc tính dependency nên được set là final để đảm bảo thread-safety.

```
@RestController
public class HelloController {

    private HelloService helloService;

    /**
     * DI by Constructor
     *
     */
    @Autowired
    public HelloController(HelloService helloService) {
        this.helloService = helloService;
    }

    @GetMapping("/hello")
    public String hello() {

        return helloService.hello();
    }
}
```

4. Qualifying

- Như chúng ta đã biết, lớp bean có thể implement một interface.
- Khi khởi tạo bean, Spring IoC container có thể tự động tìm kiếm implementation của một interface. Tuy nhiên, nếu interface có nhiều implementation thì khi autowiring, Spring IoC container sẽ không thể biết chúng ta đang muốn inject implementation nào. Ngoại lệ org.springframework.beans.factory.NoUniqueBeanDefinitionException sẽ được ném ra.
- Lúc này, chúng ta cần sử dụng đến annotation org.springframework.beans.factory.annotation.Qualifier với thuộc tính value là tên của bean (implementation).

```
@Repository("HelloGreetingRepository")
public class HelloGreetingRepositoryImpl implements GreetingRepository {

    public String sayGreeting() {

        return "GREETING HELLO !!!";
    }
}
```

```java
@Repository("GoodbyeGreetingRepository")
public class GoodbyeGreetingRepositoryImpl implements GreetingRepository {

    public String sayGreeting() {

        return "GREETING GOODBYE !!!";
    }
}
```

```java
@Service
public class GreetingService {

    @Autowired
    @Qualifier("HelloGreetingRepository")
    private GreetingRepository helloGreetingRepository;

    @Autowired
    @Qualifier("GoodbyeGreetingRepository")
    private GreetingRepository goodbyeGreetingRepository;

    public String hello() {

        return helloGreetingRepository.sayGreeting();
    }

    public String goodbye() {

        return goodbyeGreetingRepository.sayGreeting();
    }
}
```

❖ Tài liệu tham khảo

https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-dependencies

https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-autowired-annotation

https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-autowired-annotation-qualifiers