Chap02. IoC Container

1. IoC Container

1-1. IoC & IoC Container 정의

1-1-1. IoC(Inversion of Control)란?

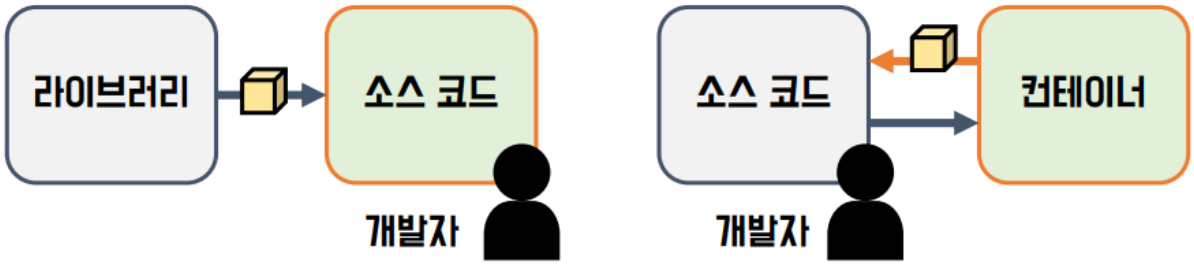
> 💡 제어의 역전(IoC, Inversion of Control)은 일반적인 프로그래밍에서, 프로그램의 제어 흐름 구조가 뒤바뀌는 것을 의미한다.

⇒ 객체의 `생성`과 `관리`, 객체 간의 `의존성 처리` 등을 프레임워크에서 대신 처리해주는 것이 IoC의 대표적인 예이다.

1-1-2. IoC Container란?

> 💡 IoC Container는 IoC를 구현한 구체적인 프레임워크를 말한다.
> IoC Container를 사용하면 객체의 생성, 초기화, 의존성 처리 등을 자동으로 수행할 수 있다.

⇒ 대표적인 IoC Container로는 Spring Framework의 `ApplicationContext`가 있다.



1-2. Spring IoC Container

1-2-1. Bean이란?

💡 Bean은 Spring IoC Container에서 관리되는 객체를 말한다.

⇒ 스프링은 Bean을 `생성`하고, `초기화`하고, `의존성 주입`하고, `제거`하는 등의 일을 IoC Container를 통해 자동으로 처리할 수 있다.

1-2-2. Bean Factory란?

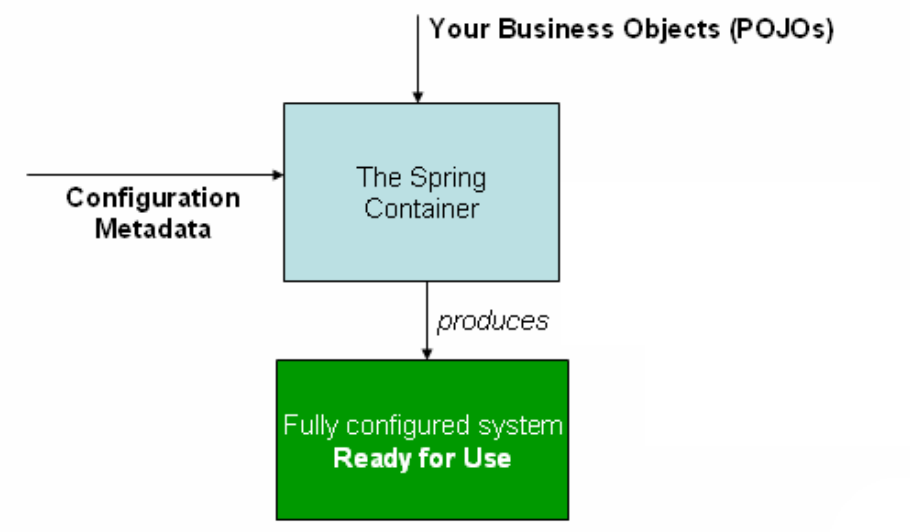
💡 BeanFactory는 Spring IoC Container의 가장 기본적인 형태로, Bean의 생성, 초기화, 연결, 제거 등의 라이프사이클을 관리한다.

⇒ 이를 위해 `Configuration Metadata`를 사용한다.

1-2-3. Configuration Metadata란?

💡 BeanFactory가 IoC를 적용하기 위해 사용하는 설정 정보이다.

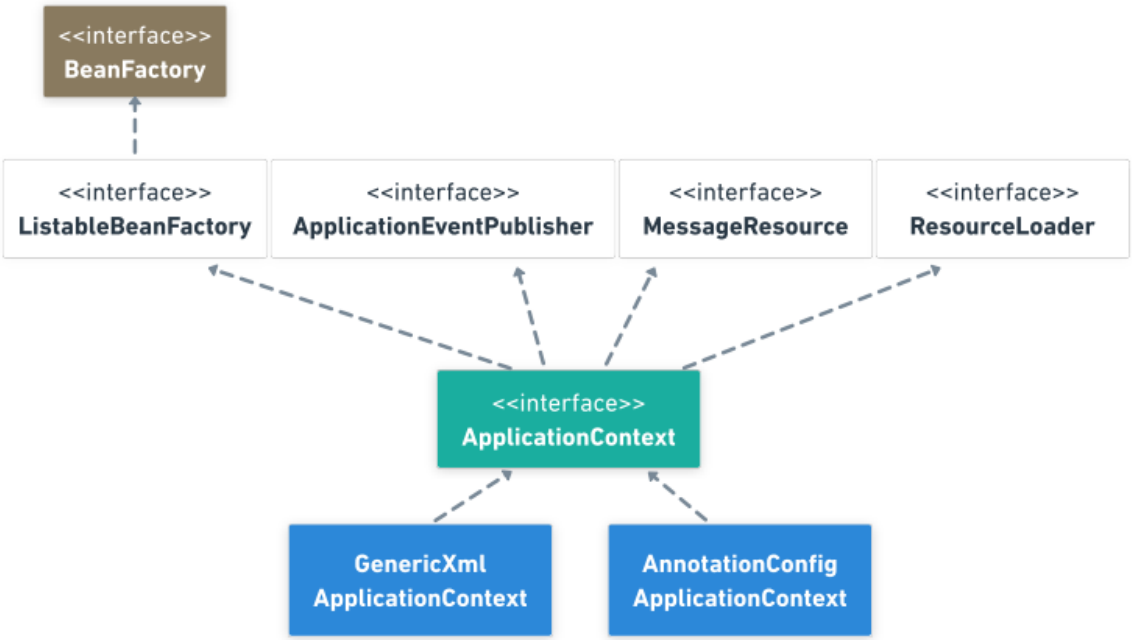
⇒ 설정 메타 정보는 IoC 컨테이너에 의해 관리 되는 Bean 객체를 생성하고 구성할 때 사용 된다.



https://docs.spring.io/spring-framework/reference/core/beans/basics.html

1-2-4. Application Context란?

💡 BeanFactory를 확장한 IoC 컨테이너로 Bean을 등록하고 관리하는 기능은 BeanFactory와 동일하지만 스프링이 제공하는 각종 부가 기능을 추가로 제공한다.



- `ListableBeanFactory` : BeanFactory가 제공하는 모든 기능을 포함한다.
- `ApplicationEventPublisher` : `이벤트 처리(Event Handling)` 기능을 제공한다.
- `MessageSource` : `국제화(i18n)` 를 지원하는 메세지를 해결하는 부가 기능을 제공한다.
- `ResourceLoader` : `리소스 핸들링(Resource Handling)` 기능을 제공한다.
- `GenericXmlApplicationContext` : ApplicationContext를 구현한 클래스. XML MetaData Configuration을 읽어 컨테이너 역할을 수행한다.
- `AnnotationConfigApplicationContext` : ApplicationContext를 구현한 클래스. Java MetaData Configuration을 읽어 컨테이너 역할을 수행한다.

2. IoC Container 사용하기

아래 코드는 테스트에 공통적으로 사용할 `MemberDTO` 클래스이다.

```
@Getter @Setter @ToString
@AllArgsConstructor
public class MemberDTO {

    private int sequence;     //회원번호
    private String id;        //아이디
    private String pwd;       //비밀번호
    private String name;      //이름

}
```

2-1. XML-based Configuration

2-1-1. GenericApplicationContext

`GenericXmlApplicationContext` 는 Spring IoC Container 중 하나이다. XML 형태의 Configuration Metadata를 사용하여 Bean을 생성한다.

```
/* GenericXmlApplicationContext 클래스를 사용하여 ApplicationContext를 생성한다.
 * 생성자에 XML 설정 메타 정보를 인자로 전달한다. */
ApplicationContext context
  = new GenericXmlApplicationContext("section01/xmlconfig/spring-context.xml"
    /*XML Configuration Metadata 파일 경로*/);
```

2-1-2. XML 기반 Configuration Metadata

XML 형태의 Configuration Metadata 파일은 다음과 같이 작성할 수 있다. 다음 예제에서는 `MemberDTO` 클래스의 객체를 Bean으로 등록하고 있다.

```
<?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">

    <bean id="member" class="com.ohgiraffers.common.MemberDTO">
        <!-- int 타입의 첫 번째 파라미터에 1 값을 전달 -->
        <constructor-arg index="0" value="1"/>
        <!-- String 타입의 id 파라미터에 "user01" 값을 전달 -->
        <constructor-arg name="id" value="user01"/>
        <!-- String 타입의 세 번째 파라미터에 "pass01" 값을 전달 -->
        <constructor-arg index="2"><value>pass01</value></constructor-arg>
        <!-- String 타입의 name 파라미터에 "홍길동" 값을 전달 -->
        <constructor-arg name="name"><value>홍길동</value></constructor-arg>
    </bean>

</beans>
```

`<beans>` 태그는 Bean 설정 정보를 담고 있는 메타데이터를 정의하는 태그이다.

- xmlns : XML Namespace를 정의
- xmlns:xsi : XML Schema Instance Namespace를 정의
- xsi:schemaLocation : XML Schema Location을 정의
- XML Schema는 XML 문서의 구조와 유효성을 검사하기 위한 언어로, XML Schema Definition(XSD) 파일을 통해 정의

`<beans>` 태그 내부에 `<bean>` 태그를 사용해 하나의 bean 설정 정보를 작성할 수 있다.

- id : Bean의 이름을 정의
- class : 객체의 클래스를 지정

`<constructor-arg>` 태그는 생성자를 호출할 때 전달할 인자를 정의한다. 만약 `<bean>` 태그 내부에 아무 것도 작성하지 않으면 기본 생성자를 사용한다는 의미이다.

- index : 메서드의 파라미터 순서(index)로 전달
- name : 파라미터 이름으로 전달
- value : 파라미터 값으로 전달

2-1-3. GenericApplicationContext(스프링 컨테이너) 생성 테스트

`GenericApplicationContext`에 `bean` 이 등록 되고 생성 되었는지 확인한다.

1. bean의 id를 이용해서 bean을 가져오는 방법

```
MemberDTO member = (MemberDTO) context.getBean("member");
System.out.println(member);
```

▼ 실행 결과

```
MemberDTO(sequence=1, id=user01, pwd=pass01, name=홍길동)
```

2. bean의 클래스 메타 정보를 전달하여 가져오는 방법

```
MemberDTO member = context.getBean(MemberDTO.class);
System.out.println(member);
```

▼ 실행 결과

```
MemberDTO(sequence=1, id=user01, pwd=pass01, name=홍길동)
```

3. bean의 id와 클래스 메타 정보를 전달하여 가져오는 방법

```
MemberDTO member = (MemberDTO) context.getBean("member");
System.out.println(member);
```

▼ 실행 결과

```
MemberDTO(sequence=1, id=user01, pwd=pass01, name=홍길동)
```

2-2. Java-based Configuration

2-2-1. AnnotationConfigApplicationContext

`AnnotationConfigApplicationContext`는 Spring IoC Container 중 하나이다. Java Configuration 형태의 Configuration Metadata를 사용하여 Bean을 생성한다.

```
/* AnnotationConfigApplicationContext클래스를 사용하여 ApplicationContext를 생성한다.
 * 생성자에 @Configuration 어노테이션이 달린 설정 클래스의 메타 정보를 전달한다. */
ApplicationContext context
  = new AnnotationConfigApplicationContext(ContextConfiguration.class);
```

2-2-2. Java 기반 Configuration Metadata

Java Configuration 형태의 Configuration Metadata 파일은 다음과 같이 작성할 수 있다. 다음 예제에서는 `MemberDTO` 클래스의 객체를 Bean으로 등록하고 있다.

```
@Configuration
public class ContextConfiguration {

    @Bean(name="member")
    public MemberDTO getMember() {

        return new MemberDTO(1, "user01", "pass01", "홍길동");
    }

}
```

`@Configuration` 어노테이션은 해당 클래스가 빈을 생성하는 클래스임을 표기한다.

`@Bean` 어노테이션은 해당 메소드의 반환 값을 스프링 컨테이너에 빈으로 등록한다는 의미이다.

- 이름을 별도로 지정하지 않으면 메소드 이름을 bean의 id로 자동 인식한다.
- `@Bean("myName")` 또는 `@Bean(name="myName")` 의 형식으로 bean의 id를 설정할 수 있다.

2-2-3. AnnotationConfigApplicationContext(스프링 컨테이너) 생성 테스트

`AnnotationConfigApplicationContext`에 `bean` 이 등록 되고 생성 되었는지 확인한다.

```
MemberDTO member = context.getBean("member", MemberDTO.class);
System.out.println(member);
```

▼ 실행 결과

```
MemberDTO(sequence=1, id=user01, pwd=pass01, name=홍길동)
```

2-3. Annotation-based Configuration

2-3-1. @ComponentScan이란?

- base package로 설정 된 하위 경로에 특정 어노테이션을 가지고 있는 클래스를 bean으로 등록하는 기능이다.
- @Component 어노테이션이 작성 된 클래스를 인식하여 bean으로 등록한다.
- 특수 목적에 따라 세부 기능을 제공하는 @Controller, @Service, @Repository, @Configuration 등을 사용한다.

@Component	객체를 나타내는 일반적인 타입으로 <bean> 태그와 동일한 역할
@Controller	프리젠테이션 레이어. 웹 어플리케이션에서 View에서 전달된 웹 요청과 응답을 처리하는 클래스 EX) Controller Class
@Service	서비스 레이어, 비즈니스 로직을 가진 클래스 EX) Service Class
@Repository	퍼시스턴스(persistence) 레이어, 영속성을 가지는 속성(파일, 데이터베이스)을 가진 클래스 EX) Data Access Object Class
@Configuration	빈을 등록하는 설정 클래스

2-3-2. @Component 어노테이션으로 자동 빈 등록하기

아래 코드는 테스트에 공통적으로 사용할 `MemberDAO` 클래스이다.

```
@Component
public class MemberDAO {

    private final Map<Integer, MemberDTO> memberMap;

    public MemberDAO() {
        memberMap = new HashMap<>();

        memberMap.put(1, new MemberDTO(1, "user01", "pass01", "홍길동"));
        memberMap.put(2, new MemberDTO(2, "user02", "pass02", "유관순"));
    }

    /* 매개변수로 전달 받은 회원 번호를 map에서 조회 후 회원 정보를 리턴하는 메소드 */
    public MemberDTO selectMember(int sequence) {
        return memberMap.get(sequence);
    }

    /* 매개변수를 전달 받은 회원 정보를 map에 추가하고 성공 실패 여부를 boolean으로 리턴하는 메소드 */
    public boolean insertMember(MemberDTO newMember) {

        int before = memberMap.size();

        memberMap.put(newMember.getSequence(), newMember);

        int after = memberMap.size();

        return after > before;
```

```
    }
}
```

bean으로 등록하고자 하는 클래스 위에 `@Component` 어노테이션을 기재하면 스프링 컨테이너 구동 시 빈으로 자동 등록된다.

- 이름을 별도로 지정하지 않으면 클래스명의 첫 문자를 소문자로 변경하여 bean의 id로 자동 인식한다.
- `@Component("myName")` 또는 `@Component(value="myName")` 의 형식으로 bean의 id를 설정할 수 있다.

2-3-3. @ComponentScan어노테이션으로 base packages 설정하기

```
@ComponentScan(basePackages="com.ohgiraffers")
public class ContextConfiguration {}
```

`@ComponentScan` 어노테이션의 basePackagese 속성에 입력한 패키지가 빈 스캐닝의 범위가 된다.

스프링 컨테이너에 빈 스캐닝을 통해 `bean` 이 자동으로 등록 되고 생성 되었는지 확인한다.

```
ApplicationContext context
  = new AnnotationConfigApplicationContext(ContextConfiguration.class);

/* getBeanDefinitionNames : 스프링 컨테이너에서 생성 된 bean들의 이름을 배열로 반환한다. */
String[] beanNames = context.getBeanDefinitionNames();
for(String beanName : beanNames) {
    System.out.println("beanName : " + beanName);
}
```

▼ 실행 결과

```
...생략
beanName : contextConfiguration
beanName : memberDAO
```

2-3-4. excludeFilters

`@ComponentScan` 의 `excludeFilters` 속성을 이용해서 일부 컴포넌트를 빈 스캐닝에서 제외할 수 있다.

1. Type으로 설정하는 방법

```
@ComponentScan(basePackages="com.ohgiraffers",
  excludeFilters={
    @ComponentScan.Filter(type=FilterType.ASSIGNABLE_TYPE,
    classes={MemberDAO.class})
```

```
})
public class ContextConfiguration {}
```

2. Annotation 종류로 설정하는 방법

```
@ComponentScan(basePackages="com.ohgiraffers",
  excludeFilters={
    @ComponentScan.Filter(type=FilterType.ANNOTATION,
    classes={org.springframework.stereotype.Component.class})
})
public class ContextConfiguration {}
```

3. 표현식으로 설정하는 방법

```
@ComponentScan(basePackages="com.ohgiraffers",
  excludeFilters={
    @ComponentScan.Filter(type=FilterType.REGEX,
    pattern={"com.ohgiraffers.section03.annotationconfig.java.*"})
})
public class ContextConfiguration {}
```

빈 스캐닝에서 `MemberDAO` 클래스가 제외되어 아래 코드를 실행하면 해당 이름을 가진 빈이 없다는 오류가 발생한다.

```
ApplicationContext context = new AnnotationConfigApplicationContext(ContextConfiguration.class);

String[] beanNames = context.getBeanDefinitionNames();
for(String beanName : beanNames) {
    System.out.println("beanName : " + beanName);
}

MemberDAO memberDAO = context.getBean("memberDAO", MemberDAO.class);
```

▼ 실행 결과

```
...생략
beanName : contextConfiguration
Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException:
No bean named 'memberDAO' available
```

2-3-5. includeFilters

`@ComponentScan` 의 `includeFilters` 속성을 이용해서 일부 컴포넌트를 빈 스캐닝에 포함 시킬 수 있다.

```
@ComponentScan(basePackages="com.ohgiraffers",
  useDefaultFilters=false,
  includeFilters={@ComponentScan.Filter(type=FilterType.ASSIGNABLE_TYPE,
    classes= {MemberDAO.class})
})
public class ContextConfiguration {}
```

`useDefaultFilters` 속성의 기본 값은 true로 @Component 종류의 어노테이션을 자동으로 등록 처리 해준다. 해당 속성을 false로 변경하면 컴포넌트 스캔이 일어나지 않게 된다.

별도로 `includeFilters` 속성을 정의해 컴포넌트 스캔이 발생하도록 한다.

`excludeFilters`에서 설정하는 방식과 동일하므로 종류별 예시는 생략한다.

아래 코드를 실행하면 빈 스캐닝에서 `MemberDAO` 클래스가 포함되어 해당 이름을 가진 빈을 확인할 수 있다.

```
ApplicationContext context
  = new AnnotationConfigApplicationContext(ContextConfiguration.class);

String[] beanNames = context.getBeanDefinitionNames();
for(String beanName : beanNames) {
    System.out.println("beanName : " + beanName);
}
```

▼ 실행 결과

```
...생략
beanName : contextConfiguration
beanName : memberDAO
```

2-3-6. XML에서 Component Scan 설정

아래와 같이 XML 설정 파일에서 Component Scan의 `base package` 를 설정할 수도 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           https://www.springframework.org/schema/context/spring-context.xsd">

  <context:component-scan base-package="com.ohgiraffers"/>

</beans>
```

`<context:component-scan>` 태그는 context 네임스페이스의 기능이기 때문에 `context:` 접두어가 필요하다.

또한 XML 설정 파일의 기본 네임스페이스가 `beans`로 설정 되어 있기 때문에 `context` 네임스페이스 추가가 필요하다.

base package에 `@Component` 어노테이션을 작성한 `MemberDAO` class를 배치해 두고 아래와 같은 코드를 실행하면 빈이 등록 되어있음을 확인할 수 있다.

```
ApplicationContext context
  = new GenericXmlApplicationContext("section03/javaannotation/spring-context.xml");

String[] beanNames = context.getBeanDefinitionNames();
for(String beanName : beanNames) {
    System.out.println("beanName : " + beanName);
}
```

▼ 실행 결과

```
beanName : memberDAO
...생략
```

아래와 같이 `<context:component-scan>` 태그 내부에 `<exclude-filter>` 또는 `<include-filter>` 태그를 정의할 수 있다.

`<include-filter>` 의 경우 동일한 방식으로 정의하므로 생략하였다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           https://www.springframework.org/schema/context/spring-context.xsd">

  <context:component-scan base-package="com.ohgiraffers">
        <context:exclude-filter type="assignable" expression="com.ohgiraffers.common.MemberDAO"/>
  </context:component-scan>

</beans>
```

빈 스캐닝에서 `MemberDAO` 클래스가 제외되어 아래 코드를 실행하면 해당 이름을 가진 빈이 없다는 오류가 발생한다.

```
ApplicationContext context
  = new GenericXmlApplicationContext("section03/javaannotation/spring-context.xml");

String[] beanNames = context.getBeanDefinitionNames();
for(String beanName : beanNames) {
    System.out.println("beanName : " + beanName);
}

MemberDAO memberDAO = context.getBean("memberDAO", MemberDAO.class);
```

▼ 실행 결과

```
...생략
Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException
: No bean named 'memberDAO' available
```

정리

- XML 설정은 전통적으로 사용하던 방식으로 최근에는 Java 설정이 선호된다.
- 개발자가 직접 컨트롤 가능한 Class의 경우 @Component를 클래스에 사용하여 빈 스캐닝을 통한 자동 빈 등록을 한다.
- 개발자가 직접 제어할 수 없는 외부 라이브러리는 @Bean을 메소드에 사용하여 수동 빈 등록을 한다.
    - 다형성을 활용하고 싶은 경우에도 @Bean을 사용할 수 있다.