

IoC Container

IoC와 DI

- ▶ **Inversion of Control (제어 역전)** : 프로그램 코드 내에서 참조하려는 객체를 직접 생성하지 않고 외부의 다른 존재가 생성하여 제공한다는 개념
- ▶ 외부의 다른 존재(객체)를 **Container**라고 하며 **IoC** 오브젝트 혹은 **IoC Container**라고 한다
- ▶ **IoC의 구현 방법**
 1. **Dependency Lookup**
 - 객체가 **JNDI**를 이용하여 직접 의존하고 있는 객체를 찾는다.
 - 객체가 **JNDI** 코드를 구현해야 한다
 - Exception** 처리 구조가 복잡
 - EJB, Apache Avalon**
 2. **Dependency Injection**
 - 컨테이너 자체가 **Lookup**을 한다 (객체 내에서 **lookup** 하는 코드가 사라짐)
 - 컨테이너에 의존적인 코드를 작성하지 않아도 된다
 - 특별한 인터페이스 구현과 특정 클래스의 상속이 필요하지 않다
 - Setter / Constructor Injection**

Spring Container

- ▶ Spring Framework에서 Container 기능을 제공해 주는 클래스
- ▶ 흔히, Container 객체 혹은 Factory 객체라고 부른다
- ▶ Spring Framework를 초기화하는 역할을 한다
- ▶ XML에 등록된 Bean 들의 Life Cycle / Dependency 관리를 담당하기 때문에 Spring IoC 기능을 제공해 준다
- ▶ 종류
 1. `org.springframework.beans.factory.BeanFactory` : Bean Factory 인터페이스
 2. `org.springframework.context.ApplicationContext` : 애플리케이션 컨텍스트 인터페이스
- ▶ 2개의 인터페이스를 구현한 다수의 Container 클래스가 존재
 - ▶ XmlBeanFactory, ClassPathXmlApplicationContext, FileSystemXmlApplication, XmlWebApplicationContext (JEE 환경에서 설정이 가능)

Spring Container

- ▶ EJB의 비즈니스 서비스 컨테이너의 기능은 유지하며 복잡성을 제거한 컨테이너의 필요성
- ▶ 애플리케이션 **Life Cycle**을 관리해주는 컨테이너의 기본적인 기능을 가지며 코드 안에 컨테이너에 의존적인 부분들, 예를 들어 컨테이너에서 제공하는 **API**를 상속받거나 구현하여 코드를 작성하는 부분들을 제거
- ▶ **Lighweight(경량) Container:**
컨테이너를 이루는 파일 자체가 수 **MB**밖에 안되는 작은 사이즈이며 구동에 필요한 시간이 거의 없어 컨테이너 자체의 부하는 무시할 수준이고 컨테이너 내에 객체를 배치하는 복잡한 과정이 없음을 의미
- ▶ 컨테이너의 필요성
 - 1) 컴포넌트, 객체의 자유로운 삽입이 가능하도록 하기 위한 호출의 독립성
 - 2) 서비스를 설정하거나 찾기 위한 일관된 방법을 제시
 - 3) 개발자가 직접 싱글톤이나 팩토리를 구현할 필요 없이 단일화된 서비스의 접근 방법을 제공
 - 4) 비즈니스 객체에 부가적으로 필요한 각종 엔터프라이즈 서비스를 제공

Spring Container

: Bean Factory (XMLBeanFactory)

- ▶ BeanFactory (XMLBeanFactory)
 - ▶ 기본적인 Bean의 생성과 소멸을 담당
 - ▶ XML 파일에 기술되어 있는 정의를 바탕으로 Bean을 생성

```
BeanFactory beanFactory =  
    new XmlBeanFactory(new ClassPathResource("config/applicationContext.xml"));  
  
MyBean obj = beanFactory.getBean(MyBean.class);
```

- ▶ 생성자에 xml 파일 로딩에 필요한 Resource 객체를 넘겨 주어야 한다
- ▶ FileSystemResource와 ClassPathResource를 주로 사용한다
(ByteArrayResource, DescriptiveResource, InputStreamResource 등)

Spring Container

▶ XML 파일

- ▶ XML에 프레임워크 초기화에 필요한 **Bean** 설정을 한다
- ▶ applicationContext.xml 예시

```
<?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
    http://www.springframework.org/schema/context/spring-context-4.2.xsd">

  <!-- Bean 설정 -->
  <bean id="myBean" class="com.example.springcontainer.MyBean" />

</beans>
```

Spring Container

: 연습

- ▶ XmlBeanFactory 기반의 Spring Container를 생성하고 설정되어 있는 Bean을 사용해 보는 연습
 - ▶ 설정해야 하는 Bean (POJO)

```
public class MyBean {  
    public String getName() {  
        return "MyBean";  
    }  
}
```

- ▶ xml의 Bean 설정

```
<bean id="myBean" class="com.example.springcontainer.MyBean" />
```

- ▶ 컨테이너로부터 Bean 가져오기 (id 또는 name으로 가져오기)

```
MyBean obj = (MyBean)beanFactory.getBean("myBean");
```

- ▶ 다음 내용도 연습해 봅시다
 - ▶ 타입으로 가져오기

Spring Container

: 연습

- ▶ XmlBeanFactory 기반의 Spring Container를 생성하고 설정되어 있는 Bean을 사용해 보는 연습
 - ▶ annotation으로 설정해 보기 (applicationContext2.xml)

```
<context:annotation-config />
<context:component-scan base-package="com.example.springcontainer">
  <context:include-filter type="annotation"
    expression="org.springframework.stereotype.Component"/>
</context:component-scan>
```


Spring Container

: ApplicationContext

- ▶ BeanFactory는 기초적인 컨테이너 기능을 제공하지만 Spring Framework의 완벽한 기능을 사용하기 위해서는 ApplicationContext를 사용한다
- ▶ BeanFactory와 유사하지만, Bean을 로딩하는 방법에 차이가 있다
- ▶ 리스너로 등록된 빈에 이벤트를 발생시킬 수 있다
- ▶ 추가적인 기능과 성능 차이로 ApplicationContext를 SpringContainer로 사용한다
- ▶ 종류
 - ClassPathXmlApplicationContext
 - XmlWebApplicationContext
 - FileSystemXmlApplicationContext

Spring Container

: 연습

- ▶ ClassPathXmlApplicationContext 기반의 Spring Container를 생성하고 설정되어 있는 Bean을 사용해 보는 연습

- ▶ Container 생성

```
ApplicationContext appContext =  
    new ClassPathXmlApplicationContext("config/applicationContext.xml");
```

- ▶ applicationContext의 위치
클래스 패스 아래 config 디렉터리 밑에 있어야 한다
- ▶ 다음 실습을 해 본다
 - id와 name 함께 설정하기
 - 같은 클래스의 빈 설정에서 같은 아이디 주기
 - 같은 클래스의 빈 설정에서 타입으로 빈 받아보기

Spring Container

: 다양한 Bean 설정 연습

▶ 생성자를 이용한 값 세팅

```
public class User {  
    private String name;  
    public User(String name) {  
        this.name = name;  
    }  
}
```

▶ Bean 설정

```
<bean id="user" class="com.example.springcontainer.User">  
    <constructor-arg value="짱구" />  
</bean>
```

Spring Container

: 다양한 Bean 설정 연습

- ▶ 생성자 파라미터가 둘 이상인 경우

```
public class User {  
    private long no;  
    private String name;  
  
    public User(long no, String name) {  
        this.no = no;  
        this.name = name;  
    }  
}
```

- ▶ Bean 설정

```
<bean id="user" class="com.example.springcontainer.User">  
    <constructor-arg index="0" type="long" value="1" />  
    <constructor-arg index="1" type="java.lang.String" value="짱구" />  
</bean>
```

Spring Container

: 다양한 Bean 설정 연습

▶ Setter(...)를 이용한 세팅

```
public class User {  
    private long no;  
    private String name;  
  
    public User() {}  
  
    public void setNo(long no) {  
        this.no = no;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

```
<bean id="user" class="com.example.springcontainer.User">  
    <property name="no" value="1" />  
    <property name="name" value="짱구" />  
</bean>
```

Spring Container

: 다양한 Bean 설정 연습

▶ 객체의 주입

```
public class Friend {  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

```
public class User {  
    ...  
    private Friend friend;  
  
    public Friend getFriend() {  
        return friend;  
    }  
  
    public void setFriend(Friend friend) {  
        this.friend = friend;  
    }  
    ...  
}
```

Spring Container

: 다양한 Bean 설정 연습

▶ 객체의 주입

```
<bean id="friend" class="com.example.springcontainer.Friend">
    <property name="name" value="맹구" />
</bean>

<bean id="user" class="com.example.springcontainer.User">
    <property name="no" value="1" />
    <property name="name" value="짱구" />
</bean>
```

▶ 위의 설정은 다음과 같이 바꿀 수 있다

```
<bean id="user" class="com.example.springcontainer.User">
    <property name="no" value="1" />
    <property name="name" value="짱구" />
    <property name="friend">
        <bean class="com.example.springcontainer.Friend">
            <property name="name" value="맹구" />
        </bean>
    </property>
</bean>
```

Spring Container

: 다양한 Bean 설정 연습

▶ 집합 객체의 주입

```
public class User {  
    ...  
    private List<String> friends;  
  
    public List<String> getFriends() {  
        return friends;  
    }  
  
    public void setFriends(List<String> friends) {  
        this.friends = friends;  
    }  
    ...  
}
```

```
<bean id="user" class="com.example.springcontainer.User">  
    ...  
    <property name="friends">  
        <list>  
            <value>맹구</value>  
            <value>철수</value>  
        </list>  
    </property>  
</bean>
```


Spring Container

: 다양한 Bean 설정 연습

▶ 집합 객체의 주입 (set)

```
<bean id="player" class="Player">
  <property name="value">
    <set>
      <value>서태웅</value>
      <value>강백호</value>
    </set>
  </property>
</bean>
```

▶ 집합 객체의 주입 (map)

```
<bean id="player" class="Player">
  <property name="value">
    <map>
      <entry key="SF"><value>서태웅</value></entry>
      <entry key="PF"><value>강백호</value></entry>
    </map>
  </property>
</bean>
```