스프링부트로 **RestFulAPI** 구현하기

2장 SPRING DI/IOC

- **DI / IOC**
- **@Configuraion**
- **@Bean**

박명회

2장 SPRING DI / IOC

의존성 주입(Dependency Injection, DI)과 제어의 역전(Inversion of Control, IoC) 개념을 그림으로 설명하면 이해하기 더 쉬울 수 있습니다. 아래는 이 두 개념의 기본적인 설명과 그림입니다.

의존성 주입(Dependency Injection, DI)

의존성 주입은 객체가 자신이 필요로 하는 의존성을 외부에서 주입받는 방식입니다. 객체는 자신이 의존하고 있는 다른 객체를 생성하거나 관리하지 않고, 외부에서 주입받습니다.

DI의 흐름

1. **클라이언트 객체**는 자신의 의존성을 선언합니다.
2. **컨테이너**는 의존성 객체를 생성하거나 찾아서 클라이언트 객체에 주입합니다.
3. **클라이언트 객체**는 주입받은 의존성 객체를 사용합니다.

제어의 역전(Inversion of Control, IoC)

제어의 역전은 객체 생성과 의존성 관리를 프레임워크 또는 컨테이너가 맡는 디자인 패턴입니다. 즉, 객체의 생명주기와 의존성 관리의 제어를 애플리케이션 코드에서 프레임워크나 컨테이너로 역전시킵니다.

IoC의 흐름

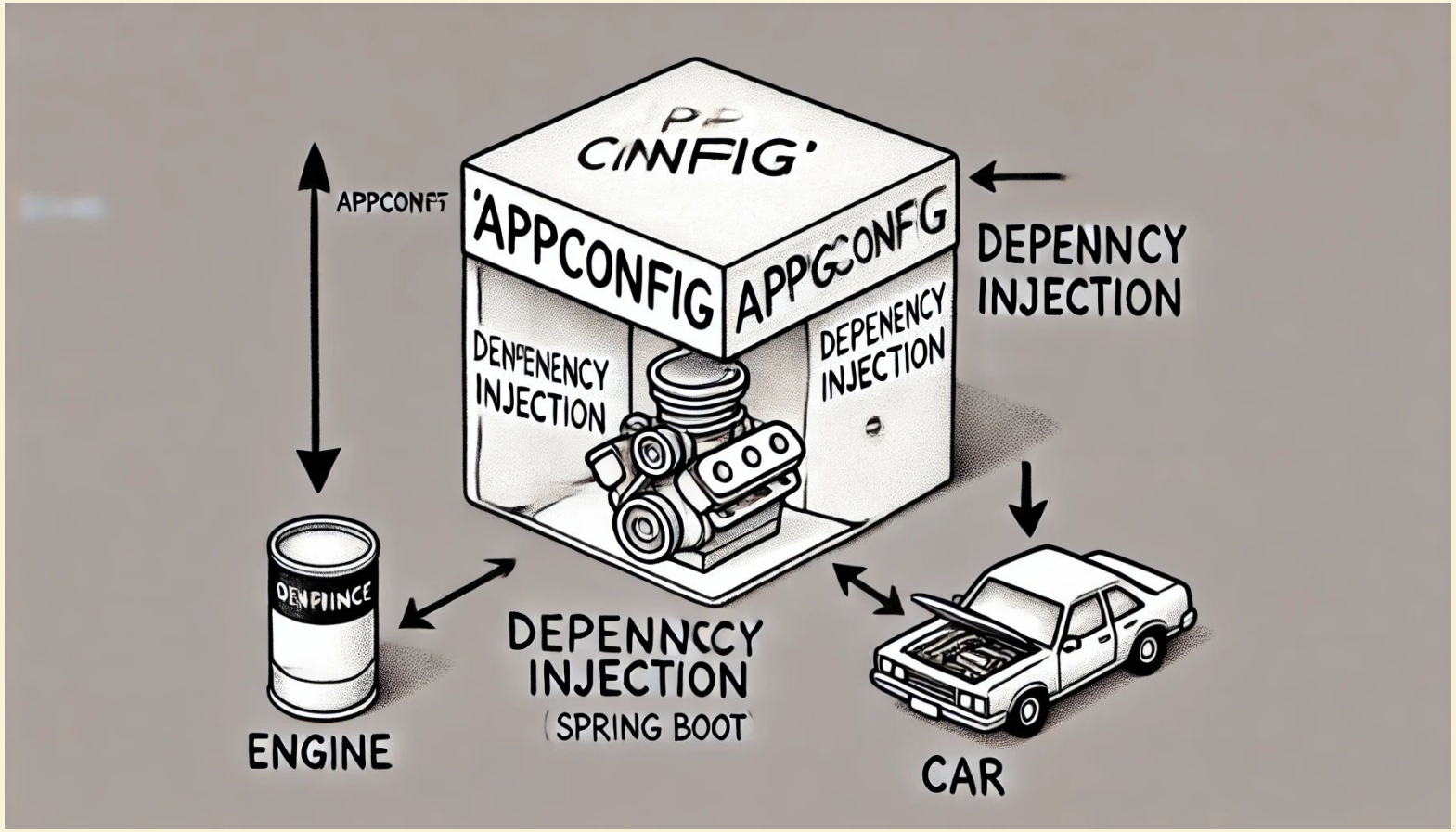
1. **애플리케이션 코드**는 단순히 기능을 요청합니다.
2. **컨테이너**가 객체를 생성하고, 초기화하며, 의존성을 주입합니다.
3. **애플리케이션 코드**는 생성된 객체를 사용합니다.

2장 SPRING DI / IOC

AppConfig 클래스에 **engine()**과 **car()** 메서드가 있습니다.

engine() 메서드는 **Engine** 객체를 생성합니다.

car() 메서드는 **Engine** 객체를 사용하여 **Car** 객체를 생성합니다.



2장 @Configuration과 @Bean

자바에서 **@Configuration**을 사용하여 객체를 조립하는 예제를 제공해드리겠습니다. **@Configuration**은 **Spring Framework**에서 제공하는 어노테이션으로, 설정 클래스를 정의하고 **Spring**의 **IoC** 컨테이너가 사용할 빈을 정의하는 데 사용됩니다.

아래는 **@Configuration**을 사용하여 객체를 조립하는 기본적인 예제입니다.

1. **의존성 관리: Spring**의 의존성 주입을 사용할 수 있도록 **spring-context** 라이브러리를 포함합니다.
2. **설정 클래스 작성: @Configuration** 어노테이션을 사용하여 **Spring**의 설정 클래스를 정의합니다.
3. **빈 정의:** 메서드에 **@Bean** 어노테이션을 사용하여 빈을 정의합니다.

예제 코드

먼저, 두 개의 간단한 클래스를 정의하겠습니다: **Car**와 **Engine**.

2장 @Configuration과 @Bean

Engine.class

```
public class Engine {
    private String type;

    public Engine(String type) {
        this.type = type;
    }

    public String getType() {
        return type;
    }
}
```

Car.class

```
public class Car {
    private Engine engine;

    public Car(Engine engine) {
        this.engine = engine;
    }

    public Engine getEngine() {
        return engine;
    }
}
```

2장 @Configuration과 @Bean

@Configuration을 사용하여 **Car**와 **Engine** 객체를 조립하는 설정 클래스를 작성합니다.

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {

    @Bean
    public Engine engine() {
        return new Engine("V8");
    }

    @Bean
    public Car car() {
        return new Car(engine());
    }
}
```

2장 @Configuration과 @Bean

이제 **AppConfig**를 사용하여 **Spring** 컨테이너를 설정하고 **Car** 빈을 가져오는 예제입니다.

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {
    public static void main(String[] args) {
        // Spring 컨테이너 생성
        ApplicationContext context =
                new AnnotationConfigApplicationContext(AppConfig.class);

        // Car 빈 가져오기
        Car car = context.getBean(Car.class);
        System.out.println("Engine Type: " + car.getEngine().getType());
    }
}
```

2장 Maven 라이브러리추가

pom.xml 추가 부분

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.6.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.28</version>
    </dependency>
</dependencies>
```

2장 @Configuration과 @Bean

AppConfig 클래스: @Configuration 어노테이션이 붙어 있는 이 클래스는 **Spring**의 설정 클래스로, 빈을 정의하는 역할을 합니다. **engine()** 메서드는 **Engine** 객체를 생성하고, **car()** 메서드는 **Car** 객체를 생성하면서 **engine()** 메서드를 호출하여 의존성을 주입합니다.
Main 클래스: AnnotationConfigApplicationContext를 사용하여 **Spring** 컨테이너를 초기화하고, **Car** 빈을 가져와서 **Engine**의 타입을 출력합니다.

감사합니다