

스프링 프레임워크

1. 스프링 프레임워크

스프링 프레임워크(Spring Framework)는 자바 플랫폼을 위한 오픈 소스 애플리케이션 프레임워크이며 간단히 스프링(Spring)이라고도 불린다. 동적인 웹 사이트를 개발하기 위한 다양한 서비스를 제공하고 있고, 대한민국 공공기관의 전자정부 표준 프레임워크 기반 기술로서 사용되고 있다.

이 프레임워크는 2003년 6월에 최초로 아파치 2.0 라이선스로 공개되었으며, 현재 표준 프레임워크 기반 기술로 다양한 실무 프로젝트 개발에 적용되고 있다.

※ 스프링은 다음과 같은 특징이 있다.

- 경량 컨테이너로서 자바 객체를 직접 관리한다. 각각의 객체 생성, 소멸과 같은 생명 주기를 관리하며, 스프링으로부터 필요한 객체를 얻어올 수 있다.
- 스프링은 POJO(Plain Old Java Object: 경량의 자바 객체 또는 별도로 종속되지 않는 자바 객체를 의미) 방식의 프레임워크이다. 일반적인 J2EE에 비하여 기존에 존재하는 라이브러리 등의 지원이 용이하고 객체가 가볍다.
- 스프링은 제어 반전(Inversion Of Control)을 지원한다. 컨트롤의 제어권이 사용자가 아니라 프레임워크에서 필요한 사용자의 코드를 호출한다.
- 스프링은 의존성 주입(Dependency Injection: **객체 사이의 의존 관계가 자기 자신이 아닌 외부에 의해 설정된다는 개념**)을 지원한다. 각각의 계층이나 서비스들 간에 의존성은 설정 파일을 통하여 프레임워크가 서로 연결시켜준다.
- 스프링은 관점 지향 프로그래밍(Asspect-Oriented Programming)을 지원한다.
- 스프링은 영속성과 관련된 다양한 API를 지원한다. JDBC, iBatis(MyBatis)나 Hibernate 등 완성도 높은 데이터베이스 처리 라이브러리와 연결할 수 있는 인터페이스를 제공한다.
- 스프링은 확장성이 높다. 스프링 프레임워크에 통합하기 위해 간단하게 기존 라이브러리 사용이 가능하기 때문에 수많은 라이브러리가 이미 스프링에서 지원되고 있고 스프링에서 사용되는 라이브러리를 별도로 분리하기도 용이하다

(1) 프레임워크란? [프레임워크이란 잘 정의된 약속된 구조의 클래스의 집합을 의미한다.]

사전적 의미로는 "어떠한 것을 이루는 뼈대, 기본 구조"를 뜻한다.

소프트웨어에서의 프레임워크란 소프트웨어의 특정 문제를 프로그램으로 쉽게 그리고 편리하게 개발할 수 있도록 미리 뼈대를 이루는 클래스와 인터페이스를 제작하여 이것들을 모아둔 것이라고 할 수 있다. 다시 말해 어떤 영역의 API들을 사용하기 편리한 형태로 포장해 놓은 것이라 할 수 있다.

간단한 사례를 통해 프레임워크의 중요성을 확인해본다.

두명의 개발자에게 인형을 만들도록 지시하고 모든 권한을 각 개발자에게 위임하여 자유롭게 만들도록 한다. 그러면 각 개발자는 다음과 같이 자신이 가진 경험과 기술력을 활용하여 각각 인형을 만들 것이고 이렇게 만들어진 인형이 시장에 판매되었다. 그런데 A 개발자의 인형을 구매한 고객이 인형을 가지고 놀다가 오른쪽 다리가 부러졌고 고객은 인형에 대한 수리를 요청했다. 이때 인형을 개발한 A개발자가 회사를 그만뒀거나 다른 부서로 이동했을 수도 있다. 아니면 다른 업무로 인해 시간적 여유가 없다면 어쩔 수 없이 남아있는 B 개발자가 수리를 담당할 것이다. 이 부분에서 문제가 발생할 소지가 있다. 만약 B 개발자가 A 개발자와 의사소통이 원활하고 시간적 여유가 충분하다면 A 개발자가 만든 인형의 구조를 정확하게 파악하고 고객의 요구사항에 맞게 수리할 수 있을 것이다. 하지만 그렇지 못하다면 자신이 경험과

기술력에 의존하여 수리할 수 밖에 없다. 그리고 그 결과는 다음과 같이 이상한 구조가 될 가능성이 매우 높다. 왜냐하면 B 개발자가 오랜 시간 동안 만들어온 인형의 구조와 A 개발자가 만든 인형의 구조는 다르기 때문이다. 당연히 고객의 입장에서는 처음에 구매한 인형과 다른 모습으로 변형된 인형을 보면서 불만을 제기할 것이다.

시스템을 개발하는 과정에서 대부분 개발자들은 산출물에 입각해서 개발하므로 아키텍처의 일관성이 잘 유지된다. 하지만 유지보수 과정에서 인력과 시간 부족으로 인해 산출물은 무시되기 쉽고, 개발자들의 경험에 의존하여 유지보수가 진행되는 경우가 경우가 많다.

프레임워크는 이런 문제를 근본적으로 해결해 준다. 프레임워크는 애플리케이션을 개발하든 개발에서 기본이 되는 뼈대나 틀을 제공한다. 즉 개발자에게 모든 것을 위임하는 것이 애플리케이션의 기본 아키텍처는 프레임워크가 제공하고, 그 뼈대에 살을 붙이는 작업만 개발자가 하는 것이다.

1.1 프레임워크의 장단점

잘 만들어진 프레임워크를 사용하면 애플리케이션에 대한 분석, 설계, 구현 모두에서 재사용성이 증가하는데, 이를 통해 다음과 같은 장점들을 얻을 수 있다.

- 프레임워크의 장점- 개발자의 할 일을 줄여준다.
- 정해진 틀 안에서 코딩하기 때문에 프로그램의 가독성이 높아지므로 유지보수가 용이하다.
- 선언적인 방법을 도입하여 프로그램의 유지보수가 용이하다.

① 빠른 구현 시간

프레임워크를 사용하면 아키텍처에 해당하는 골격 코드를 프레임워크에서 제공한다. 따라서 개발자는 비즈니스 로직만 구현하면 되므로 제한된 시간에 많은 기능을 구현 할 수 있다.

② 쉬운 관리

같은 프레임워크가 적용된 애플리케이션들은 아키텍처가 같으므로 관리하기가 쉽다. 결과적으로 유지보수에 들어가는 인력과 시간을 줄 일 수 있다.

③ 개발자들은 역량 획일화

숙련된 개발자와 초급 개발자는 지식과 경험이 다르므로 두 개발자가 만든 소스의 품질은 당연히 다를 수밖에 없다. 하지만 프레임워크를 사용하면 숙련된 개발자와 초급 개발자가 생성하는 코드가 비슷해진다. 이는 초급 개발자도 프레임워크를 통해서 세련되고 효율적인 코드를 생성해 낼 수 있다는 것이다. 결과적으로 관리자 입장에서 개발 인력을 더 효율적으로 구성할 수 있다.

④ 검증된 아키텍처의 재사용과 일관성 유지

프레임워크를 이용하여 애플리케이션을 개발하면, 프레임워크에서 제공하는 아키텍처를 이용하므로 아키텍처에 관한 별다른 고민이나 검증 없이 소프트웨어를 개발할 수 있다. 또한 이렇게 개발한 시스템은 시간이 지나도 유지보수 과정에서 아키텍처가 왜곡되거나 변형되지 않는다.

• 프레임워크의 단점

- 일관성의 가치를 지나치게 높게 평가한 나머지 코딩 패턴에 너무나 많은 구속이 따른다.

- 코드가 길어지고 복잡해질 수 있다.

1.2 자바 기반의 프레임워크

자바 기반의 프레임워크는 대부분 오픈소스 형태로 제공된다. 따라서 별도의 라이선스나 비용을 지불하지 않고 누구나 사용할 수 있으며, 기존의 프레임워크를 이용하여 자신만의 프레임워크를 구축할 수도 있다.

대표적인 자바 기반의 프레임워크로는 다음과 같은 것들이 있다.

처리 영역	프레임워크	설명
Presentation	Struts	Struts 프레임워크는 UI Layer에 중점을 두고 개발된 MVC (Model View Controller) 프레임워크이다.
	Spring(MVC)	Struts와 동일하게 MVC 아키텍처를 제공하는 UI Layer 프레임워크이다. 하지만 Struts처럼 독립된 프레임워크는 아니고 Spring 프레임워크에 포함되어 있다.
Business	Spring(IoC, AOP)	Spring은 컨테이너 성격을 가지는 프레임워크이다. Spring의 IoC 와 AOP 모듈을 이용하여 Spring 컨테이너에서 동작하는 엔터프라이즈 비즈니스 컴포넌트를 개발할 수 있다.
Persistence	Hibernate or JPA	Hibernate는 완벽한 ORM(Object Relation Mapping) 프레임워크이다. ORM 프레임워크는 SQL 명령어를 프레임워크가 자체적으로 생성하여 DB 연동을 처리한다. JPA는 Hibernate를 비롯한 모든 ORM의 공통 인터페이스를 제공하는 자바 표준 API이다.
	Mybatis or Ibatis	Ibatis 프레임워크는 개발자가 작성한 SQL 명령어와 자바 객체 (VO 혹은 DTO)를 매핑해주는 기능을 제공하며, 기존에 SQL 명령어를 재사용하여 개발하는 프로젝트에 유용하게 적용할 수 있다. Mybatis는 Ibatis에서 파생된 상위 프레임워크이다.

1.3 이클립스에서는 STS(Spring Tool Suite) 플러그 인 설치

1.4 apache-tomcat-9.0.43 톰캣 설정

1.5 스프링 환경설정 및 프로젝트 생성

2. 스프링 프레임워크 필요성

스프링 프레임워크는 로드 존슨(Rod Johnson)이 2004년에 만든 오픈소스 프레임워크다. 스프링 프레임워크가 등장하기 이전에 자바 기반의 엔터프라이즈 애플리케이션은 대부분 EJB(Enterprise Java Beans)로 개발되었다. 그러나 EJB 기술은 EJB 컨테이너가 제공하는 많은 기능과 나름의 장점에도 불구하고 다음과 같은 여러 가지 문제점 때문에 개발자들로부터 외면 받을 수밖에 없었다.

우선 EJB는 스펙이 너무 복잡해서 학습에 시간이 필요하며 개발 및 유지보수 역시 복잡하고 힘들다. 그리고 EJB의 문제는 아마도 기술 자체의 문제라기보다는 EJB를 EJB답게 사용하기가 어려워서일 수도 있다. EJB를 제대로 사용하려면 EJB의 성능을 유지해주고, 유지보수의 편의성을 향상해주는 다양한 디자인 패턴을 이해하고 적용할 수 있어야 한다. 이런 디자인 패턴에 대한 이해없이 EJB를 사용하려고 하니 문제가 생기는 것이다. 하지만 이런 디자인 패턴을 반드시 사용해야 하는것도 EJB의 문제로 볼 수 있다.

우리가 지금부터 학습할 스프링 프레임워크는 평범한 POJO(Plain Old Java Object) 사용하면서도 EJB에서만 가능했던 많은 일을 가능하도록 지원한다. 따라서 EJB보다 매우 간단하다. 그리고 EJB를 사용할 때 알아야 했던 수많은 디자인 패턴 역시 신경쓰지 않아도 된다. 왜냐하면 스프링 프레임워크는 이미 많은 디자인 패턴이 적용되어 배포되므로 프레임워크를 사용하는 것 자체가 디자인 패턴을 사용하는 것이다.

어쩌면 스프링 프레임워크가 아니더라도 기존의 EJB를 대체할 만한 새로운 무언가가 등장할 수밖에 없었을지도 모른다. 스프링은 그런 시장의 요구를 충분히 반영했고 개발자들은 열광할 수 밖에 없었다.

POJO(Plain Old Java Object)란?

POJO란 말 그대로 평범한 옛날 자바 객체를 의미한다. POJO를 좀 더 쉽게 이해하기 위해서 반대로 POJO가 아닌 클래스가 무엇인지 이해하면 된다. 대표적인 Not POJO 클래스가 Servlet 클래스이다. Servlet 클래스는 우리 마음대로 만들 수 없으며, 반드시 Servlet에서 요구하는 규칙에 맞게 클래스를 만들어야 실행할 수 있다. 다음은 Servlet 클래스 작성 규칙이다.

- javax.servlet, javax.servlet.http 패키지를 import해야 한다.
- Public 클래스로 선언되어야 한다.
- Servlet, GenericServlet, HttpServlet 중 하나를 상속해야 한다.
- 기본 생성자(Default Constructor)가 있어야 한다.
- 생명주기에 해당하는 메소드를 재정의(Overriding)한다.

2.1 스프링 프레임워크의 특징

스프링의 특징을 한 줄로 서술하면, "IoC와 AOP를 지원하는 경량의 컨테이너 프레임워크"로 표현할 수 있다.

① 경량

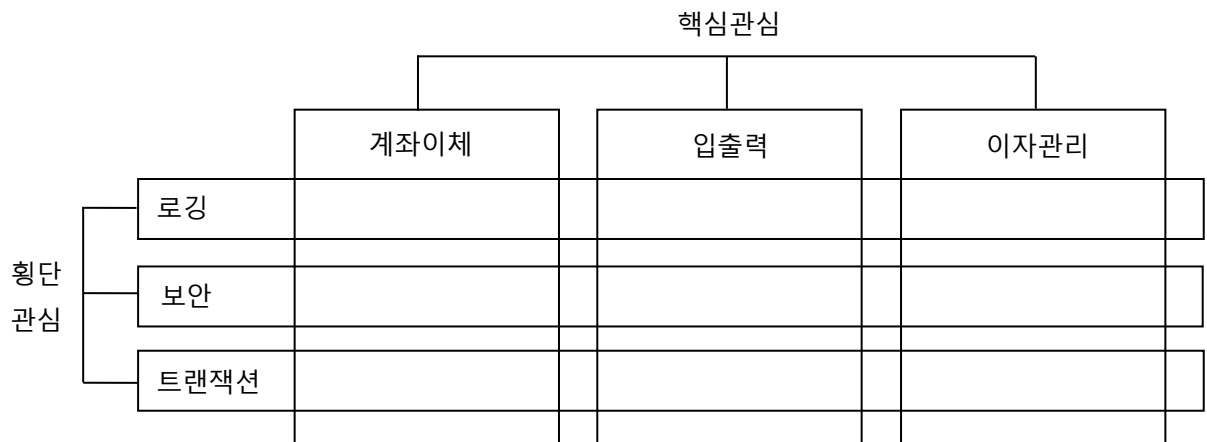
우선 스프링은 크기 측면에서 가볍다. 스프링은 여러 개의 모듈로 구성되어 있으며, 각 모듈은 하나 이상의 JAR 파일로 구성되어 있다. 그리고 몇 개의 JAR 파일만 있으면 개발과 실행이 모두 가능하다. 따라서 스프링을 이용해서 만든 애플리케이션이 배포 역시 매우 빠르고 쉽다. 스프링을 경량이라고 하는 두 번째 이유는 스프링 프레임워크가 POJO(Plain Old Java Object) 형태의 객체를 관리하기 때문이다. POJO 클래스를 구현하는데 특별한 규칙이 없는 단순하고 가벼운 객체이므로 POJO를 관리하는 것은 기존의 EJB 객체를 관리하는 것보다 훨씬 가볍고 빠를 수밖에 없다.

② 제어의 역행(Inversion of Control, IoC)

비즈니스 컴포넌트를 개발할 때, 항상 신경 쓰는 것이 바로 낮은 결합도와 높은 응집도이다. 스프링은 제어의 역행(Inversion of Control, IoC)을 통해 애플리케이션을 구성하는 객체 간의 느슨한 결합, 즉 낮은 결합도를 유지한다. IoC가 적용되기 전에는 애플리케이션 수행에 필요한 객체의 생성이나 객체와 객체 사이의 의존관계를 개발자가 직접 자바 코드로 처리했었다. 이런 상황에서는 의존관계에 있는 객체를 변경할 때 반드시 자바 코드를 수정해야 한다. 하지만 IoC가 적용되면 객체 생성을 자바 코드로 직접 처리하는 것이 아니라 컨테이너가 대신 처리한다. 그리고 객체와 객체 사이의 의존관계 역시 컨테이너가 처리한다. 결과적으로 소스에 의존관계가 명시되지 않으므로 결합도가 떨어져서 유지보수가 편리해진다.

③ 관점지향 프로그래밍(Asspect Oriented Programming, AOP)

관계지향 프로그래밍은 비즈니스 메서드를 개발할 때 핵심 비즈니스 로직과 각 비즈니스 메소드마다 반복해서 등장하는 공통 로직을 분리함으로써 응집도가 높게 개발할 수 있도록 지원한다.



공통으로 사용하는 기능들을 외부의 독립된 클래스로 분리하고, 해당 기능을 프로그램 코드에 직접 명시하지 않고 선언적으로 처리하여 적용하는 것이 관점지향 프로그래밍의 기본 개념이다. 이렇게 되면 공통 기능을 분리하여 관리할 수 있으므로 응집도가 높은 비즈니스 컴포넌트를 만들 수 있을 뿐만 아니라 유지보수를 혁신적으로 향상시킬 수 있다.

④ 컨테이너

컨테이너는 특정 객체의 생성과 관리를 담당하여 객체 운용에 필요한 다양한 기능을 제공한다. 컨테이너는 일반적으로 서버 안에 포함되어 배포 및 구동된다. 대표적인 컨테이너로 Servlet 객체를 생성하고 관리하는 'Servlet 컨테이너'와 EJB 객체를 생성하고 관리하는 'EJB 컨테이너'가 있다. 그리고 Servlet 컨테이너는 우리가 사용하는 톰캣 서버에도 포함되어 있다. 애플리케이션 운용에 필요한 객체를 생성하고 객체 간의 의존관계를 관리한다는 점에서 스프링도 일종의 컨테이너라고 할 수 있다.

2.2 IoC(Inversion of Control) 컨테이너

스프링 프레임워크를 이해하는데 가장 중요한 개념이 바로 컨테이너이다. 컨테이너의 개념은 스프링에서 가장 사용된 것은 아니며 기존의 서블릿이나 EJB 기술에서 이미 사용해왔다. 그리고 대부분 컨테이너는 비슷한 구조와 동작 방법을 가지고 있으므로 서블릿 컨테이너를 통해 스프링 컨테이너의 동작 방식을 유추해 볼 수 있다.

다음과 같이 간단한 서블릿 클래스를 만들었다고 가정하자.

```
package com.site.example;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public HelloServlet() {
        System.out.println("=====> HelloServlet 객체 생성");
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        System.out.println("=====> doGet() 메서드 호출");
    }
}
```

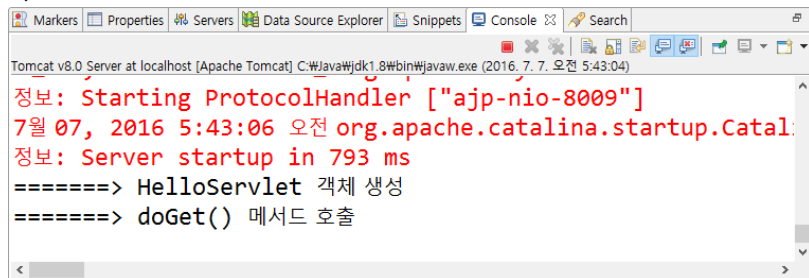
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
    <display-name>springExample</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>com.site.example.HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello.html</url-pattern>
```

```
</servlet-mapping>
```

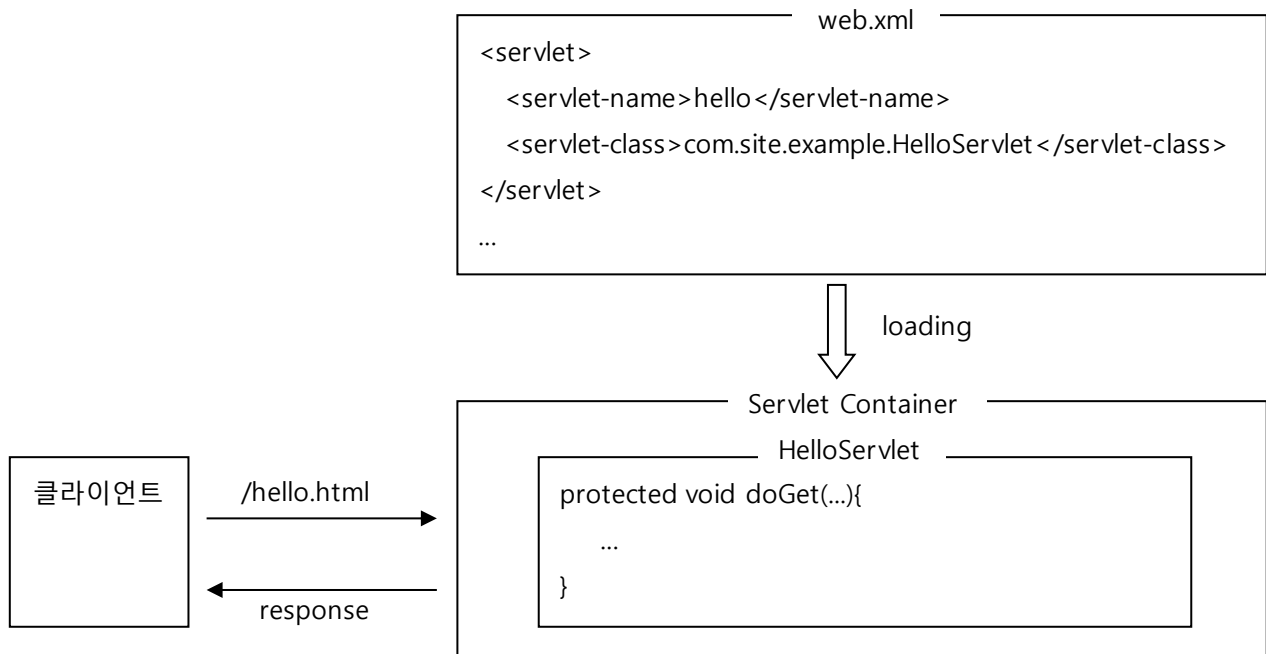
```
</web-app>
```

위 설정은 `http://localhost:8080/springExample/hello.html`라는 URL 요청을 전송하면, `hello`라는 이름으로 등록된 `com.site.example.HelloServlet` 클래스를 찾아 객체를 생성하고 실행한다는 설정이다.



서블릿은 자바로 만들어진 클래스이다. 따라서 반드시 객체 생성을 해야 하는데 어느 부분에도 명시되어 있지 않다. 그렇다면 도대체 누가 서블릿 객체를 생성했으며, `doGet()` 메서드도 호출 역시 확인할 수 없다. 그렇다면 도대체 누가 서블릿 객체를 생성했으며, `doGet()` 메서드를 호출해줬을까? 정답은 바로 서블릿 컨테이너이다.

다음은 서블릿 컨테이너가 Servlet 클래스 객체를 생성하고 운용하는 과정을 그림으로 표현한 것이다.



서블릿 컨테이너는 다음 순서에 따라 동작한다.

- ① `/WEB-INF/web.xml` 파일을 로딩하여 구동
- ② 브라우저로부터 `http://localhost:8080/springExample/hello.html` 요청 수신
- ③ `com.site.example.HelloServlet` 클래스를 찾아 객체를 생성하고 `doGet()` 메서드 호출
- ④ `doGet()` 메서드 실행 결과를 클라이언트 브라우저로 전송

이렇듯 컨테이너는 자신이 관리할 클래스들이 등록된 XML 설정파일을 로딩하여 구동한다. 그리고 클라이언트의 요청이 들어오는 순간 XML 설정 파일을 참조하여 객체를 생성하고, 객체의 생명주기를 관리한다. 스프링 컨테이너 역시 서블릿 컨테이너와 유사하게 동작하므로 위에서 살펴본 요소들과 비슷한 요소들이

존재한다.

제어의 역행(loc)은 결합도와 관련된 개념으로 이해할 수 있다. 기존에 자바 기반으로 애플리케이션을 개발할 때 객체를 생성하고 객체들 사이의 의존관계를 처리하는 것에 대한 책임은 전적으로 개발자에게 있었다. 즉 개발자가 어떤 객체를 생성할지 판단하고 객체 간의 의존관계 역시 소스 코드로 표현해야 했다.

하지만 제어의 역행이라는 것은 이런 일련의 작업들을 소스코드로 처리하지 않고 컨테이너로 처리하는 것을 의미한다. 따라서 제어의 역행을 이용하면 소스에서 객체 생성과 의존관계에 대한 코드가 사라져 결과적으로 낮은 결합도의 컴포넌트를 구현할 수 있게 한다.

3. 스프링 프레임 워크 모듈 구성

스프링 프레임워크의 모듈은 Core, Container, Date Access/Integration, Web, AOP, Aspects, Instrumentation, test의 7개 카테고리에 20여개의 모듈로 구성되어 있다.

(1) 코어 컨테이너

코어 컨테이너(Core Container)는 4개의 Core, Beans, Context, EL(Expression Language) 모듈로 구성되어 있다.

- Core 모듈과 Beans 모듈은 프레임워크의 기반이 되는 가장 핵심 부분으로 IoC와 DI 기능을 제공한다.
- Context 모듈은 Core 모듈과 Beans 모듈에서 제공하는 기반하에 구성되었다. Context 모듈은 Beans 모듈의 기능을 상속받고, 리소스 로딩, 서블릿 컨테이너와 같은 컨텍스트의 생성 기능들을 함께 제공한다.
- EL(Expression Language) 모듈은 런타임에서 객체 그래프를 조회하고 조작할 수 있는 강력한 표현언어 기능을 제공한다.

(2) 데이터 접근/통합

데이터 접근/통합(Data Access/Integration) 계층은 5개의 JDBC, ORM, OXM, JMS, 트랜잭션 모듈들로 구성되어 있다.

- JDBC 모듈은 JDBC 추상화 계층을 제공하여 데이터베이스의 종류에 따른 JDBC 관련 코딩의 에러 코드를 대신 다루어준다.
- ORM 모듈은 iBatis(MyBatis), JPA, JDO, 하이버네이트(Hibernate)와 같이 잘 알려진 객체-관계 매핑 API에 대한 통합 계층을 제공한다.
- OXM 모듈은 JAXB, Castor, XMLBeans, JiBX, XStream과 같은 객체/XML 매핑 구현을 지원하는 계층을 제공한다.
- JMS(Java Messaging Service) 모듈은 메시지의 생성과 소비 기능을 제공한다.
- Transation 모듈은 특별한 인터페이스와 POJO(Plain Old Java Object)의 클래스에 대한 트랜잭션 관리 기능을 제공한다.

(3) 웹

웹(web) 계층은 4개의 Web, Web-Servlet, Web-Struts, Web-Portlet 모듈로 구성되어 있다.

- Web 모듈은 멀티파트 파일업로드, 서블릿 리스너와 웹 지향적인 애플리케이션 컨텍스트를 사용한 IoC 컨테이너의 초기화 등 기본적인 웹 지향적인 통합기능을 제공한다.
- Web-Servlet 모듈은 웹 애플리케이션에 필요한 스프링 MVC(Model-View-Controller) 구현을 제공하며, JSP에 대한 뷰 연동을 지원한다.
- Web-Struts 모듈은 스프링 애플리케이션고 스트럿츠의 연동 기능을 제공하며, 스프링 3.0부터 폐기되었다.
- Web-Portlet 모듈은 포틀릿 환경에서 사용되는 MVC 구현과 웹-서블릿 모듈 기능의 미러(mirror) 기능을 제공한다.

(4) AOP와 Instrumentation

- AOP 모듈은 AOP Alliance 규약에 호환되는 관점-지향 프로그래밍 구현체로서 메서드 인터셉트와 포인트 컷을 정의하여 기능별로 깔끔하게 분리하도록 할 수 있다.
- Aspects 모듈은 AspectJ와의 통합을 제공한다.
- Instrumentation(인스트루멘테이션) 모듈은 Instrumentation을 지원하는 클래스와 특정 애플리케이션 서버에서 사용되는 클래스 로더(class loader) 구현체를 제공한다.

(5) 테스트

- 테스트(Test) 모듈은 JUnit 또는 TestNG를 사용하여 스프링 컴포넌트의 테스트를 지원한다.

4. 스프링 프레임워크 jar

스프링 프레임워크 5.x에 구성된 jar 파일이다. jar 파일에서 제공되는 기능은 다음과 같다.

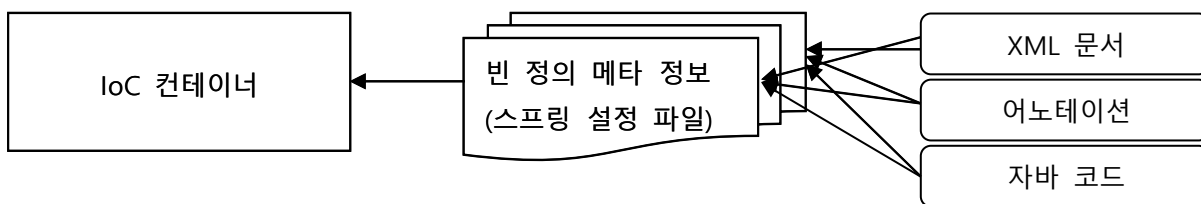
(1) 내부 모듈

jar 파일명	설명
spring-aop-*.jar	AOP Alliance에 호환되는 AOP 구현 기능을 제공한다.
spring-aspects-*.jar	AspectJ와 통합 기능을 제공한다.
spring-beans-*.jar	BeanFactory 인터페이스를 통한 구현기능을 제공한다.
spring-context-*.jar	코어와 빈 모듈을 확장해서 이벤트 처리, 리소스 로딩, 서블릿 컨테이너를 위한 컨텍스트 등의 추가 기능을 제공하고, applicationContext 인터페이스를 통해 구현한다.
spring-context-support-*.jar	spring-context 모듈 확장, 메일, 스케줄링 기능을 제공한다.
spring-core-*.jar	DI 기능의 프레임워크의 기반을 제공한다.
spring-expression-*.jar	스프링 표현 언어(SpEL) 지원 클래스가 제공된다.
spring-instrument-*.jar	Instrument 지원 클래스가 제공된다.
spring-instrument-tomcat-*.jar	톰캣 서버의 Instrument 지원 클래스를 제공한다.
spring-jdbc-*.jar	JDBC 지원 기능을 제공한다.
spring-jms-*.jar	JMS(Java Message Service)의 메시지 생성과 수신 기능을 제공한다.
spring-orm-*.jar	Hibernate, iBatis 등 ORM API를 위한 통합 레이어를 제공한다.
spring-oxm-*.jar	객체와 XML 매핑을 지원하는 추상 레이어를 제공한다.
spring-test-*.jar	JUnit 등의 스프링 컴포넌트의 단위 테스트를 지원한다.
spring-tx-*.jar	스프링의 트랜잭션 관리 기능을 제공한다.
spring-web-*.jar	파일업로드 등 웹 통합 기능과 원격자원 기능 등 웹 관련 기능을 제공한다.
spring-webmvc-*.jar	스프링 MVC 프레임워크 기능을 제공한다.
spring-webmvc-portlet-*.jar	포틀릿 환경에서 사용되는 스프링 MVC 구현 기능을 제공한다.

5. IoC(Inversion of Contral)

IoC(Inversion of Contral)란 “역 제어 또는 제어의 역전”으로 해석되며, 객체의 생명주기를 관리하고 의존성 주입(Dependency Injection)을 통해 각 계층이나 서비스들 간의 의존성을 맞춰두는 스프링에서 가장 핵심되는 기능이다.

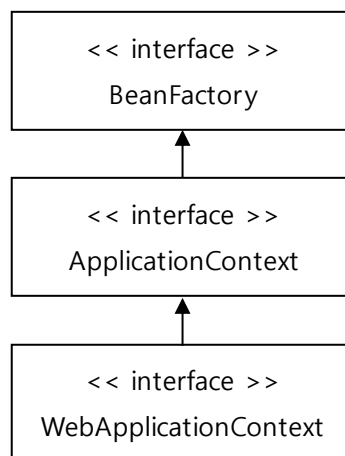
스프링 프레임워크에서는 객체의 생성, 소멸, 의존성에 관한 생명주기를 컨테이너가 관리한다. 개발자는 객체 생성에 신경 쓰지 않고 코드 작성만 하면 된다. 그러나 컨테이너가 어떻게 알 수 있을까? 그것은 xml 파일의 형태나 어노테이션, 자바 프로퍼티 파일 형태의 설정 파일에 빈을 정의하기 때문에 가능하다. 스프링에서 설정 파일은 DI 방식을 사용하며, 빈 정의(Been Definition) 정보는 클래스 사이의 의존 관계를 IoC 컨테이너가 자동으로 설정한다. 스프링에서 애플리케이션의 중요 부분을 형성하고 스프링 IoC 컨테이너에 의해 관리되는 객체는 빈(bean) 정의 메타 정보로 참조된다. 빈들과 각각의 의존성은 IoC 컨테이너에 의해 사용되는 설정 메타 정보로 반영된다.



자바는 클래스가 부모 클래스 또는 다른 클래스로부터 상속받는 관계이다. 스프링에서는 컨테이너에 의해 빈이 생성될 때 의존성 주입이 정반대이기 때문에 제어의 역전이라고 명명하였다.

5.1 IoC 컨테이너

스프링 프레임워크의 IoC 컨테이너를 위한 기본 패키지는 org.springframework.beans와 org.springframework.context이다. 스프링 IoC 컨테이너의 역할을 수행하는 BeanFactory 인터페이스와 ApplicationContext 인터페이스, WebApplicationContext 인터페이스가 있다.



① BeanFactory

BeanFactory 인터페이스는 빈 객체를 관리하고, 빈 객체간의 의존 관계 설정 기능을 제공하는 가장 기본적인 컨테이너이다. Resource 구현 클래스이다.

클래스	설명
org.springframework.core.io.FileSystemResource	파일 시스템의 특정 파일에서 정보를 읽음
org.springframework.core.io.InputStreamResource	입력 스트림으로부터 정보를 읽음

org.springframework.core.io.ClassPathResource	클래스 패스에 있는 자원에서 정보를 읽음
org.springframework.core.io.UrlResource	특정 URL로부터 정보를 읽음
org.springframework.core.io.ServletContextResource	웹 애플리케이션의 루트 경로를 기준으로 지정 한 경로의 자원에서 정보를 읽음

외부 자원의 설정 정보를 읽어 빈 객체를 생성하는 org.springframework.beans.factory.xml.XmlBeanFactory 클래스와 인터페이스를 사용하여 다양한 종류의 자원을 동일한 방식으로 표현하는

org.springframework.core.io.Resource 인터페이스가 제공된다. **특정 자원의 설정 정보로 XmlBeanFactory 객체를 생성한 후 getBean() 메서드로 빈을 가져와서 사용한다.**

Resource 구현 클래스를 이용한 BeanFactory 객체 생성의 예는 다음과 같다.

<pre>Resource re = new FileSystemResource("xml 설정 파일명"); XmlBeanFactory f = new XmlBeanFactory(re); 클래스명 dao = (클래스명)f.getBean("xml 설정 파일의 빈_id");</pre>
--

② ApplicationContext

ApplicationContext 인터페이스는 BeanFactory 인터페이스의 하위 인터페이스로 BeanFactory 기능, AOP, 메시지 자원 핸들링, 이벤트 위임, XML 스키마 확장 설정 등 웹 애플리케이션의 전사적 중심의 기능을 제공한다.

· org.springframework.context.support.ClassPathXmlApplicationContext
구현 클래스는 클래스 패스에 위치한 XML 파일로부터 설정 정보를 로딩한다.
· org.springframework.context.support.FileSystemXmlApplicationContext
구현 클래스는 파일 시스템에 위치한 XML 파일로부터 설정 정보를 로딩한다.
· org.springframework.context.support.XmlWebApplicationContext
구현 클래스는 웹 애플리케이션에 위치한 XML 파일로부터 설정 정보를 로딩한다.

③ WebApplicationContext

웹 애플리케이션을 위한 ApplicationContext로 하나의 웹 애플리케이션 마다 한 개 이상의 WebApplicationContext를 가질 수 있다. 구현 클래스는 ApplicationContext 인터페이스와 동일하다.

5.2 설정 메타 데이터

설정 메타 데이터는 스프링 IoC 컨테이너에 “인스턴스화, 설정, 그리고 애플리케이션내의 객체 조합”하는 방법이며, 가장 공통적으로 XML 형식을 사용한다. **XML 기반의 메타 데이터는 스프링 2.5에서 도입한 어노테이션 기반의 설정과 스프링 3.0에서 도입한 자바 기반의 설정 방법이 있다.** 설정 메타 데이터는 빈으로 정의하며, 가장 상위레벨 요소로 <beans>와 내부에 <bean>으로 설정된다. 빈으로 설정하는 것은 애플리케이션에 관한 서비스 레이어 객체, 데이터 접근 객체(DAO) 등이 있다. 다음은 XML 기반의 기본적인 설정 데이터의 구조이다.

servlet-context.xml

<pre><?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:beans="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context"</pre>
--

```

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<bean class= "..." id="..."><!-- 추가 빈의 객체나 설정 --></bean>
...
</beans>

```

5.3 빈 정의

빈이란 스프링 IoC 컨테이너가 관리하는 객체를 말하며, 애플리케이션의 객체이다. 빈은 IoC 컨테이너에 게 제공한 빈 정의 설정 메타 정보에 의해서 생성된다. 빈 정의란 객체를 생성하는 방법이며, XML 기반의 설정 메타 정보는 <bean> 요소로 id와 class의 필수 속성으로 정의한다. id는 빈을 구분하기 위한 문자열이며, class는 빈의 클래스명을 기술한다. 메타 데이터는 다음과 같은 정보를 포함한다.

- 패키지명을 포함한 클래스명 : 정의된 빈의 실제 구현 클래스
- 빈의 설정 요소 : 빈의 동작 상태와 참조에 관한 정보
- 생성된 객체에 관한 기타 설정 값.

빈은 id, class, name, scope, constructor-arg, property, autowire 등의 속성으로 정의한다.

```

<beans>
  <bean id="service" class= "com.service.Service" />
  <bean id="listController" class= "com.controller.ListController" />
</beans>

```

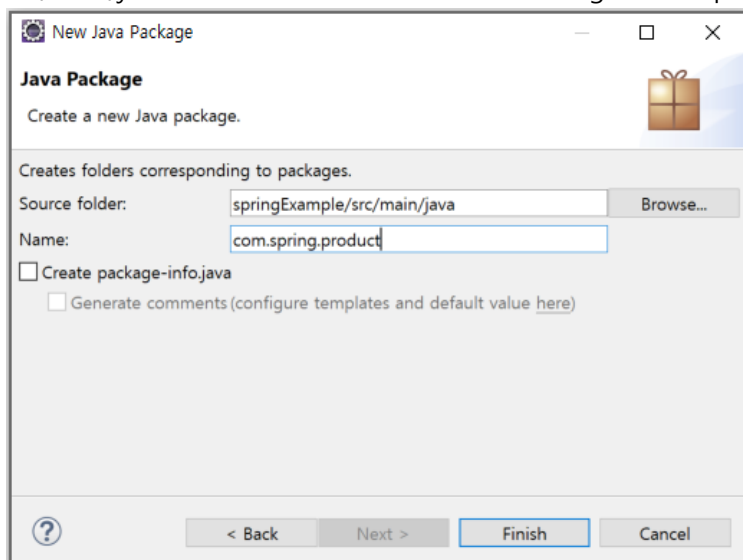
5.4 결합도(Coupling)가 높은 프로그램

결합도란 하나의 클래스가 다른 클래스와 얼마나 많이 연결되어 있는지를 나타내는 표현이며, 결합도가 높은 프로그램은 유지보수가 어렵다. 이 결합도와 유지보수 관계를 이해하기 위해서 간단한 실습을 한다.

Project Name : springExam

패키지 : com.spring.example

src/main/java -> 오른쪽 클릭 -> New -> Package : com.spring.product



SamsungTV 클래스 생성

```
package com.spring.product;

public class SamsungTV {
    public void powerOn() {
        System.out.println("SamsungTV---전원을 켜다.");
    }
    public void powerOff() {
        System.out.println("SamsungTV---전원을 끈다.");
    }
    public void volumeUp() {
        System.out.println("SamsungTV---소리를 올린다.");
    }
    public void volumeDown() {
        System.out.println("SamsungTV---소리를 내린다.");
    }
}
```

SamsungTV 클래스에는 TV 시청에 필요한 필수 기능인 네 개의 메소드가 있다. 그리고 SamsungTV와 같은 기능의 LgTV 클래스도 작성한다.

```
package com.spring.product;

public class LgTV {
    public void turnOn() {
        System.out.println("LgTV---전원 켜다.");
    }
    public void turnOff() {
        System.out.println("LgTV---전원 끈다.");
    }
    public void soundUp() {
        System.out.println("LgTV---소리를 올린다.");
    }
    public void soundDown() {
        System.out.println("LgTV---소리를 내린다.");
    }
}
```

LgTV 클래스에도 SamsungTV 클래스와 같은 기능을 수행하는 메서드가 있지만, SamsungTV의 메소드 이름과 다르다. 이제 이 두 TV 클래스를 번갈아 사용하는 TVUser 프로그램을 구현한다.

src/test/java

```
package com.spring.product;

public class TVUser {
```

```

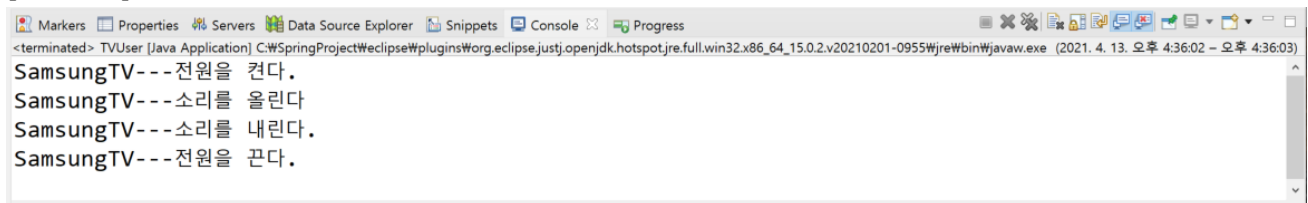
public static void main(String[] args) {
    SamsungTV tv = new SamsungTV();
    tv.powerOn();
    tv.volumeUp();
    tv.volumeDown();
    tv.powerOff();
}
}

```

SamsungTV 객체를 생성하여 메서드를 호출했으므로 프로그램의 실행결과는 다음과 같다.

Run As -> Java Application

[실행결과]



```

<terminated> TVUser [Java Application] C:\SpringProject\weclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (2021. 4. 13. 오후 4:36:02 - 오후 4:36:03)
SamsungTV---전원을 켜다.
SamsungTV---소리를 올린다
SamsungTV---소리를 내린다.
SamsungTV---전원을 끈다.

```

이제 SamsungTV를 시청하는 TVUser 프로그램을 LgTV를 시청하는 프로그램으로 수정한다.

```

package com.spring.product;

public class TVUser {
    public static void main(String[] args) {
        LgTV tv = new LgTV();
        tv.turnOn();
        tv.soundUp();
        tv.soundDown();
        tv.turnOff();
    }
}

```

SamsungTV와 LgTV는 시그니처(signature)가 다르므로 TVUser 코드 대부분을 수정해야 TV를 교체할 수 있다. 현재 상태에서는 두 TV 클래스가 같은 메서드를 가지게끔 강제할 어떤 수단도 없다. 만약 TVUser와 같은 클라이언트 프로그램이 하나가 아니라 여러 개라면 유지보수는 더욱더 힘들 것이며, TV 교체를 결정하기가 쉽지 않을 것이다.

5.5 다형성(polymorphism) 이용하기

결합도를 낮추기 위해서 다양한 방법을 사용할 수 있겠지만, 가장 쉽게 생각할 수 있는 것이 객체 지향 프로그래밍의 핵심 개념인 다형성을 이용하는 것이다. 앞에서 작성한 프로그램을 다형성을 이용하여 수정한다. 다형성을 이용하려면 구현과 메서드 재정의 그리고 형변환이 필요하다.

TV 클래스들을 구현하여 사용하기 위해 **TV 인터페이스**를 추가하고, 모든 TV가 **공통으로 가져야 할 메서드들을 추상 메서드로 선언**한다.

```

package com.spring.product;

public interface TV {

```

```

    public void powerOn();
    public void powerOff();
    public void volumeUp();
    public void volumeDown();
}

```

이제 SamsungUTV와 LgUTV 클래스를 만들어 위에서 작성한 TV 인터페이스를 구현하도록 한다.

```

package com.spring.product;

public class SamsungUTV implements TV {
    public SamsungUTV() {
        System.out.println("SamsungUTV 객체 생성");
    }
    @Override
    public void powerOn() {
        System.out.println("SamsungUTV---전원을 켜다.");
    }
    @Override
    public void powerOff() {
        System.out.println("SamsungUTV---전원을 끈다.");
    }
    @Override
    public void volumeUp() {
        System.out.println("SamsungUTV---소리를 올린다.");
    }
    @Override
    public void volumeDown() {
        System.out.println("SamsungUTV---소리를 내린다.");
    }
}

```

이렇게 하면 SamsungUTV와 LgUTV 클래스는 TV 인터페이스에 선언된 추상 메서드들을 모두 재정의 해야한다. 공통의 메소드명 사용.

```

package com.spring.product;

public class LgUTV implements TV {
    public LgUTV() {
        System.out.println("LgUTV 객체 생성");
    }
    @Override
    public void powerOn() {
        System.out.println("LgUTV---전원 켜다.");
    }
}

```



```

    }
    @Override
    public void powerOff() {
        System.out.println("LgUTV---전원 끈다.");
    }
    @Override
    public void volumeUp() {
        System.out.println("LgUTV---소리를 올린다.");
    }
    @Override
    public void volumeDown() {
        System.out.println("LgUTV---소리를 내린다.");
    }
}

```

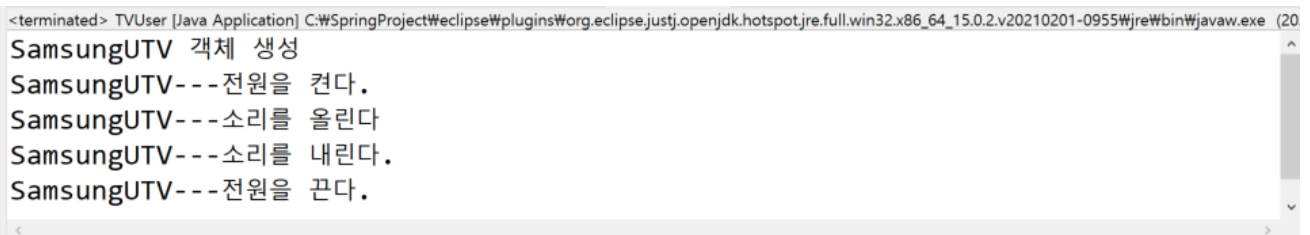
인터페이스를 이용하여 모든 TV 클래스가 같은 메서드들을 가질 수밖에 없도록 강제할 수 있게 되었다. 이제 이 두 TV를 이용하는 TVUser 클래스를 다음과 같이 수정한다.

```

package com.spring.product;

public class TVUser {
    public static void main(String[] args) {
        // LgTV tv = new LgTV();
        TV tv = new SamsungUTV(); // TV tv = new LgUTV();
        tv.powerOn();
        tv.volumeUp();
        tv.volumeDown();
        tv.powerOff();
    }
}

```



```

<terminated> TVUser [Java Application] C:\SpringProject\workspace\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (20
SamsungUTV 객체 생성
SamsungUTV---전원을 켜다.
SamsungUTV---소리를 올린다.
SamsungUTV---소리를 내린다.
SamsungUTV---전원을 끈다.

```

TVUser 클래스는 TV 인터페이스 타입의 참조변수 tv에 SamsungUTV 클래스의 참조값을 대입했다. 이렇게 묵시적 형변환을 이용하여 객체를 참조하면 SamsungUTV를 LgUTV 객체로 변경할 때, 참조하는 객체만 변경하면 되므로 객체를 쉽게 교체할 수 있다. 이렇게 다형성을 이용하면 TVUser와 같은 클라이언트 프로그램이 여러 개 있더라도 최소한의 수정으로 TV를 교체할 수 있다. 따라서 유지보수가 용이해졌다고 할 수 있다.

5.6 디자인 패턴 이용하기

결합도를 낮추기 위한 또 다른 방법으로 디자인 패턴을 이용하는 방법이 있다. 앞에서 살펴본 다형성을 이용하는 방법은 메서드를 호출할 때 인터페이스를 이용함으로써 좀 더 쉽게 TV를 교체할 수 있었다. 하지만 이 방법 역시 TV를 변경하고자 할 때 TV 클래스 객체를 생성하는 소스를 수정해야만 한다.

TV를 교체할 때, 클라이언트 소스를 수정하지 않고 TV를 교체할 수만 있다면 유지보수는 더욱 편리해질 것이다. 이를 위해서 Factory 패턴을 적용해야 하는데, Factory 패턴은 클라이언트에서 사용할 객체 생성을 캡슐화하여 TVUser와 TV 사이를 느슨한 결합 상태로 만들어준다.

다음과 같이 Factory 패턴이 적용된 BeanFactory 클래스를 추가한다.

```
package com.spring.product;

public class BeanFactory {

    public Object getBean(String beanName){
        if(beanName.equals("samsung")){
            return new SamsungUTV();
        } else if(beanName.equals("lg")){
            return new LgUTV();
        }
        return null;
    }
}
```

BeanFactory 클래스의 getBean() 메서드는 매개변수로 받은 beanName에 해당하는 객체를 생성하여 리턴한다. 이제 이 BeanFactory 클래스를 이용하여 사용할 TV객체를 획득하도록 TVUser클래스를 수정한다.

```
package com.spring.product;

import java.util.Scanner;

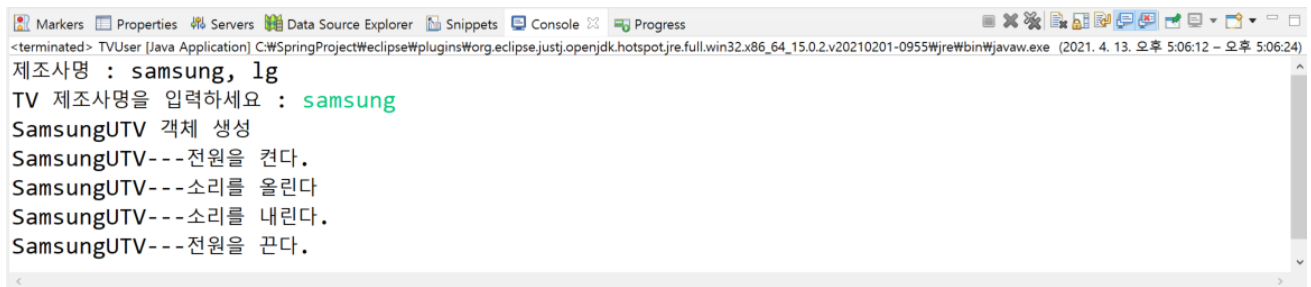
public class TVUser {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);

        System.out.println("제조사명 : samsung, lg");
        System.out.print("TV 제조사명을 입력하세요 : ");
        String comName = scan.nextLine();

        BeanFactory factory = new BeanFactory();
        TV tv = (TV)factory.getBean(comName);
        tv.powerOn();
        tv.volumeUp();
        tv.volumeDown();
        tv.powerOff();
    }
}
```

Run As -> Java Application



```
<terminated> TVUser [Java Application] C:\SpringProject\weclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (2021. 4. 13. 오후 5:06:12 ~ 오후 5:06:24)
제조사명 : samsung, lg
TV 제조사명을 입력하세요 : samsung
SamsungUTV 객체 생성
SamsungUTV---전원을 켜다.
SamsungUTV---소리를 올린다
SamsungUTV---소리를 내린다.
SamsungUTV---전원을 끈다.
```

실행되는 TV를 변경하고 싶을 때는 명령행 매개변수만 수정하여 실행한다. 결국 클라이언트 소스를 수정하지 않고도 실행되는 객체를 변경할 수 있다. 이런 결과를 얻을 수 있었던 것은 TV객체를 생성하여 리턴하는 BeanFactory 때문이다. 클라이언트에 해당하는 TVUser는 자신이 필요한 객체를 직접 생성하지 않는다. 만약 그랬다면 TV가 변경될 때마다 소스를 수정해야 했을 것이다. TVUser는 단지 객체가 필요하다는 것을 BeanFactory에 요청했을 뿐이고, BeanFactory가 클라이언트가 사용할 TV 객체를 적절하게 생성하여 넘겨준 것이다.

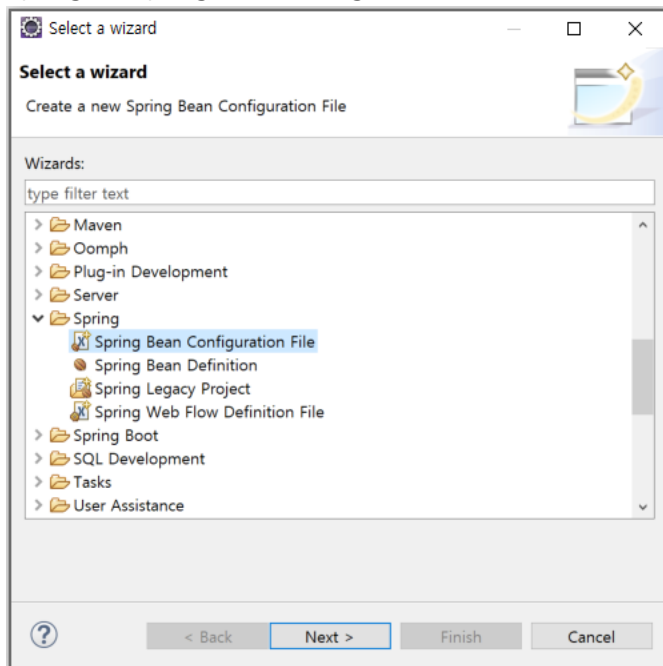
6. 스프링 컨테이너 및 설정파일

대부분 IoC 컨테이너는 각 컨테이너에서 관리할 객체들을 위한 별도의 설정파일이 있다. Servlet 컨테이너는 web.xml파일에 해당 컨테이너가 생성하고 관리할 클래스들을 등록한다. 스프링 프레임워크도 다른 컨테이너와 마찬가지로 자신이 관리할 클래스들이 등록된 XML 설정 파일이 필요하다.

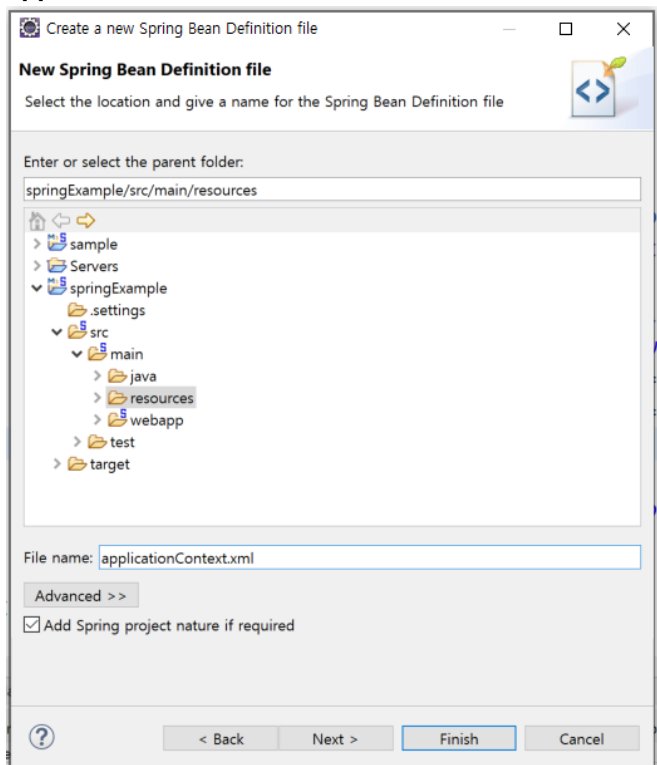
6.1 스프링 IoC 시작하기

src/main/resources -> 오른쪽 클릭 -> New -> Other

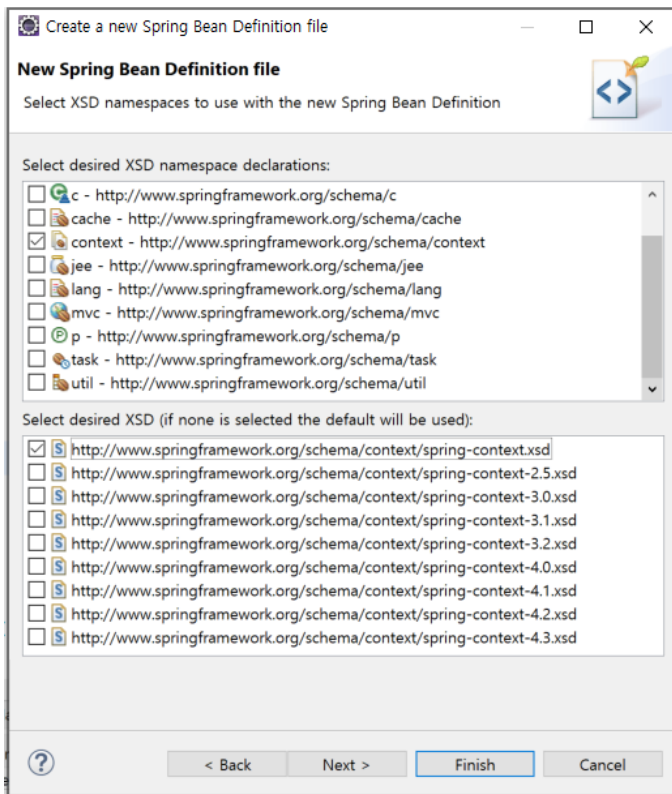
Spring -> Spring Bean Configuration File 선택 Next 클릭



applicationContext.xml 파일을 생성한다.



Next 클릭



Finish 클릭

이때 기본으로 <beans> 루트 엘리먼트와 네임스페이스 관련 설정들이 정의되어야 한다. 앞에서 작성한 TV 예제를 스프링 기반으로 테스트하기 위해서 SamsungTV 클래스를 스프링 설정 파일에 등록한다. 이때 <bean> 엘리먼트를 사용하는데 클래스당 하나의 <bean> 설정이 필요하다.

applicationContext.xml

```
<?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-4.3.xsd">

    <bean id="tv" class="com.spring.product.SamsungUTV" />

</beans>
```

<bean> 엘리먼트에서 가장 중요한 것은 class 속성값이다. 여기에 패키지 경로가 포함된 전체 클래스 명으로 지정해야 한다.

6.2 스프링 컨테이너 구동 및 테스트(src/test/java/)

```
package com.spring.product;

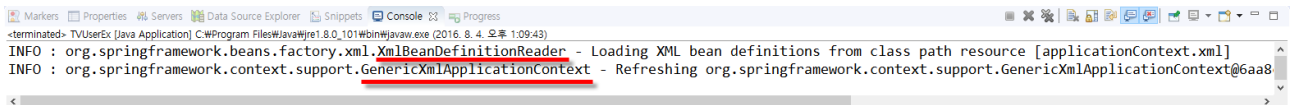
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;
```

```

public class TVUserEx {
    public static void main(String[] args) {
        // 1. Spring 컨테이너를 구동한다.
        AbstractApplicationContext factory =
            new GenericXmlApplicationContext("applicationContext.xml");
    }
}

```

우선 클래스 경로에 있는 applicationContext.xml 파일을 로딩한다는 메시지가 가장 먼저 출력되며, 다음으로 GenericXmlApplicationContext 객체가 생성되어 스프링 컨테이너가 구동됐다는 메시지가 출력된다.



```

INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from class path resource [applicationContext.xml]
INFO : org.springframework.context.support.GenericXmlApplicationContext - Refreshing org.springframework.context.support.GenericXmlApplicationContext@6aa8

```

구동된 컨테이너로부터 SamsungUTV 객체를 생성한다.

스프링 컨테이너를 구동하고 이름이 tv인 객체를 getBean() 메서드를 이용하여 요청하도록 TVUser를 수정한다.

```

package com.spring.product;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class TVUserEx {
    public static void main(String[] args) {
        // 1. Spring 컨테이너를 구동한다.
        AbstractApplicationContext factory =
            new GenericXmlApplicationContext("applicationContext.xml");

        // 2. Spring 컨테이너로부터 필요한 객체를 요청(Lookup)한다.
        TV tv = (TV)factory.getBean("tv");
        tv.powerOn();
        tv.volumeUp();
        tv.volumeDown();
        tv.powerOff();

        // 3. Spring 컨테이너를 종료한다.
        factory.close();
    }
}

```

작성된 프로그램을 실행하면 다음과 같은 메시지가 출력된다.

```

<terminated> TVUserEx [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (2016. 8. 4. 오후 1:11:57)
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from class path resource [applicationContext.xml]
INFO : org.springframework.context.support.GenericXmlApplicationContext - Refreshing org.springframework.context.support.GenericXmlApplicationContext@6aa8ce
SamsungTV 객체 생성
SamsungTV---전원을 켜다.
SamsungTV---소리를 올린다.
SamsungTV---소리를 내린다.
SamsungTV---전원을 끈다.
INFO : org.springframework.context.support.GenericXmlApplicationContext - Closing org.springframework.context.support.GenericXmlApplicationContext@6aa8ce

```

- ① TVUserEx 클라이언트가 스프링 설정 파일을 로딩하여 컨테이너 구동
- ② 스프링 설정 파일에 <bean> 등록된 SamsungTV 객체 생성
- ③ getBean() 메서드로 이름이 'tv'인 객체를 요청
- ④ SamsungTV 객체 반환.

중요한 것은 실행되는 TV를 LgTV로 변경할 때, applicationContext.xml 파일만 수정하면 된다는 점이다. 즉 TVUser 클라이언트 소스를 수정하지 않고도 동작하는 TV를 변경할 수 있으며, 기존에 BeanFactory 클래스를 사용했던 것보다 유지보수가 좀 더 편해졌다 할 수 있다.

6.3 스프링 컨테이너의 종류

스프링에서는 BeanFactory와 이를 상속한 ApplicationContext 두 가지 유형의 컨테이너를 제공한다. 먼저 BeanFactory는 스프링 설정 파일에 등록된 <bean> 객체를 생성하고 관리하는 가장 기본적인 컨테이너 기능만 제공한다. 그리고 컨테이너가 구동될 때 <bean> 객체를 생성하는 것이 아니라, 클라이언트의 요청에 의해서만 <bean> 객체가 생성되는 지연 로딩 방식을 사용한다. 따라서 일반적인 스프링 프로젝트 BeanFactory를 사용할 일은 전혀 없다.

반면 ApplicationContext는 BeanFactory가 제공하는 <bean> 객체 관리 기능 외에도 트랜잭션 관리나 메시지 기반의 다국어 처리 등 다양한 기능을 지원한다. 또한 컨테이너가 구동되는 시점에 <bean> 등록된 클래스들을 객체 생성하는 즉시 로딩식으로 동작한다. 그리고 웹 애플리케이션 개발도 지원하므로 대부분 스프링 프로젝트는 ApplicationContext 유형의 컨테이너를 이용한다.

ApplicationContext의 구현 클래스는 매우 다양하다. 하지만 그 클래스들을 모두 살펴볼 수도 없거니와 의미도 없다. 실제로 가장 많이 사용하는 두 개의 클래스만 알고 있으면 된다.

구현 클래스	설명
GenericXmlApplicationContext	파일 시스템이나 클래스 경로에 XML 설정 파일을 로딩하여 구동하는 컨테이너이다.
XmlWebApplicationContext	웹 기반의 스프링 애플리케이션을 개발할 때 사용하는 컨테이너이다.

GenericXmlApplicationContext은 실습에서 사용한 컨테이너로 TVUser 클라이언트에서 직접 객체를 생성하여 구동한 컨테이너다. 하지만 XmlWebApplicationContext이 어떻게 구동되고 사용되는지 SpringMVC 패턴에서 다시 확인해 보겠다.

6.4 스프링 XML 설정

- ① <beans> 루트 엘리먼트

스프링 컨테이너는 <bean> 저장소에 해당하는 XML 설정 파일을 참조하여 <bean>의 생명주기를 관리하고 여러 가지 서비스를 제공한다. 따라서 스프링 프로젝트 전체에서 가장 중요한 역할을 담당하며, 이 설정파일을 정확하게 작성하고 관리하는 것이 무엇보다 중요하다. 스프링 설정 파일 이름은 무엇을 사용하든 상관없지만 <beans>를 루트 엘리먼트로 사용해야 한다. <beans> 엘리먼트 시작 태그에 네임스페이스를 비롯한 XML 스키마 관련 정보가 설정된다. beans 네임스페이스가 기본 네임스페이스로 선언되어 있

으며, spring-beans.xsd 스키마 문서가 schemaLocation 등록되어 있다. 따라서 <bean>, <description>, <alias>, <import> 등 네 개의 엘리먼트를 자식 엘리먼트로 사용할 수 있다. 이 중에서 <bean>, <import> 정도가 프로젝트 개발 시 사용하게 된다.

② <import> 엘리먼트

스프링 설정 파일 하나에 우리가 만든 모든 클래스를 <bean>으로 등록하고 관리할 수도 있다. 하지만 스프링 기반의 애플리케이션은 단순한 <bean> 등록 외에도 트랜잭션 관리, 예외처리, 다국어 처리 등 복잡하고 다양한 설정이 필요하다. 이런 모든 설정을 하나의 파일로 모두 처리할 수도 있지만, 그렇게 하면 스프링 설정 파일이 너무 길어지고 관리도 어렵다. 결국, 기능별 여러 XML 파일로 나누어 설정하는 것이 더 효율적인데, 이렇게 분리하여 작성한 설정 파일들을 하나로 통합할 때 <import> 엘리먼트를 사용한다.

context-datasource.xml	context-transaction.xml
<pre><beans> DataSource 관련 설정 </beans></pre>	<pre><beans> Transaction 관련 설정 </beans></pre>
<pre>applicationContext.xml <beans> <import resource="context-datasource.xml" /> <import resource="context-transaction.xml" /> </beans></pre>	

<import> 태그를 이용하여 여러 스프링 설정 파일을 포함함으로써 한 파일에 작성하는 것과 같은 효과를 낼 수 있다.

③ <bean> 엘리먼트

스프링 설정 파일에 등록하려면 <bean> 엘리먼트를 사용한다. 이때 id와 class 속성을 사용하는데, id 속성은 생략할 수 있지만 class 속성은 필수이다. class 속성에 클래스를 등록할 때는 정확한 패키지 경로와 클래스 이름을 지정해야 한다. class 속성만 사용해도 객체가 생성되는지 확인하기 위해서 기존에 작성한 스프링 설정 파일을 수정한다.

applicationContext.xml
<pre><?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.xsd"> <bean class="com.spring.product.SamsungUTV" /> </beans></pre>

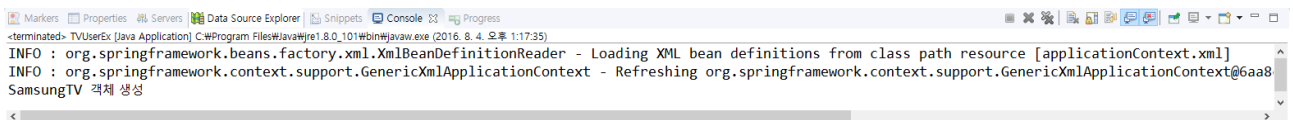
이렇게 class 속성만 이용하여 bean 등록을 했으면, TVUser 프로그램을 컨테이너만 구동하도록 수정한 후 실행한다.

```
package com.spring.product;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class TVUserEx {
    public static void main(String[] args) {
        // 1. Spring 컨테이너를 구동한다.
        AbstractApplicationContext factory =
            new GenericXmlApplicationContext("applicationContext.xml");
    }
}
```

TVUserEx 프로그램의 실행 결과를 확인해보면 SamsungTV 객체가 생성되는 데는 아무 문제가 없음을 알 수 있다.



하지만 TVUser 클라이언트가 SamsungTV 객체를 요청하려면 이름이 반드시 지정되어야 한다. 이렇게 <bean> 객체를 위한 이름을 지정할 때 사용하는 속성이 id이다. id 속성은 컨테이너로부터 <bean> 객체를 요청할 때 사용하므로 반드시 스프링 컨테이너가 생성한 개체들 사이에서 유일해야 한다. 그래야 컨테이너가 각 객체를 식별할 수 있다.

그리고 id 속성값에 해당하는 문자열은 자바의 식별자 작성 규칙을 따르며, 일반적으로 CamelCase을 사용한다. 만약 숫자로 시작하거나 공백 문자 및 특수 기호를 id로 지정하면 식별자 작성 규칙에 따라 예외가 발생한다.

id와 같은 기능을 하는 속성으로 name도 있다. name 속성은 id와 다르게 자바 식별자 작성 규칙을 따르지 않는 문자열도 허용한다. 따라서 특수기호가 포함된 아이디를 <bean> 아이디고 지정할 때는 id 대신 name 속성을 쓴다. 물론 name 속성값 역시 전체 스프링 파일 내에서 유일해야 한다. 사실 id나 name 속성 중 어떤 것을 사용하든 상관은 없다.

④ <bean> 엘리먼트 속성

· init-method 속성

Servlet 컨테이너는 web.xml 파일에 등록된 Servlet 클래스의 객체를 생성할 때 디폴트 생성자만 인식한다. 따라서 생성자로 Servlet 객체의 멤버변수를 초기화할 수 없다. 그래서 서블릿은 init() 메서드를 재정의하여 멤버변수를 초기화 한다.

스프링 컨테이너 역시 스프링 설정 파일에 등록된 클래스를 객체 생성할 때 디폴트 생성자를 호출한다. 따라서 객체를 생성한 후에 멤버변수 초기화 작업이 필요하다면, Servlet의 init() 같은 메소드가 필요하다. 이를 위해 스프링에서는 <bean> 엘리먼트 init-method 속성을 지원한다.

applicationContext.xml

```
<?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-4.3.xsd">

<!-- <bean id="tv" class="com.spring.product.SamsungUTV" /> -->
<bean id="tv" class="com.spring.product.SamsungUTV" init-method="initMethod" />

</beans>

```

스프링 컨테이너는 <bean> 등록된 클래스 객체를 생성한 후에 init-method 속성으로 지정된 init-method() 메서드를 호출한다. 이 메서드에서 멤버변수에 대한 초기화 작업을 처리한다.

```

package com.spring.product;

public class SamsungUTV implements TV {
    public SamsungTV() {
        System.out.println("SamsungTV 객체 생성");
    }
    public void initMethod() {
        System.out.println("객체 초기화 작업 처리.");
    }
    public void powerOn() {
        System.out.println("SamsungTV---전원을 켜다.");
    }
    ...
}

```

```

INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from class path resource [applicationContext.xml]
INFO : org.springframework.context.support.GenericXmlApplicationContext - Refreshing org.springframework.context.support.GenericXmlApplicationContext@2f4907f
SamsungUTV 객체 생성
객체 초기화 작업 처리.
SamsungUTV---전원을 켜다.
SamsungUTV---소리를 올린다.
SamsungUTV---소리를 내린다.
SamsungUTV---전원을 끈다.
INFO : org.springframework.context.support.GenericXmlApplicationContext - Closing org.springframework.context.support.GenericXmlApplicationContext@2f4907f

```

· destroy-method 속성

init-method와 <bean> 엘리먼트에서 destroy-method 속성을 이용하여 스프링 컨테이너가 객체를 삭제하기 직전에 호출될 임의의 메서드를 지정할 수 있다.

```

<bean id="tv" class=" com.spring.product.SamsungUTV" init-method="initMethod" destroy-
method="destroyMethod" />

```

```

package com.spring.product;

public class SamsungTV implements TV {
    public SamsungTV() {
        System.out.println("==> SamsungTV(1) 객체 생성");
    }
    public void initMethod() {
        System.out.println("객체 초기화 작업 처리.");
    }
    public void destroyMethod() {
        System.out.println("객체 삭제 전에 처리할 로직 처리...");
    }
    ...
}

```

다음은 initMethod와 destroy-method 속성을 사용한 SamsungTV 객체의 실행 결과다.

init-method속성으로 지정된 initMethod() 메서드는 컨테이너가 구동되어 SamsungTV 객체가 생성된 직후에 호출된다. 그리고 컨테이너가 종료되기 직전에 컨테이너는 자신이 관리하는 모든 객체를 삭제하는데, 이때 destroy-method 속성으로 지정한 destroyMethod() 메서드는 SamsungTV 객체가 삭제되기 직전에 호출된다.

· lazy-init 속성

ApplicationContext를 이용하여 컨테이너를 구동하면 컨테이너가 구동되는 시점에 스프링 설정 파일에 등록된 <bean>들을 생성하는 즉시 로딩방식으로 동작한다. 그런데 어떤 <bean>은 자주 사용되지 않으면서 메모리를 많이 차지하여 시스템에 부담을 주는 경우도 있다.

따라서 스프링에서는 컨테이너가 구동되는 시점이 아닌 해당 <bean> 사용되는 시점에 객체를 생성하도록 lazy-init 속성을 제공한다. 특정 <bean>을 등록할 때 **lazy-init = "true"**로 설정하면 스프링 컨테이너는 해당 <bean>을 미리 생성하지 않고 **클라이언트가 요청하는 시점에 생성한다**. 결국 메모리 관리를 더 효율적으로 할 수 있게 된다.

```
<bean id="tv" class="com.spring.product.SamsungUTV" lazy-init="true" />
```

· scope 속성

프로그램을 개발하다 보면 개발자도 모르는 사이에 수많은 객체가 생성된다. 그런데 이 중에는 하나만 생성되도 상관없는 객체들이 있다. 예를 들어 우리가 사용중인 SamsungTV 클라이언트에는 price 같은 필드가 있어서 생성되는 객체들이 다른 가격을 가지는 것도 아니므로 SamsungTV 클래스는 하나의 객체만 생성되도록 한다.

```
package com.spring.product;
```

```
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class TVUserEx {
    public static void main(String[] args) {
        TV tv1 = new SamsungUTV();
        TV tv2 = new SamsungUTV();
        TV tv3 = new SamsungUTV();
    }
}
```

그러면 SamsungTV 클래스로부터 하나의 객체만 생성하여 유지하려면 어떻게 해야할까? 우선 가장 쉬운 방법은 다음처럼 객체를 생성하고 참조값을 복사하여 재사용하는 것이다.

```
TV tv1 = new SamsungUTV();
TV tv2 = tv1;
TV tv3 = tv2;
```

결과적으로 tv1, tv2, tv3은 같은 주소를 가지므로 하나의 객체를 공유하게 된다. 하지만 이렇게 프로그램을 명시하는 것은 쉬운 것은 아니다. 결국은 자연스럽게 하나의 객체만 생성하도록 제어해야 하는데, 이때 사용하는 것이 '**싱글톤 패턴**'이다. 그러나 싱글톤 패턴을 구현하려면 일일이 클래스에 패턴 관련 코드를 작성해야 하므로 귀찮은 일이다. 결국 클래스로부터 객체를 생성하는 쪽에서 자동으로 싱글톤 객체로 생성하는 것이 가장 바람직하며, 스프링에서는 바로 이런 기능을 컨테이너가 제공한다. 스프링 컨테이너는 컨테이너가 생성한 <bean>을 어느 범위에서 사용할 수 있는지를 지정할 수 있는데, 이때 scope 속성을 사용한다. **scope 속성값은 기본이 싱글톤**이다. 이는 해당 <bean>이 스프링 컨테이너에 의해 단 하나만 생성되어 운용되도록 한다.

```
<bean id="tv" class="com.spring.product.SamsungTV" scope="singleton" />
```

TVUser 클래스에서 SamsungTV 객체를 여러 번 요청하고 결과를 확인해 본다.

TVUserEx.java

```
package com.spring.product;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class TVUserEx {
    public static void main(String[] args) {
        // 1. Spring 컨테이너를 구동한다.
        AbstractApplicationContext factory =
            new GenericXmlApplicationContext("applicationContext.xml");

        // 2. Spring 컨테이너로부터 필요한 객체를 요청(Lookup)한다.
        TV tv1 = (TV)factory.getBean("tv");
```

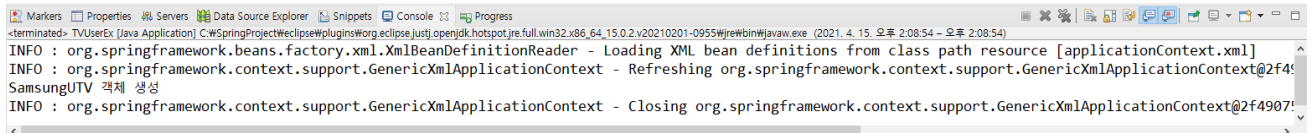
```

TV tv2 = (TV)factory.getBean("tv");
TV tv3 = (TV)factory.getBean("tv");

// 3. Spring 컨테이너를 종료한다.
factory.close();
}
}

```

SamsungTV 클래스의 scope의 속성값을 "singleton"으로 설정하고 클라이언트에서 세 번 요청하면 실행 결과에서 알 수 있듯이 SamsungTV 객체는 메모리에 하나만 생성되어 유지된다.



```

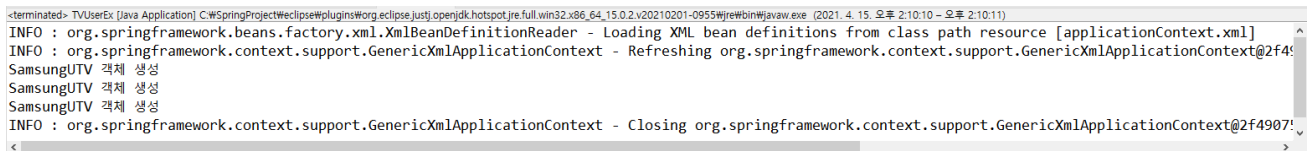
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from class path resource [applicationContext.xml]
INFO : org.springframework.context.support.GenericXmlApplicationContext - Refreshing org.springframework.context.support.GenericXmlApplicationContext@2f4907!
SamsungUTV 객체 생성
INFO : org.springframework.context.support.GenericXmlApplicationContext - Closing org.springframework.context.support.GenericXmlApplicationContext@2f4907!

```

스프링을 이용하는 애플리케이션에 따라 다르겠지만 스프링 컨테이너가 관리하는 <bean>들은 대부분 싱글톤으로 운영되어야 한다. 따라서 scope의 속성값을 "singleton"으로 설정하거나 아예 생략하는 경우가 일반적이다.

<bean>의 scope의 속성값을 "prototype"으로 지정할 수 있는데, 이렇게 지정하면 스프링 컨테이너는 해당 <bean>이 요청될 때마다 매번 새로운 객체를 생성하여 반환한다.

```
<bean id="tv" class="com.spring.product.SamsungUTV" scope="prototype" />
```



```

INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from class path resource [applicationContext.xml]
INFO : org.springframework.context.support.GenericXmlApplicationContext - Refreshing org.springframework.context.support.GenericXmlApplicationContext@2f4907!
SamsungUTV 객체 생성
SamsungUTV 객체 생성
SamsungUTV 객체 생성
INFO : org.springframework.context.support.GenericXmlApplicationContext - Closing org.springframework.context.support.GenericXmlApplicationContext@2f4907!

```