전자정부 표준프레임워크 퍼스트북

eGovFrame First Book

대한민국 공공정보시스템의 핵심기술
자바 오픈소스 기반 표준기술

임철홍 지음

플랫폼기술연구소

전자정부 표준프레임워크 퍼스트북

임철홍 지음

(주)플랫폼기술연구소

책을 펴내며

전자정부 표준프레임워크가 만들어지고 활용된지 10년이 되어가고 있고, 많은 프로젝트에 활용되고 있다. 그동안 개발자 및 대학생을 대상으로 표준프레임워크 교육을 진행해온 결과 자바 개발 경험이 부족한 교육생에게 교육을 하기에 많은 어려움을 느꼈다. SW프레임워크는 개발패턴이 발전되어 Best Practice를 쉽게 활용할 수 있도록 만들어진 반제품이고, 대규모 프로젝트에도 적용이 가능하도록 패턴이 만들어진 까닭에 초보 개발자의 경우 왜 이렇게 활용을 해야 하고 무엇이 좋은지를 전혀 인식하지 못하고 매우 복잡하고 어려운 것으로 인식하고 기계적으로 코딩하여 적용하는 것이 일반적이었다. 이 책은 초보자가 쉽게 표준프레임워크에 입문할 수 있도록 일반적인 자바개발과 표준프레임워크를 활용했을 때를 비교하고 있고, 예제도 쉽게 이해할 수 있도록 단순화 하여 제공하고 있다.

표준프레임워크 및 스프링 프레임워크 등 관련 기술을 이해하는데 더욱 많은 분들에게 도움이 되도록 하기 위해서 이 책을 오픈소스처럼 무료로 제공하기로 하였다. 표준프레임워크를 구성하고 있는 기술이 거의 외산기술로 되어 있어 빠른 시간내에 대한민국 SW 기술이 발전하여 스프링 프레임워크와 같은 우수한 SW를 개발하고, 더욱 많은 SW들이 우수성을 인정받아 표준프레임워크가 순수 국산 SW기술로 만들어지기를 소망해본다.

마지막으로 본 책이 출간 될 수 있도록 도와주신 저와 함께 표준프레임워크 관리 및 운영사업 참여 중이신 개발자분들과 한국정보화진흥원 및 행정안전부 관계자분들에게 감사드립니다.

■ 저자 – 임철홍 (imich@platformtech.co.kr)
(주)플랫폼기술연구소 대표 | 컴퓨터시스템응용 기술사
고려대학교 재료공학사, 전자컴퓨터 석사, 영상정보처리 박사

SK C&C, SK 커뮤니케이션즈 등에서 정보시스템 개발과 아키텍트 업무를 수행하였으며, '09년부터 현재까지 전자정부 표준프레임워크 개발사업 및 유지보수 사업에 참여하였다. 한국산업기술대학교, 고려사이버대학교, 한국SW기술진흥협회에서 강의를 하고 있다. 대한민국 SW발전에 도움이 되고자 노력하고 있다.

■ 소스코드 다운로드 - https://github.com/cholhongim/egovframe-firstbook
■ 표지디자인 – 임성한 (slim@olcreative.co.kr)

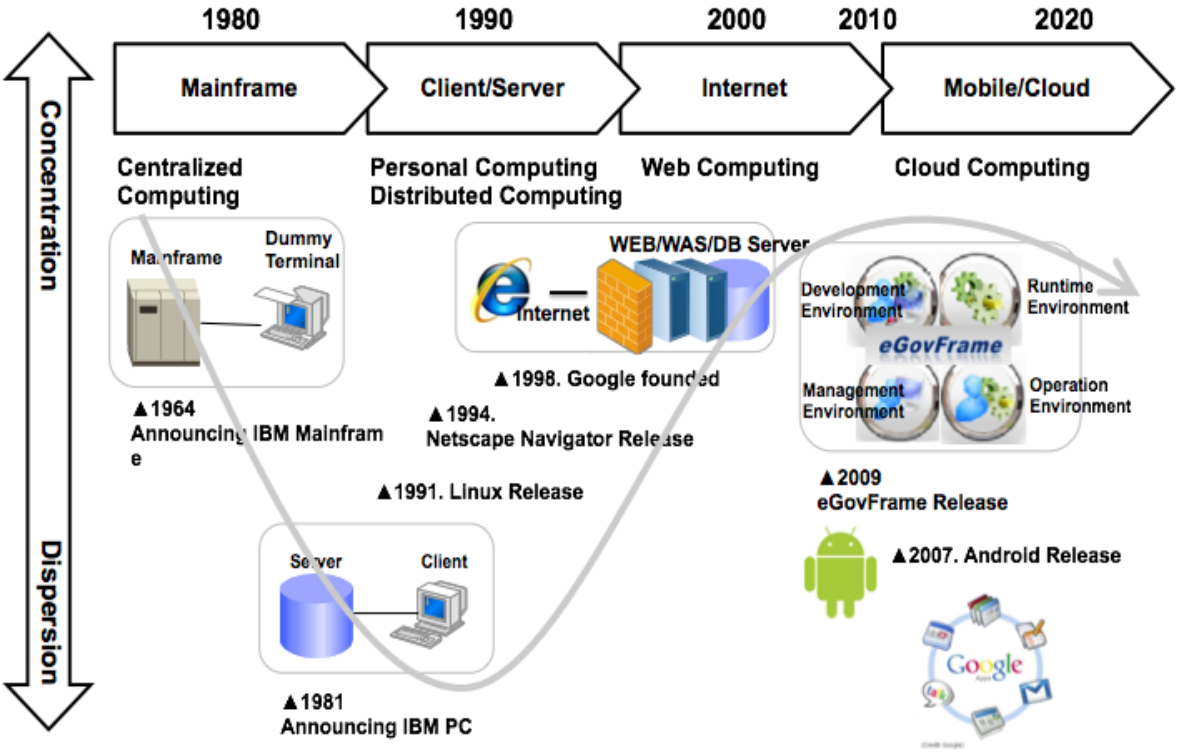
목차

1. 표준프레임워크 소개

"전자정부 표준프레임워크"는 공공사업에 적용되는 SW 프레임워크의 표준을 정립하고, 응용 SW 표준화, 품질 및 재사용성을 높일 수 있는 기반을 제공한다.

1.1 SW 프레임워크 등장배경

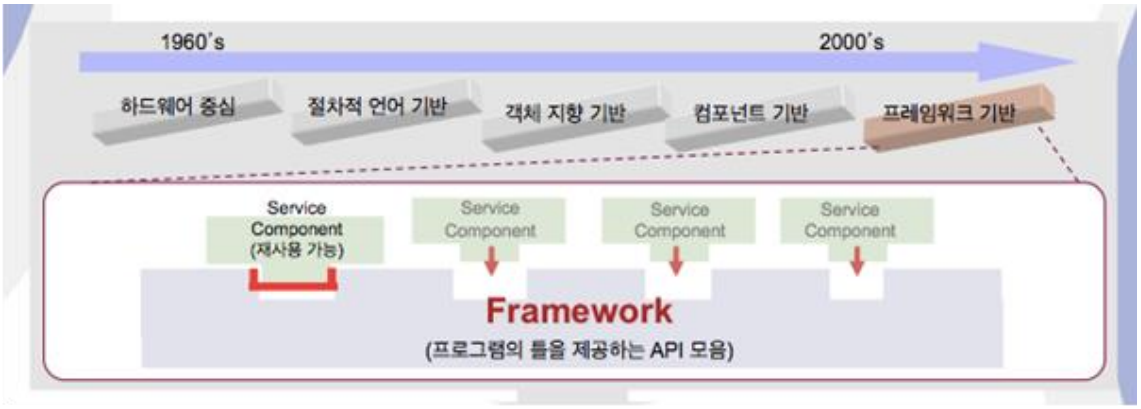
메인프레임 시대부터, Client/Server, 시대의 전환인 WEB의 등장 모바일/클라우드 시대까지 변화해오면서, 이제는 누구나 소프트웨어를 사용하고, 사용해야만 생활할 수 있는 시대가 되었다. 최근 들어서는 twitter, Facebook 등 소셜미디어의 등장으로 거대한 데이터들이 날마다 생산되고, 이러한 데이터를 분석하고 활용/처리하는데 대해 IT 업계의 대표적인 이슈로 자리매김하고 있다.



[그림 1-1] 소프트웨어 개발의 발전

이러한 변화 가운데에서도, 소프트웨어에 대한 사용자 요구사항 중 초기부터 현재에 이르기까지 가장 중요하게 생각되었던 부분은 "자동화", "재사용" 처리이다. 이러한 자동화는 기술이 발전함에 따라 실시간(real-time) 처리 부분과 일괄(Batch)처리로 구분하게 되었고, 오늘날과 같이 대규모의 데이터가 관리되는 시점에서는 대용량 데이터 처리와 같은 수 많은 분야들이 만들어지게 되었다. 소프트웨어는 서비스 사용자 입장에서도 많은 변화가 있어왔지만, 소프트웨어 서비스를 개발하고 제공하는 방법 면에서도, 위기 극복을 위해 끊임없이 진화·발전하며 많은 변화가 있었다. 특히, 소프트웨어 규모의 대형화

및 복잡화로 개발비용이 증대 되었고, 일관되지 않은 개발방식으로인해 유지보수성이 악화되었으며, 새로운 기술에 대한 교육과 훈련이 부족 하여 SW 개발 및 운영에 많은 문제점이 발생되게 되었다. 이러한 문제를 해결하기 위해 재사용 방식을 높일 수 있도록 다양한 형태의 기술들이 발전되었다.



[그림 1-2] 재사용 방식의 발전

1) 소스 재사용

초보적인 재사용 방식으로 과거에 유사한 문제를 코딩한 적이 있거나 비슷한 예제를 다른 코드에서 복사해서 사용하는 방법을 말한다.

2) 재사용 메소드

복사/붙이기 방식과 동일한 코드가 여러 클래스에서 나오는 것을 지양하기 위한 방법으로 C언어에서 하던 것처럼 자주 이용하는 기능을 라이브러리로 만들어 재사용하는 방식을 말한다.

3) 재사용 객체

기존의 재사용방식 - 소스 재사용, 재사용 메소드 - 들은 자바뿐만 아니라 다른 언어를 사용할 때도 가능한 방식이나, 확장이 쉽지 않은 방식이었다면, 자바 언어에서는 클래스를 설계하고 상속/확장하는 방식으로 재사용을 진행한다.

4) 디자인패턴

클래스의 재사용방식이 객체의 수직적인 재사용 방식이었다면 디자인 패턴은 상황적인 문제를 해결해주는 재사용 방식이다. 공통적인 로직 문제에 대한 일반화된 해결 방법으로 클래스의 재사용이 아니라 메커니즘의 재사용으로 진행되는 방식이다.

5) SW 프레임워크

디자인패턴은 상황에 대한 해결은 가능하였지만, 시스템 전체에 대해서는 건건이 직접

패턴을 적용하여 문제를 해결해야 되었다. 이러한 문제점을 해결하기 위해 전체 시스템적 관점에서 표준화된 디자인 패턴을 적용한 설계 및 구현체가 SW 프레임워크이다.

1.2 SW 프레임워크 정의

SW 프레임워크의 사전적 의미는 “무엇을 이루는 뼈대 혹은 기반 구조”를 말한다. 소프트웨어 프레임워크는 소프트웨어 개발을 위한 뼈대 구조로 개발을 도와주는 부분부터 사상에 이르기까지 다양한 범위로 논의 되고 있다. 이외에도 여러 SW 프레임워크 정의를 찾아볼 수 있다.

" 소프트웨어 시스템에 대한 라이브러리 또는 클래스의 재사용 세트"

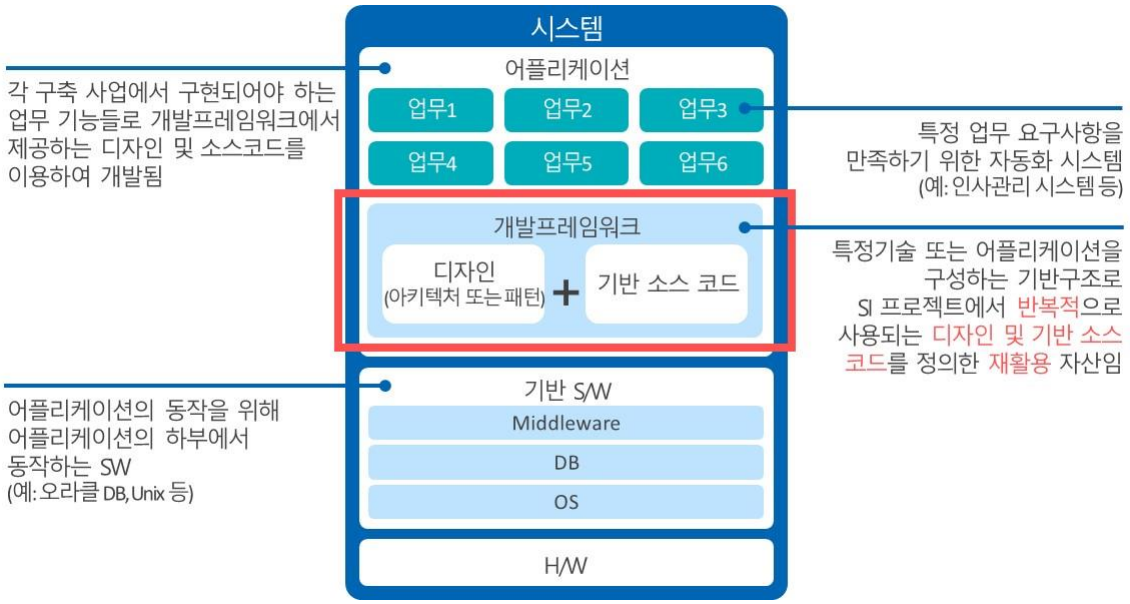
" 웹 애플리케이션 프레임워크 (WAF)는 동적 웹 사이트, 웹 애플리케이션, 웹 서비스 및 웹 자원의 개발을 지원하도록 설계된 소프트웨어 프레임워크이다. 이 프레임워크는 웹 개발 수행 일반적인 활동과 관련된 오버 헤드를 완화하는 것을 목적으로 한다. 예를 들어, 많은 프레임 워크는 데이터베이스 액세스, 템플릿 워크 및 세션 관리를 위한 라이브러리를 제공하고, 코드 재사용을 지원한다.”

“ 프레임워크는 소프트웨어의 특정 클래스에 대한 재사용 가능한 디자인을 구성하는 클래스를 협력의 집합 - GoF의 패턴 : 랄프 존슨"

소프트웨어 개발위기에서 특정한 소프트웨어 개발 영역에서 자주 발생하는 문제를 해결하기 위해 경험적으로 축적된 설계 패턴들로써, Best Practice의 모음인 디자인패턴은 해결을 위한 많은 도움을 주었다. 그러나, 디자인 패턴은 실체가 아닌 개념적인 방법만을 제시하고 있는 점에 그 한계가 있다. 반면에 프레임워크는 이러한 디자인 패턴을 실체로 구현한 일종의 솔루션이라 할 수 있다. 최근에는 여기에 다양한 개발 가이드와 개발 도구 등도 함께 제공하여 프레임워크로 볼 수 있다.

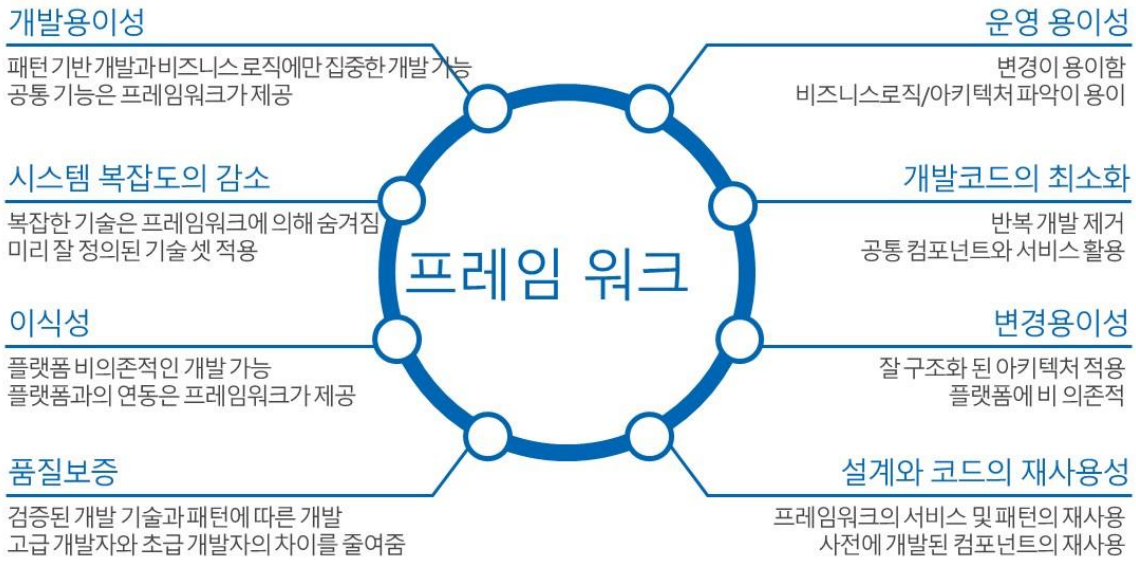
이러한 의미들을 종합적으로 고려하여 다음과 같이 SW 프레임워크를 정의하였다.

- 일련의 문제 해결을 위한 추상화된 디자인을 구현한 클래스들의 집합으로 클래스보다는 큰 규모의 재사용을 지원함
- 구체적이며 확장 가능한 기반 코드, 설계자가 의도하는 아키텍처와 디자인 패턴 집합
- 실전에서 얻은 최적화 개발 경험을 반영한 재사용 가능한 API 집합
- 라이브러리와 달리 애플리케이션의 틀과 구조를 결정, 그 위에 개발된 개발자의 코드를 제어하는 반제품 성격의 소프트웨어



[그림 1-3] SW 프레임워크의 정의

SW 프레임워크 기반 개발 방식을 통하여 얻을 수 있는 장점은 다음과 같다.



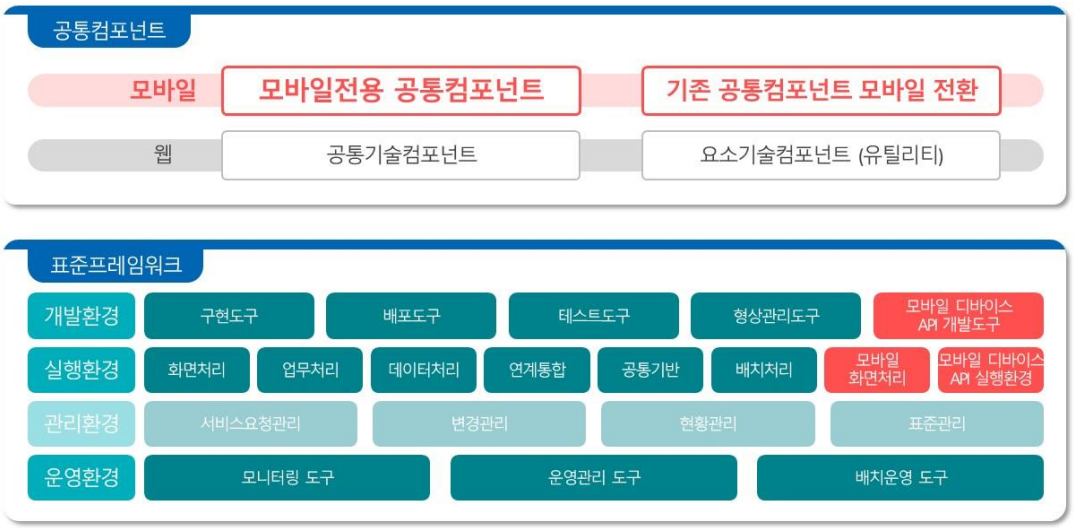
[그림 1-4] SW 프레임워크의 적용 효과

1.3 표준프레임워크 개요

국내 대표적인 공공정보화사업 추진체계인 전자정부지원사업 사례를 살펴보면, 표준프레임워크 개발 이전 대부분의 공공정보화 사업에서 대기업의 SW 프레임워크가 도입 활용되었다. 대기업 SW 프레임워크를 활용했을 때 발생되는 문제점으로는 폐쇄적인 정책으로 인해 자사 SW 프레임워크를 외부에 공개하지도 판매하지도 않기 때문에 상호

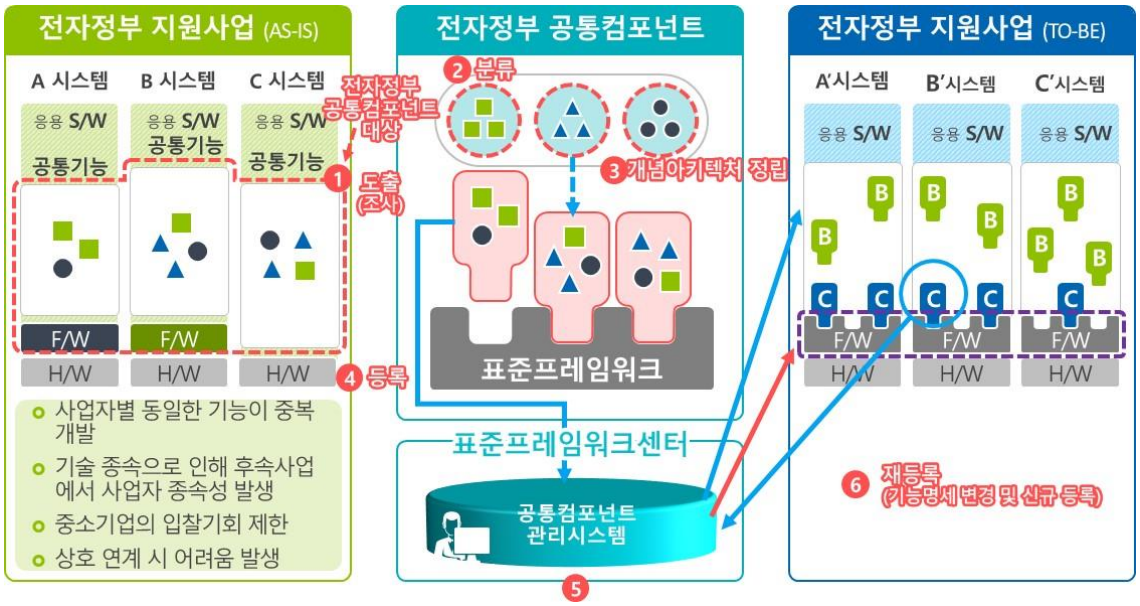
호환성을 보장하기가 매우 어렵다는 점이다. 즉, A사가 구축한 응용시스템에 대한 구조를 다른 B사가 파악하기 매우 어렵기 때문에 연속사업 혹은 유지보수사업의 사업자 변경을 원천적으로 어렵게 한다는 문제가 있었다. 또한 이미 특정 SW 프레임워크를 활용하여 구축된 소스코드를 다른 SW 프레임워크로 변경은 불가능하기 때문에 SW 프레임워크의 적용은 불가피하게 소스코드의 종속성 문제를 가지게 되었다. SW 프레임워크는 건축물의 철골과 같아서 SW 프레임워크를 교체한다는 것은 기존 건축물을 모두 허물고 철골부터 새로 만드는 것과 같다. 정보시스템이 특정 SW 프레임워크 사용으로 인해 사업자 및 기술 종속성을 가지는 문제점을 해결하기 위해 개방형 표준을 채택하여 공동으로 활용할 수 있는 SW 프레임워크를 필요로 하게 되었다.

“전자정부 표준프레임워크”는 공공사업에 적용되는 SW프레임워크의 표준을 정립하고, 응용 SW 표준화, 품질 및 재사용성을 높일 수 있는 기반을 제공한다. 표준프레임워크는 실행, 개발, 관리, 운영 등 4개의 환경과 모바일 표준프레임워크, 그리고 공통컴포넌트로 구성되며, 참조프레임워크로 대표적인 오픈소스 SW프레임워크인 스프링 프레임워크를 채택하였으며, 이외에도 다양한 OSS를 적용/활용하여 구성되었다. 표준프레임워크는 정보시스템 개발을 위해 필요한 기능 및 아키텍처를 미리 만들어 제공함으로써 효율적인 어플리케이션 구축을 지원한다.



[그림 1-5] 전자정부 표준프레임워크 구성

“전자정부 표준프레임워크”는 공공사업에 적용되는 개발프레임워크의 표준 정립으로 응용 SW 표준화, 품질 및 재 사용성 향상을 목표로 한다. 이를 통해“전자정부 서비스의 품질향상” 및 “정보화 투자 효율성 향상”을 달성하고, 대·중소기업이 동일한 개발기반 위에서 공정 경쟁이 가능하게 된다. 표준프레임워크는 기존 다양한 플랫폼(.NET, php 등) 환경을 대체하기 위한 표준은 아니며, java 기반의 정보시스템 구축에 활용하실 수 있는 개발·운영 표준 환경을 제공하기 위한 것이다.



[그림 1-6] 전자정부 표준프레임워크 목적

전자정부 표준프레임워크는 사전에 다양한 요구사항을 수렴하고 이를 토대로 개발되었다. 특정업체 및 기술에 종속되는 문제를 없애기 위해 범용적으로 많이 사용되는 오픈소스를 최대한 활용하였으며, 소스코드뿐만 아니라 개발문서(산출물)도 함께 공개함으로써 사용 편의성을 높이고자 하였다. 전자정부 표준프레임워크 구축을 위해 진행되었던 내용은 다음과 같다.

- 사전 ISP 과정을 거쳐 최적의 기능 도출
- 전세계적으로 활용되고 있는 오픈소스를 최대한 활용 → 특정업체 기술종속성 배제
- 상용 솔루션이 존재하는 기능은 가능한 제외
- 공공 정보화 사업에 많이 활용되는 인프라 환경과의 호환성 보장
- 광범위한 보급·확산을 위해 소스 뿐만 아니라 분석·설계 산출물도 함께 공개
- 다수가 만족하는 표준이 될 수 있도록, 개발 과정에서 다양한 전문가로 구성된 협의회 의견을 청취·반영

전자정부 표준프레임워크는 자바기반의 정보시스템 개발과 운영 시에 필요한 기본기능들을 표준화하며 미리 구현해 둔 것으로 개발자는 이를 활용하여 업무 기능을 구현한 후 조립함으로써 전체 시스템을 완성할 수 있다.

[그림 1-7] 전자정부 표준프레임워크 활용

전자정부 표준프레임워크는 공공사업에 적용되는 SW프레임워크의 표준을 정립하여 SW의 품질 및 재사용성 향상을 목표로 한다. 이를 통해 “전자정부서비스의 품질향상” 및 “정보화 투자 효율성 향상”을 달성하고 대 중 소기업이 동일한 개발기반 위에서 공정경쟁이 가능하게 되었다.

1.4 표준프레임워크 활용 고려 사항

효과적인 사용을 위해서는 아래의 조건을 충족해야 한다.

① 자바 기반의 웹 응용 시스템(WAS가 존재하는 경우)
② (2.7 기준) JavaEE(J2EE) 5 혹은 JDK1.5 이상의 환경
(단,개발환경 2.7 이상에서는 JDK 1.6이상 필요1))
(3.0 ~ 3.1 기준) JavaEE(J2EE) 6 혹은 JDK1.6 이상의 환경
(3.5 ~ 기준) JavaEE(J2EE) 7 혹은 JDK1.7 환경
③ 신규 개발시스템으로써, 기존 시스템과 물리적 혹은 논리적으로 구분되는 경우
☞ 실행환경 내 모바일표준프레임워크의 사용자 경험(UX) 지원 기능은 프레임워크와 개발언어 종류에 상관없이 활용가능 (javascript 기반)

표준프레임워크는 표준으로써 목적을 만족하기 위하여 아래의 규칙을 준수하여 적용하여야 한다.

- 실행환경은 원칙적으로 변경 없이 활용해야 함
- 개발환경은 기능의 변경과 추가에 제약사항 없음
- 공통컴포넌트는 변경 가능하나, 표준프레임워크 아키텍처 준수 필요

2. 개발환경 구성하기

표준프레임워크 개발환경은 프로그램 개발, 테스트, 빌드, 형상관리 등 전체 개발 생명 주기를 지원한다.

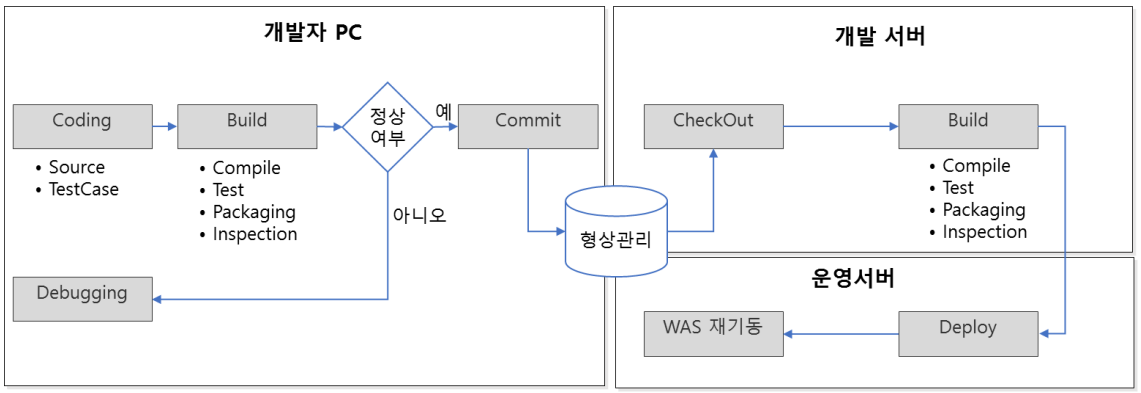
2.1 개발환경 구성

개발환경은 구현도구, 테스트도구, 형상·변경관리 도구, 배포도구로 구성된다.

[표 2-1] 개발환경 도구 설명

구분	설명
구현도구	프로그램 개발, 컴파일, 디버그 등 개발자 개별 환경 제공
테스트도구	개발자가 단위테스트를 수행하고 결과를 리포팅
형상·변경관리 도구	개발자가 작성한 소스코드를 통합하여 버전관리 수행
배포도구	형상관리의 소스코드를 통합 빌드하고 개발 서버 등에 배포

개발환경은 개발자 PC에서 작업이 진행되는 개발자 개발환경과 개발서버 및 운영서버에서 진행되는 서버개발환경으로 구성된다. 개발자는 개발자 개발환경을 활용하여 소스코드를 제작, 컴파일 및 테스트를 수행하고 실행 가능한 형태로 빌드한다. 개발자 개인별 소스코드는 형상관리 시스템을 통하여 단일 저장소에 통합되어 저장된다. 개발자는 소스코드를 수정하여 형상관리 시스템에 반영(commit)하거나, 다른 개발자의 소스코드를 자신의 소스코드로 통합(update)하는 형태로 공동작업 형태로 개발한다. 배포도구는 형상관리 시스템에 통합 저장된 소스코드를 불러와서(checkout) 통합 빌드한다. 그리고, 웹 어플리케이션 서버 등에서 실행이 가능한 형태로 구조화(packaging)하여 배포한다.



[그림 2-1] 개발자개발환경과 서버개발환경

표준프레임워크 개발환경에서 도구 별로 채택 활용하고 있는 대표적인 오픈소스는 다음과 같다. 개발환경을 구성하는 오픈소스의 라이선스는 대부분 Apache 2.0 기반으로 구성되어 있고, 일부 LGPL, EPL이 활용되었다.

[표 2-2] 개발환경 주요 오픈소스

구분	설명
구현도구	Eclipse, PMD, Maven(빌드)
테스트도구	JUnit, EasyMock, DbUnit, EclEmma
형상·변경관리 도구	Subversion, jTrac
배포도구	Hudson, Nexus, Maven

Eclipse는 자바소스코드를 편집, 컴파일, 테스트, 실행이 가능하도록 지원하는 대표적인 개발도구 이다. PMD는 Eclipse 플러그인으로 되어 있으며, 룰셋 기반으로 소스코드 오류 등을 검사하여 품질을 높이도록 한다. Maven은 소스코드 컴파일에 필요한 라이브러리를 관리하고 컴파일, 테스트, 빌드, 테스트 등 작업을 수행한다. JUnit은 단위테스트 코드를 작성하여 테스트하도록 지원한다. EasyMock은 웹 어플리케이션을 실행하지 않고도 테스트가 가능하도록 시뮬레이션 환경을 지원하며, DBUnit은 DB에 테스트 데이터를 생성하고 결과가 미리 입력된 결과값과 같은지 비교 할 수 있도록 환경을 지원한다. EclEmma는 실제 소스코드에서 단위테스트로 수행된 부분이 어느 정도인지 퍼센트 형태로 정보를 알려주는 coverage 테스트를 수행한다. Subversion은 형상관리 도구로 소스코드를 통합하고 버전관리 한다. jTrac은 변경관리 이력 등을 추적할 수 있도록 한다. Hudson은 지속적인 테스트와 통합을 지원하는 CI(Continuous Integration) 도구이다. Nexus는 Maven 라이브러리를 3rd Party로 제공할 때 활용한다.

2.2 개발환경 설치 및 구성

개발환경 구성을 위해서는 세가지 방법이 있다. 가장 쉬운 방법으로 표준프레임워크 홈페이지에서 제공하는 교육교재를 활용하는 방법이다. 필요한 소프트웨어들이 모두 구성되어 있고 압축을 풀기만 하면 바로 활용이 가능하다. 또 한가지 방법은 표준프레임워크에서 개발환경을 내려 받고 Tomcat, JDK 등 필요한 설정을 한다. 위의 두가지 방법은 모두 윈도우에서만 활용 가능하다. 마지막 방법은 기존에 활용하던 Eclipse에 표준프레임워크 플러그인을 설치하여 활용하는 방법으로 윈도우뿐만 아니라, 맥과 리눅스 환경에서도 적용 가능하다.

2.2.1 교육실습교재 사용(윈도우 환경)

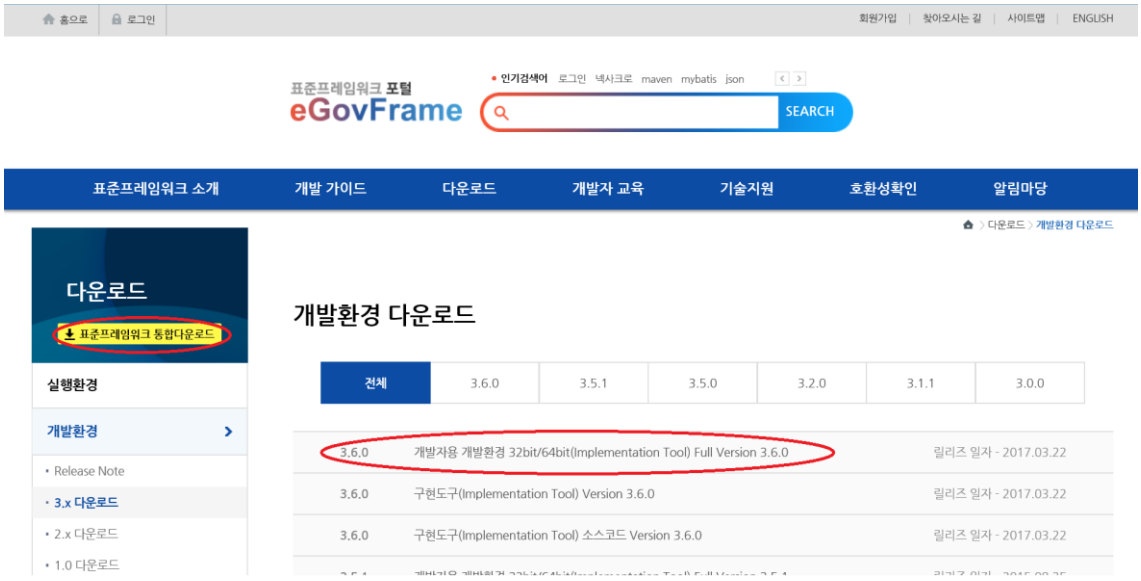
개발환경을 가장 쉽게 활용할 수 있는 방법은 개발자 교육에서 이용하고 있는 교육 실습교재를 그대로 이용하는 것이다. eGovFrame 홈페이지에서 “개발자교육>교육자료”에서 다운받을 수 있다. C디렉터리 메인(C:\\)에 압축을 풀고 eclipse를 실행하면 바로 개발환경이 실행된다. 표준프레임워크를 처음 사용하시거나 자바에 익숙하지 않는 개발자들은 처음에는 이방법을 활용하여 사용하는 것을 권장한다. 필요한 소프트웨어 및 설정이 되어 있어 바로 활용이 가능하다. bin 디렉터리 밑에 Tomcat, JDK, MySQL이 설치되어 있는 것을 확인할 수 있다. 표준프레임워크 버전에 따라 JDK, Tomcat 등의 버전은 아래와 상이할 수 있다.

[표 2-3] 표준프레임워크 교육실습교재 디렉터리 구조

디렉터리	설명
bin	실행파일
apache-tomcat-7.0.59	Apache Tomcat
apache-tomcat-8.0.24	
jdk1.7.0_80	JDK_HOME
jdk1.8.0_45	
eclipse	개발환경이 포함된 Eclipse JEE Luna SR2 (4.4.2)
maven/repository	Local Maven Repository
mysql-5.6.21	실습용 DB
textbook	교재파일
workspace.edu	Eclipse Workspace

2.2.2 개발환경 구성(윈도우 환경)

eGovFrame 홈페이지로 접속하여 "다운로드>개발환경"으로 이동한다. "표준프레임워크 통합 다운로드"를 활용하거나, 개발환경>3.x 다운로드를 선택하여 "개발자용 개발환경 32bit/64bit Full Version 3.6"을 다운로드 한다. 사용자 PC에 따라 32bit, 64bit 중에 선택한다. (3.6 이외 버전을 다운로드 하여도 무방하다)



[그림 2-2] 개발자개발환경과 서버개발환경

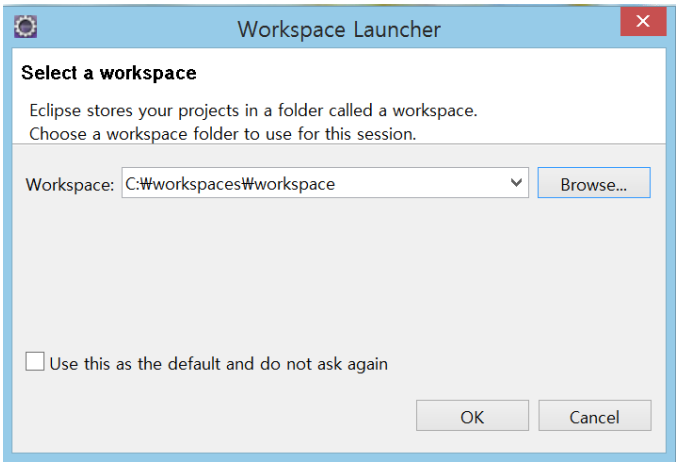
첨부된 파일을 다운로드 받아 특정 디렉터리에 압축을 해제한다. 만약 JDK가 설치되어 있지 않다면, Oracle 사이트에서 JDK1.7 이상 버전을 다운로드 받아 설치한다. JDK를 설치할 때 32bit와 64bit중에 사용자 PC에 맞도록 설치해야 한다. eclipse 디렉터리안에 eclipse.ini에 설치된 JDK 정보(-vm JDK설치경로\bin\javaw.exe)를 다음과 같이 추가한다. 추가하지 않을

경우 Fail to create Virtual Machine에러가 발생될 수 있고, 프로젝트에 따라서 JRE가 아니라 JDK환경에서 정상 구동되는 경우도 있기 때문이다. JDK를 ***Program Files***에 설치를 하게 되면 빈 공백 때문에 디렉터리를 정상적으로 인식하지 못하는 문제가 발생될 수 있어, 별도 디렉터리를 지정하여 설치하는 것이 좋다. 필자는 app 디렉터리를 생성하여 설치하였다.

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.300.v20150602-1417
-product
org.eclipse.epp.package.jee.product
--launcher.defaultAction
openFile
--launcher.XXMaxPermSize
256M
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vm C:\app\Java\jdk1.7.0_80\bin\javaw.exe
-vmargs
-Dfile.encoding=UTF-8
-Dosgi.requiredJavaVersion=1.7
-Xms40m
-Xmx512m
```

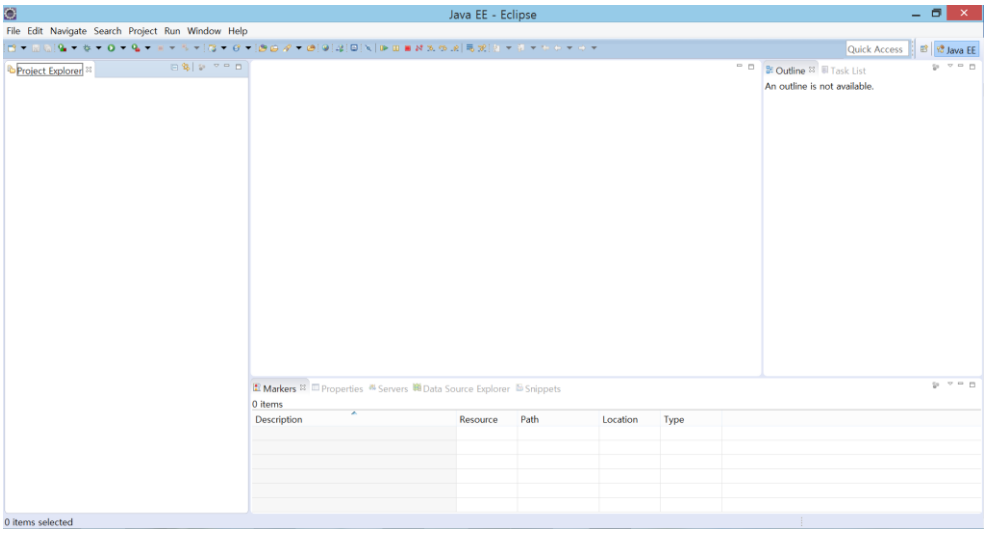
[그림 2-3] eclipse.ini 수정

eclipse.exe를 눌러서 실행을 하면 표준프레임워크 로그가 보이고, workspace를 선택하는 화면이 나타나게 된다. 디폴트로 선택된 디렉터리를 활용하여도 되고, 다른 디렉터리를 지정해도 된다.



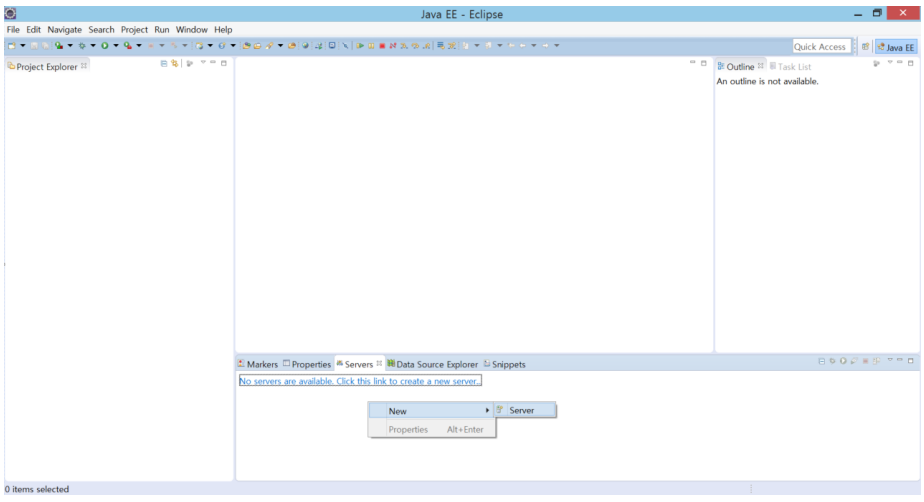
[그림 2-4] Workspace 선택

OK를 눌러서 선택하면 Eclipse가 실행되어 화면에 보여지게 된다.

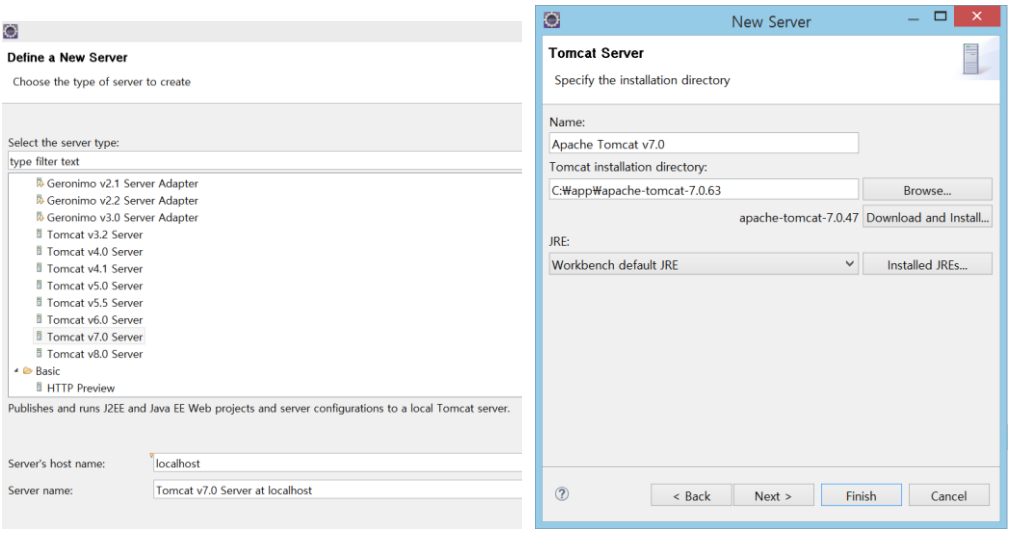


[그림 2-5] 실행된 화면

그 다음으로는 Apache Tomcat 설치가 필요하다. Tomcat은 Eclipse에서 웹 어플리케이션을 개발할 때 개발자 개인 PC에서 테스트 할 때 주로 활용된다. http://tomcat.apache.org 에서 Tomcat7 또는 8을 다운받아 설치한다. 윈도우 환경이라면 32-bit Windows zip이나 64-bit Windows zip을 받아 압축을 푸는 것이 편리하다. 서비스로 인스톨하게 되면 윈도우 시작될 때 자동으로 시작이 되어 Eclipse에서 Tomcat이 구동 될때 포트 사용 중으로 에러가 발생 될 수 있다. 설치된 Tomcat을 다음과 같이 Eclipse에 설정한다. 먼저 아래 View 영역에서 Servers를 선택하고 “Click this link to create a new server”을 선택하거나 마우스 오른쪽 버튼을 눌러 New>Server를 선택한다.



[그림 2-6] 새로운 서버 등록하기

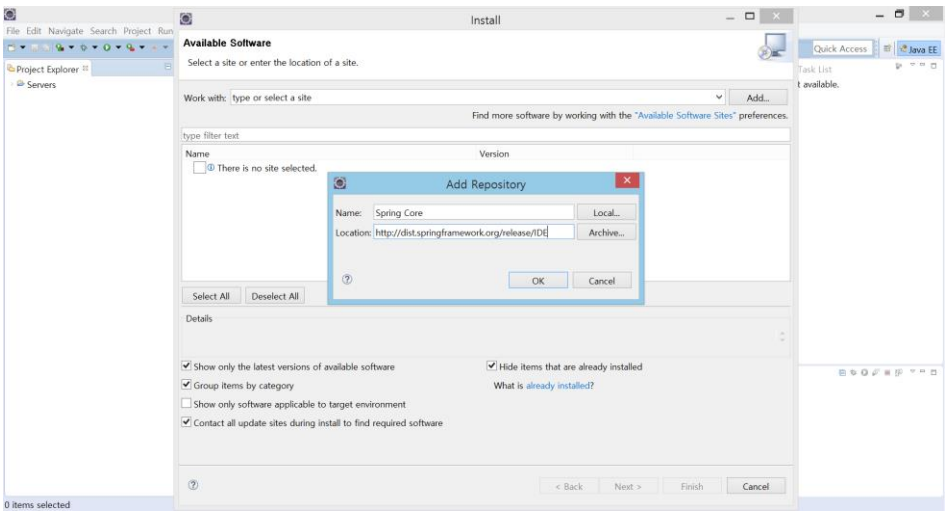


[그림 2-7] Tomcat 등록하기

다운로드 받은 Tomcat 버전을 선택하고, Next를 누른후에 Tomcat이 설치된 디렉터리를 선택하고 Finish를 누르면 설정이 끝난다.

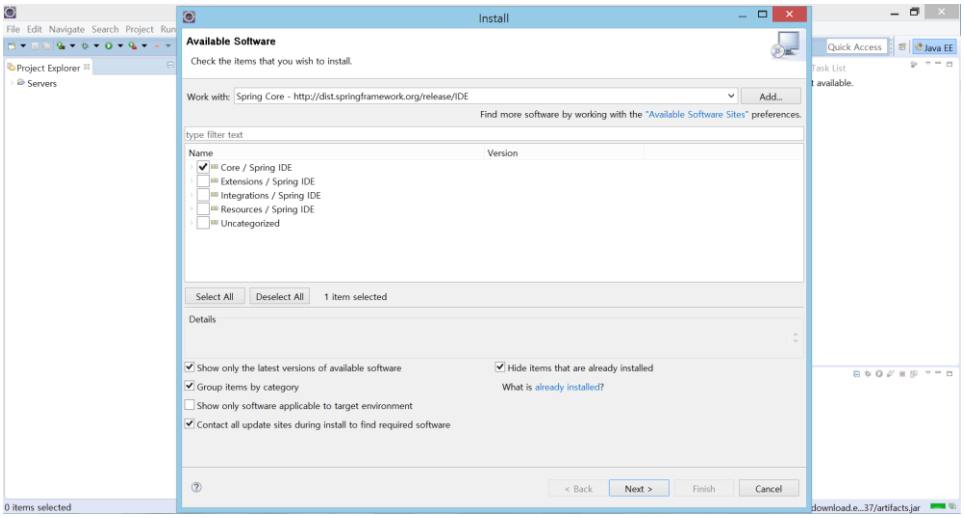
2.2.3 기존 Eclipse에 플러그인 설치 (윈도우, 맥, 리눅스)

Eclipse에 표준프레임워크 도구를 추가로 설치하여 활용하는 방법으로 맥, 리눅스 사용자는 이 방법으로 Eclipse에 플러그인 설치를 하여 활용할 수 있다. 추가적인 플러그인 설치 없이도 개발은 가능하지만 egovFrame전용 도구 (UML Editor, ERD Editor, 공통컴포넌트 활용 등) 활용이 불가능하다. 먼저 공통모듈인 Spring Core IDE를 설치하고, 표준프레임워크 플러그인을 설치한다. Spring Core IDE 설치를 위해 eclipse에서 Help를 선택하고, Install New Software를 선택하고, Add버튼을 누른다. Add Repository에서 그림과 같이 Spring Core IDE 다운로드 URL을 입력하고 OK를 누른다.



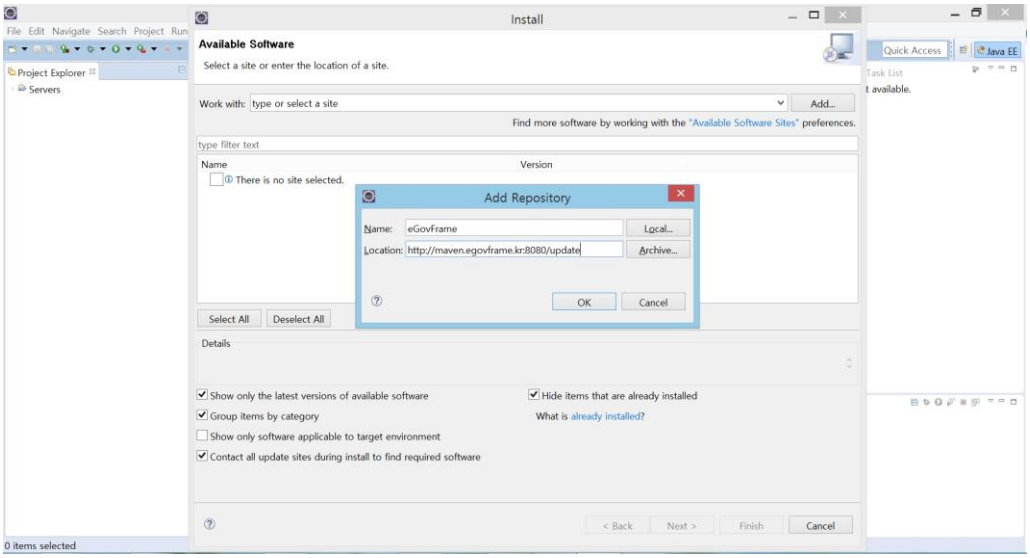
[그림 2-8] Spring Core IDE 설치 URL 설정

화면과 같이 Core를 선택하고 Next를 누르고, 약관에 동의한후 Finish를 눌러서 설치한다. 설치가 완료되면 eclipse를 재시작 한다.



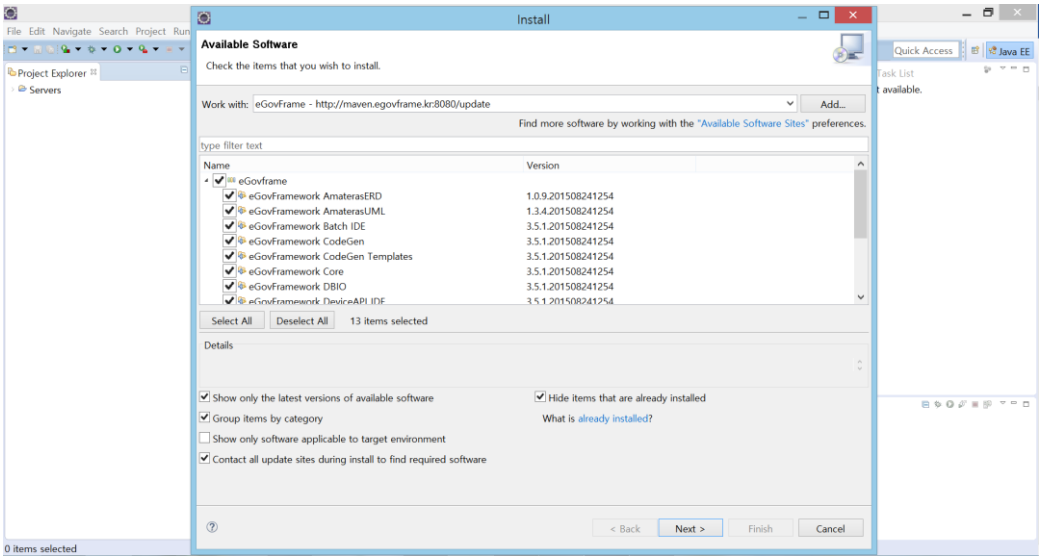
[그림 2-9] Spring Core IDE 설치

재시작이 되었으면, Help에서 Install New Software를 선택하고, Add를 누른 후에 다음과 같이 표준프레임워크 플러그인 다운로드 URL을 입력하고, OK를 누른다.



[그림 2-10] 표준프레임워크 플러그인 설치 URL 설정

화면과 같이 설치할 플러그인을 선택한 후에 Next를 누르고, 약관에 동의한후 Finish를 눌러서 설치한다. 설치가 완료되면 Eclipse를 재시작 한다.



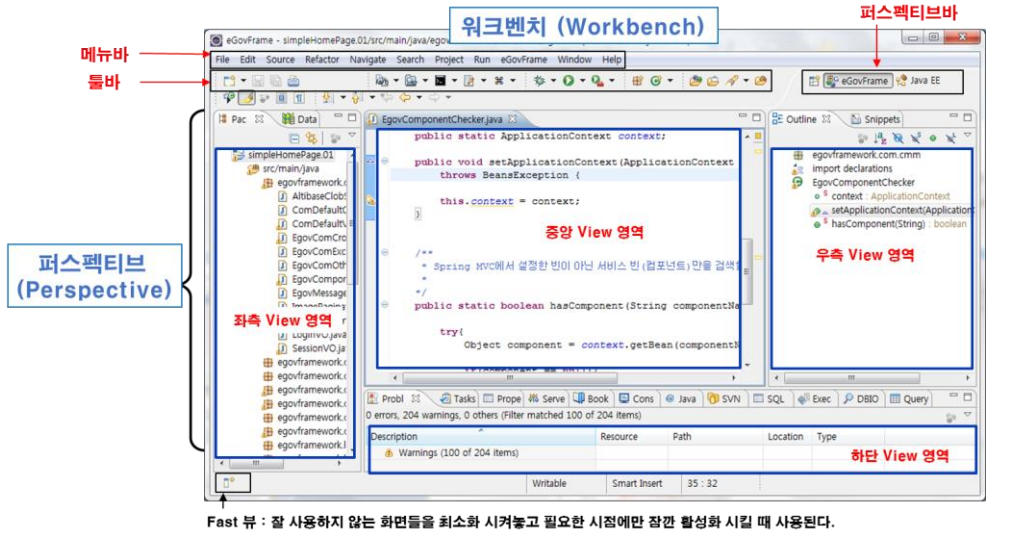
[그림 2-11] 표준프레임워크 플러그인 설치

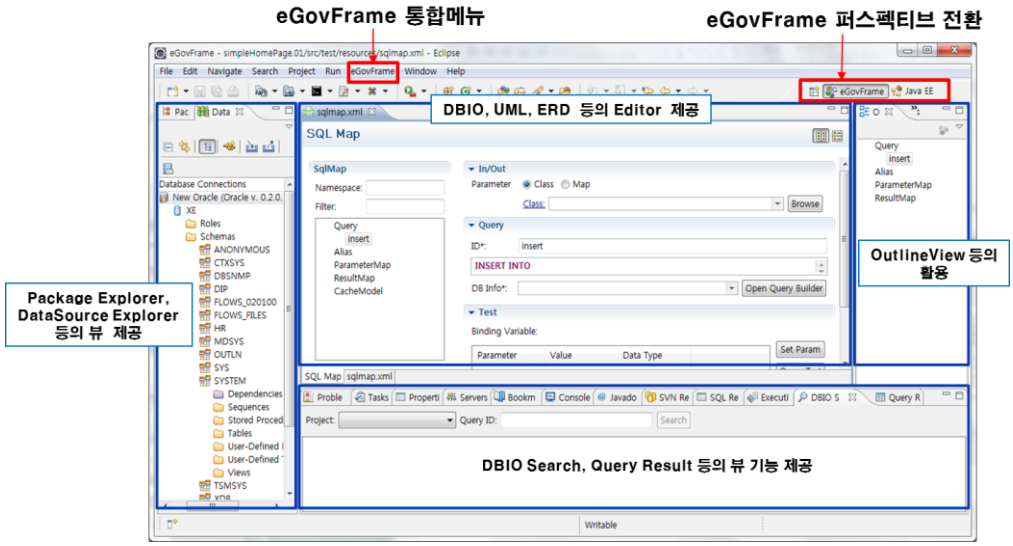
3. 개발자 개발환경 활용

표준프레임워크 개발자 개발도구는 구현도구인 Eclispe를 활용하여 개발하고, Maven을 활용하여 라이브러리 관리 및 빌드를 수행 한다.

3.1 구현도구 활용

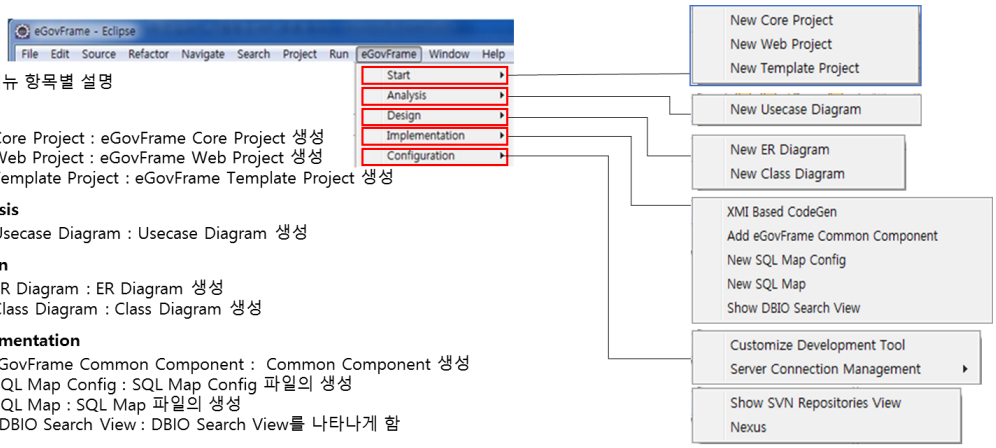
구현도구는 Eclipse Java EE를 기반으로 표준프레임워크 플러그인이 설치되어 있다. Eclipse 기본화면은 메뉴바, 툴바, 퍼스펙티브바, 4개의 뷰영역으로 크게 구분된다. 이 4개의 뷰를 포함하는 전체 영역을 퍼스펙티브(Perspective) 라고 하고, 이 Eclipse Window 전체를 워크벤치(Workbench)라 부른다.





[그림 3-1] Eclipse 실행화면

구현도구는 개발자가 개발환경을 편리하게 개발할 수 있도록 특성화된 통합메뉴, 퍼스펙티브, 뷰, 에디터 등을 제공한다. eGovFrame 퍼스펙티브를 선택하면, 메뉴에 eGovFrame 통합 메뉴가 나타나게 된다. 통합 메뉴는 유형별 프로젝트 생성, 각종 다이어그램(Use Case, ERD, Class) 작성, SQLMap 편집, 공통컴포넌트 생성 등 기능을 제공한다. 세부 메뉴 항목별 설명은 아래와 같다.



* 세부 메뉴 항목별 설명

❑ **Start**
- New Core Project : eGovFrame Core Project 생성
- New Web Project : eGovFrame Web Project 생성
- New Template Project : eGovFrame Template Project 생성

❑ **Analysis**
- New Usecase Diagram : Usecase Diagram 생성

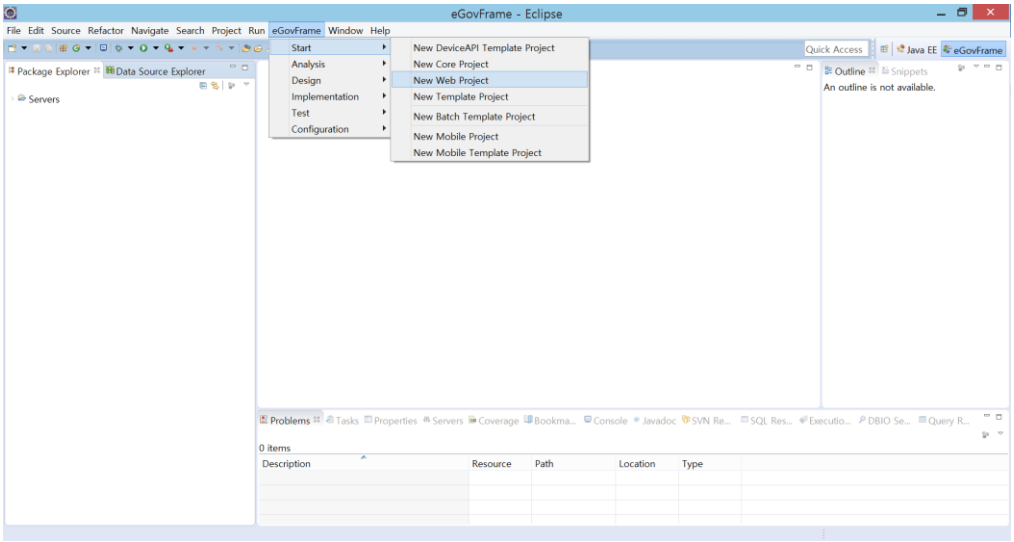
❑ **Design**
- New ER Diagram : ER Diagram 생성
- New Class Diagram : Class Diagram 생성

❑ **Implementation**
- Add eGovFrame Common Component : Common Component 생성
- New SQL Map Config : SQL Map Config 파일의 생성
- New SQL Map : SQL Map 파일의 생성
- Show DBIO Search View : DBIO Search View를 나타나게 함

❑**Configuration**
- Customize Development Tool : 필요한 기능을 선택적으로 설치
- Server Connection Management ・Show SVN Repositories View : SVN Repositories View를 나타나게 함
  ・Nexus : Nexus Repository 정보 관리

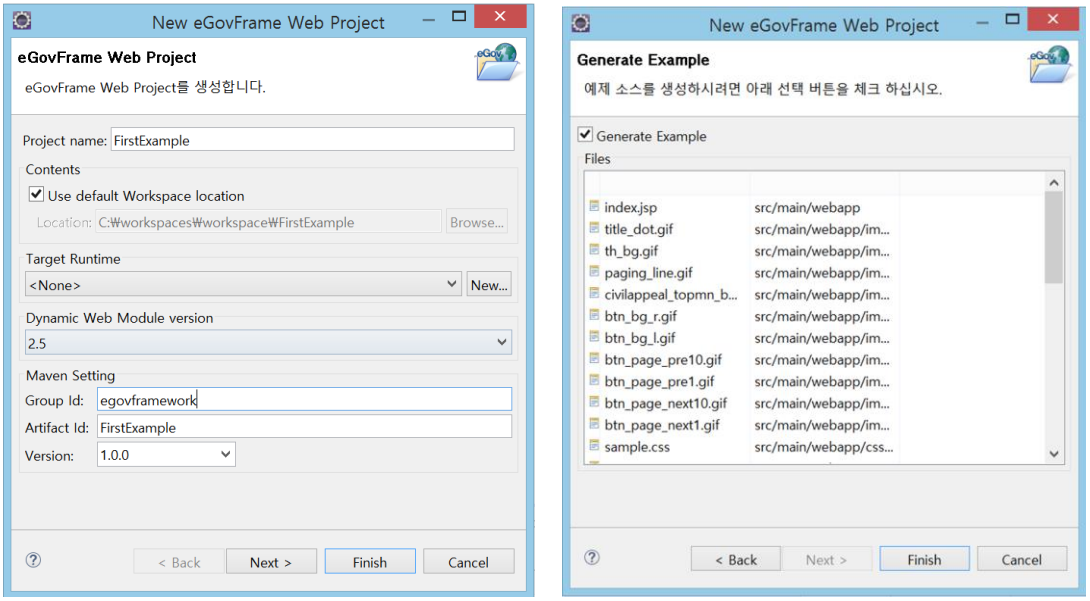
[그림 3-2] Eclipse eGovFrame 퍼스펙티브

구현도구를 테스트 해보기 위해서 다음과 같이 실행해보도록 한다. 프로젝트를 구성하고 예제코드를 자동으로 구성하여 실행하게 된다. 먼저 Eclipse에서 eGovFrame>Start>New eGovFrame Web Project를 선택한다. eGovFrame 메뉴가 보이지 않는다면 퍼스펙트가 eGovFrame으로 되어 있는지 확인해보도록 한다.



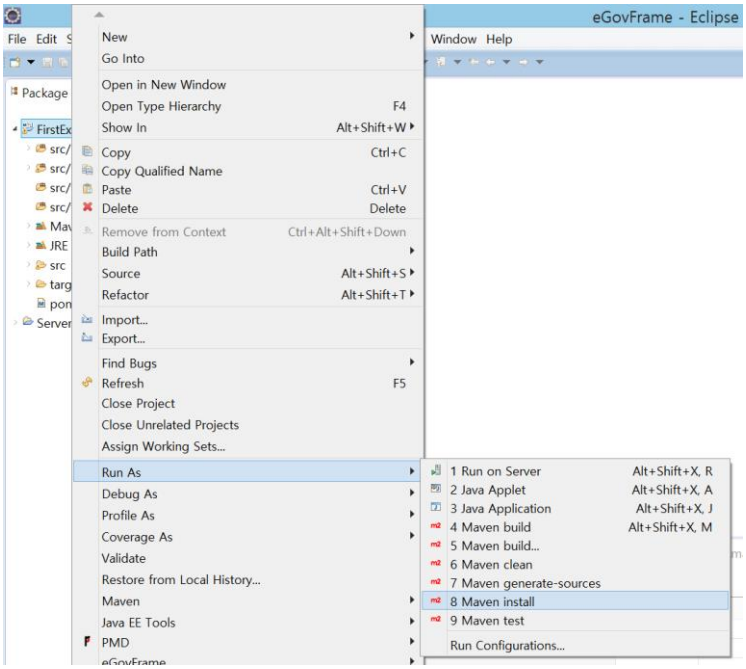
[그림 3-3] Web Project 생성

그리고 프로젝트 생성을 위해서 Project name은 FisrtExample로 입력한다. Group Id는 egovframework로 입력하고 Next를 누른다. 다음 화면에서 Generate Example을 반드시 체크하여 선택하고, Finish를 누른다.



[그림 3-4] 생성할 Project 정보입력

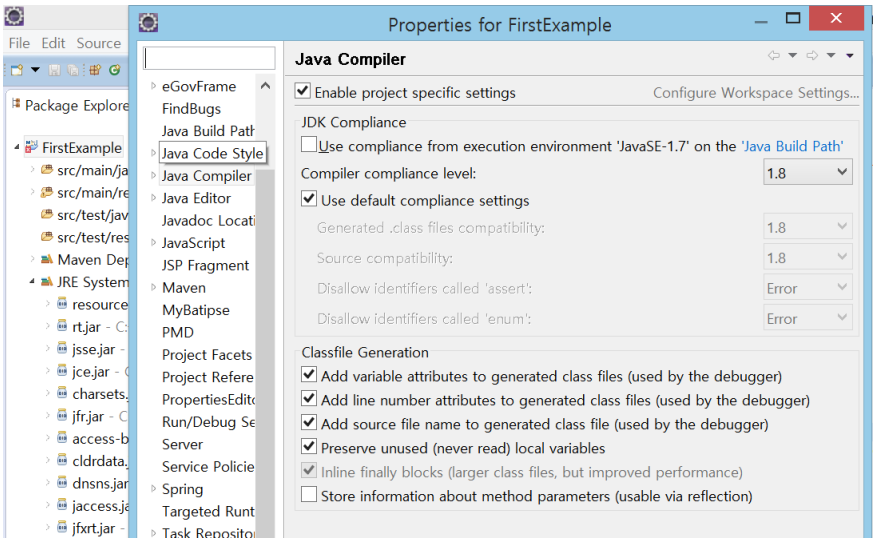
잠시 후 프로젝트 생성이 완료되면, 마우스 오른쪽 클릭하고 Run As>Maven install을 선택한다. Maven을 활용하여 프로젝트를 빌드하는 과정이다. Maven에 대해서는 3.2에서 설명하도록 하겠다.



[그림 3-5] 빌드하기

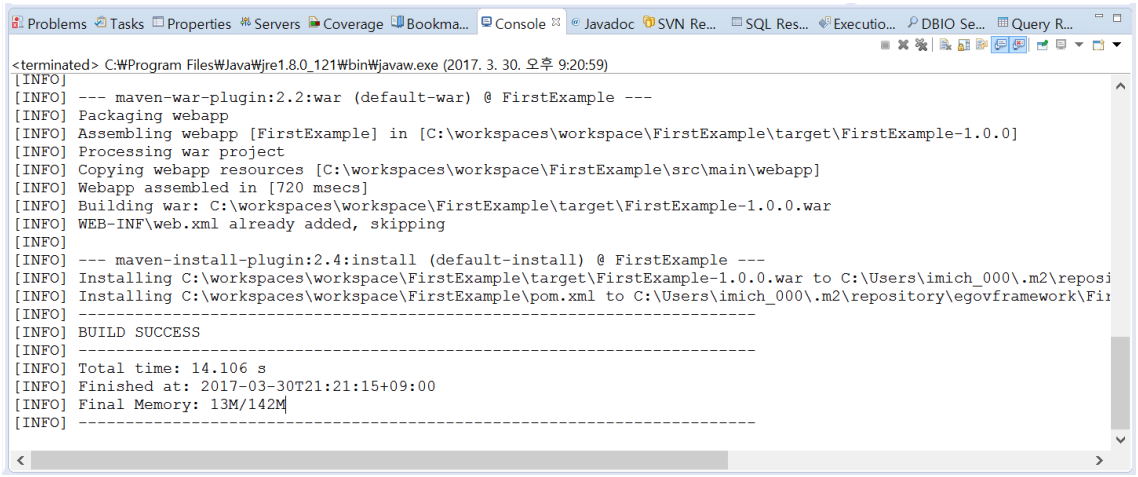
JDK를 1.7이 아닌 버전을 사용하게 되면, 다음과 같은 에러가 발생하게 된다. 현재 생성된 소스코드가 JDK 1.7로 설정이 맞춰져 있기 때문이며, Project를 선택하고 마우스 오른쪽 버튼을 클릭, Properties를 선택하고 Java Compiler를 해당 사용하고 있는 JDK버전으로 수정한 후에 다시 Maven install을 수행한다.

```
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?
```



[그림 3-6] 에러 발생 시에 컴파일러 버전 수정

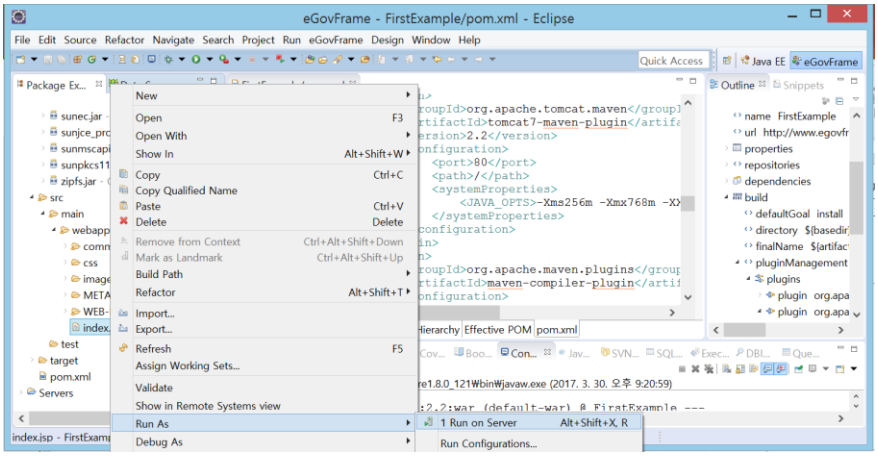
빌드가 완료되면 다음과 같이 Console에 출력되는 것을 볼 수 있다.



[그림 3-7] 빌드 완료 화면

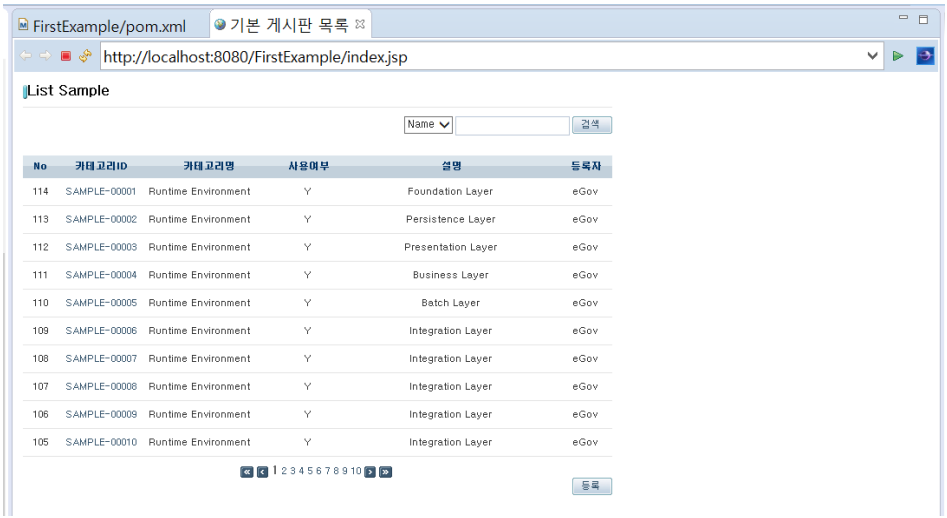
이제 생성된 소스코드를 실행하기 위하여 생성된 프로젝트를 오픈하고, src>main>webapp 밑에 index.jsp를 선택하고 마우스 오른쪽 버튼을 누른다. Run As>Run on Server를 선택한

다. 구현도구를 직접 설치한 경우에는 Tomcat이 설정되어 있어야 한다.



[그림 3-8] 에러 발생 시에 컴파일러 버전 수정

화면에서 Next를 누리고 Finish를 누르면 Console에 로그들이 보이다가 다음과 같이 게시판이 실행된 화면을 볼 수 있다.



[그림 3-9] 실행완료 화면

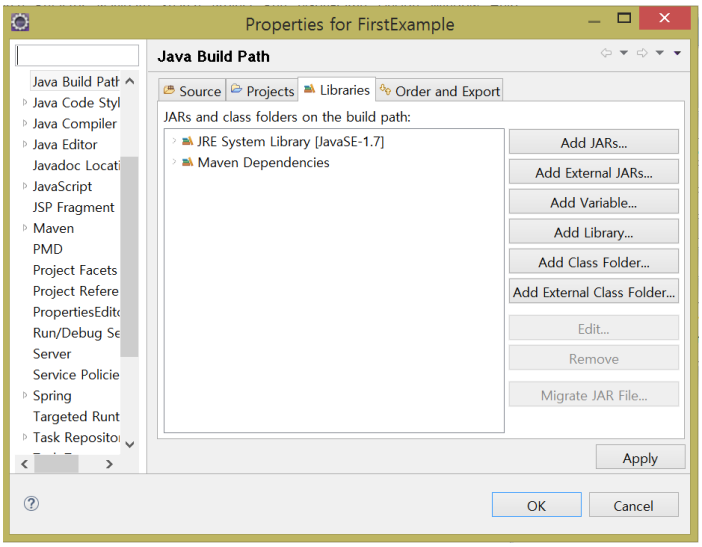
3.2 Maven 활용

Maven은 프로젝트의 라이브러리 의존성을 관리하고 컴파일, 빌드, 배포에 이르는 전체 개발 과정을 관리한다.

3.2.1 라이브러리 의존성 관리

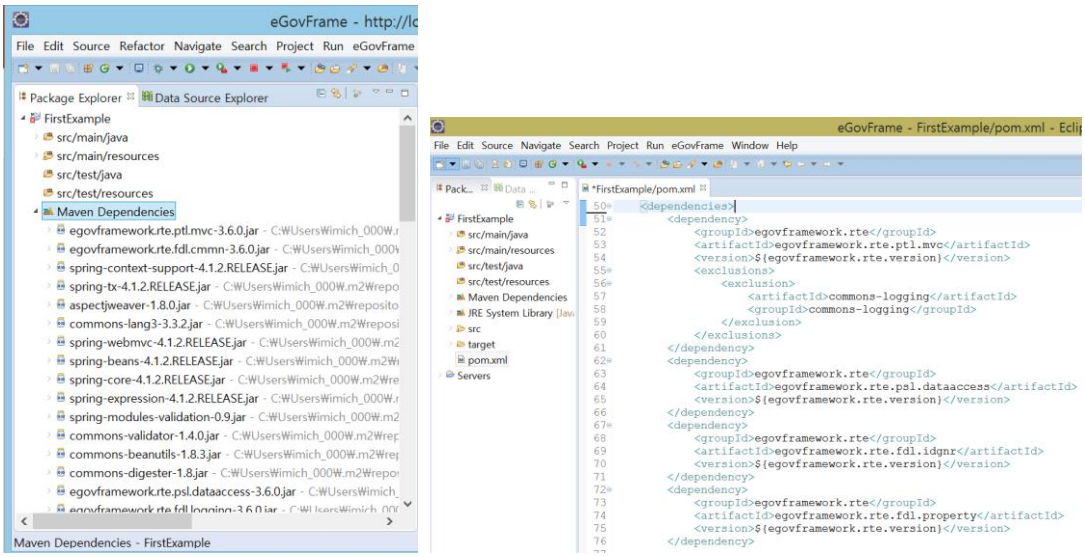
자바를 활용하여 프로그래밍을 할 때 어려운 부분 중에 하나는 컴파일과 실행을 위해 라이브러리를 설정하고 관리하는 것이다. 자바는 기존에 개발된 수많은 라이브러리를 활용하여 필요한 기능을 쉽게 확장할 수 있다는 장점이 있다. 하지만, 자바 라이브러리는 종속성 및

버전관리가 어려운 문제가 있다. 즉 특정 라이브러리를 활용하기 위해서는 참조하고 있는 다른 라이브러리도 함께 활용이 되어야 한다. A라는 라이브러리가 B와 C라이브러리 기능을 활용하고 있다면 A만 가지고는 활용이 불가능하고, B와 C를 함께 참조해야만 한다. 그리고 버전의 문제도 있다. 앞서 예를 든 A라이브러리의 버전에 따라 B 또는 C 라이브러리가 통용이 되는 버전이 존재 한다는 것이다. 예를 들어 A라이브러리 3.0은 B라이브러리 2.0 이상이 활용되어야 한다는 등의 형태이다. 일반적으로 라이브러리 관리는 Build Path를 통해서 이루어지게 된다. 프로젝트 마우스 오른쪽 클릭하여 Build Path>Configure Build Path를 선택하고 Libraries 탭을 선택하면 새로운 라이브러리를 등록하거나 기존 설정을 수정, 제거할 수 있다.



[그림 3-10] 라이브러리 Build Path 설정

프로젝트에서 활용되는 수많은 라이브러리들의 Build Path를 개별적으로 설정해야 한다면 매우 불편할 것이다. 라이브러리들 간의 의존성과 버전의 조합은 경우의 수가 너무 많기 때문에 잘 관리될 수 있는 특별한 방법을 필요로 하게 되었다. Maven은 이러한 라이브러리 관리 문제를 해결하기 위해 의존성 설정을 활용하여 라이브러리를 관리할 수 있다. 3.1의 소스코드에서 Maven Dependencies라고 되어 있는 것을 펼쳐서 열어보면 프로젝트에서 활용되고 있는 라이브러리 목록(43개의 jar파일)을 확인 할 수 있다. 프로젝트 아래에 있는 pom.xml을 열어보면 <dependency>라고 표시된 내용을 확인 할 수 있다. pom.xml의 맨 위에 있는 “egovframework.rte.ptl.mvc”라고 되어 있는 것을 확인 할 수 있고, 라이브러리 목록 맨 위에 보면 egovframework.rte.ptl.mvc-3.6.0.jar라는 라이브러리를 확인 할 수 있다. 그렇다면 pom.xml에는 라이브러리 개수 만큼 dependency가 설정되어 있는 것일까? 그렇지 않다. 설정되어 있는 Dependency의 수는 9개 이다. 그렇다면 나머지는 어떻게 설정되는 것일까? 답은 Dependency에 있다. egovfraework.rte.ptl.mvc에서 참조하고 있는 라이브러리도 함께 활용되게 된다.



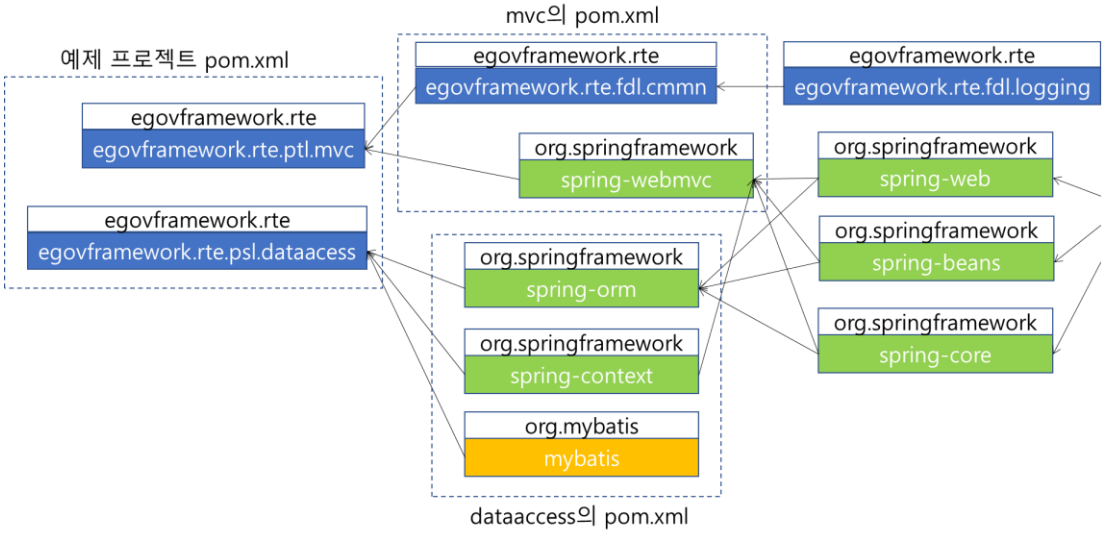
[그림 3-11] pom.xml과 Dependency

Maven Dependencies에서 jar 파일 옆에 파일 위치가 표시되는 것을 확인 할 수 있다. 표준프레임워크 교육교재로 실행을 하게 되면 C:\eGovFrame-3.6\maven\repository 밑에 위치하게 되고 개발환경을 별도로 구성한 경우에는 사용자 디렉터리 밑에 .m2\repository 밑에 위치한다. 위의 디렉터리를 Maven Local Repository라고 부르며, 위치는 설정파일(표준프레임워크에서는 settings.xml)을 활용하여 위치를 지정 할 수 있다. jar파일을 찾기 위해서 Repository 디렉터리를 찾아가보면 pom.xml에서 group Id, artifact Id, 버전명으로 디렉터리 명이 구성되고, 다시 artifact Id로 라이브러리 명이 구성되어 있는 것을 알 수 있다. 디렉터리에는 jar파일도 위치하지만 pom파일이 존재하고 있어 이 라이브러리와 의존성 있는 라이브러리 정보를 함께 지정하고 있다.

[그림 3-12] Maven Local Repository

egovframework.rte.ptl.mvc-3.6.0과 종속성을 가지는 라이브러리 정보는 pom파일에 기술되어 있다. spring-webmvc와 종속성을 가지도록 설정되어 있고, 다시 spring-webmvc의 pom.xml을 살펴보면, spring-web, spring-beans, spring-core, spring-context 등 주요

spring 라이브러리와 종속성을 가지고 있음을 알 수 있다. rte.psl.dataaccess의 종속성도 비슷하게 설명이 될 수 있다. 결국 mvc와 dataaccess를 pom.xml에 기술하게 되면 개별적으로 모든 라이브러리 의존성 설정을 하지 않아도 종속성이 있는 라이브러리들이 자동으로 설정되게 된다.



[그림 3-13] pom.xml 간의 라이브러리 의존성

pom.xml에서 라이브러리 의존성은 다음과 같이 설정한다.

```
<dependency>
    <groupId>egovframework.rte</groupId>  → 라이브러리 식별 namespace
    <artifactId>egovframework.rte.ptl.mvc</artifactId> → 라이브러리 명
    <version>${egovframework.rte.version}</version> → 버전정보
    <exclusions>  → 옵션이며, 충돌 등의 이유로 특정 라이브러리 의존성 제거
        <exclusion>
                <artifactId>commons-logging</artifactId>
                <groupId>commons-logging</groupId>
        </exclusion>
    </exclusions>
</dependency>
```

[소스 3-1] 라이브러리 설정

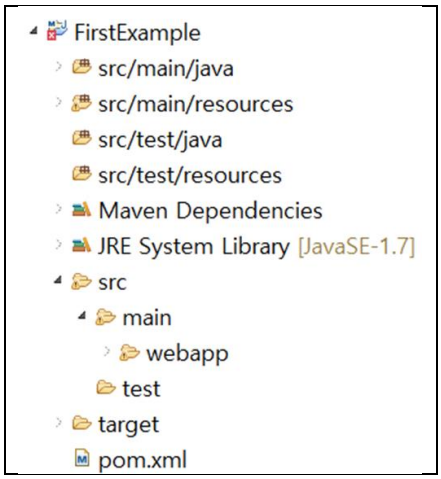
처음 Maven을 실행할 때 Local Repository에 라이브러리가 없으면 pom.xml의 repository에 기술된 Remote Repository로부터 라이브러리를 다운로드 받게 된다. 표준프레임워크는 다음과 같이 두개의 Remote Repository를 사용하고 있으며, 사용자가 Nexus 등을 활용하여 개별적으로 3rd Party 라이브러리를 구성할 수도 있다. 특히 인터넷이 차단된 폐쇄망에서는 필수적으로 별도 repository를 구성해야 한다.

```
<repository>
        <id>mvn2</id>
        <url>http://repo1.maven.org/maven2/</url>
        <releases>
                <enabled>true</enabled>
        </releases>
        <snapshots>
                <enabled>true</enabled>
        </snapshots>
</repository>
<repository>
        <id>egovframe</id>
        <url>http://www.egovframe.go.kr/maven/</url>
        <releases>
                <enabled>true</enabled>
        </releases>
        <snapshots>
                <enabled>false</enabled>
        </snapshots>
</repository>
```

[소스 3-2] Remote Repository 설정

3.2.2 빌드 관리

Maven에서는 디렉터리 구조를 정규화하여 복잡한 설정 없이 컴파일, 패키징 등을 수행할 수 있다. 먼저 실제 소스파일과 테스트 소스 파일로 구성이 되며, resources 디렉터리에는 jar나 war로 패키징 될 때 함께 배포되어야 할 설정파일 등의 리소스가 위치하게 된다. 웹 프로젝트의 경우 webapp 밑에 MVC 설정과 웹 리소스 (HTML, 자바스크립트, 이미지 등)가 위치 한다.



디렉터리/파일	설명
pom.xml	프로젝트 객체 모델로 빌드, 의존성 등 정보를 가진다.
/src/main/java	Java 소스파일 위치
/src/main/resources	배포할 리소스 XML, Properties, 설정파일
/src/main/webapp	웹 어플리케이션 리소스 (HTML, Javascript 등)
/src/test/java	테스트 케이스 소스파일
/src/test/resources	테스트 케이스 리소스
/target	빌드된 output 위치

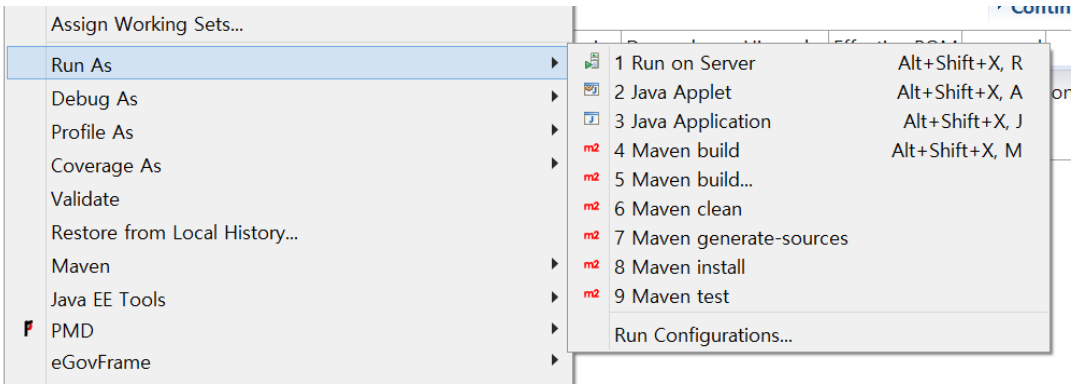
[그림 3-14] Maven 디렉터리 정보

Maven 빌드는 유효성확인부터, 컴파일, 패키징, 배포 등의 빌드 생명주기를 지원하고 있다. 다음 표는 빌드 생명주기를 설명하고 있다.

[표 3-1] Maven 빌드 생명주기 단계

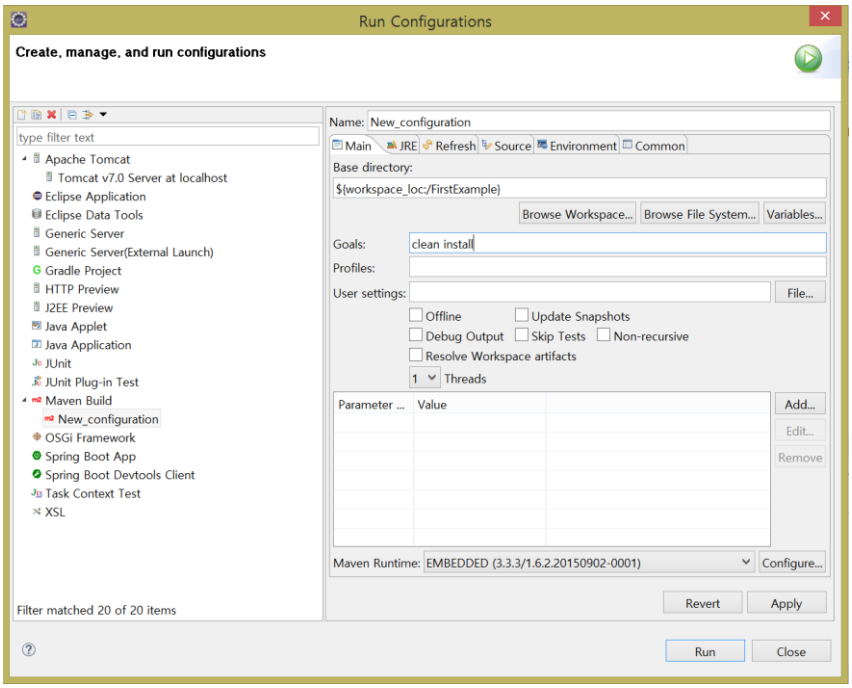
빌드 생명주기단계	설명
validate	현재설정과 pom.xml의 내용이 유효한지 확인
generate-sources	플러그인에서 추가적인 소스코드 생성이 필요한 경우 실행이 된다.
compile	소스코드를 컴파일 한다. 컴파일 된 클래스들은 타겟 디렉터리에 저장된다.
test	src/test/java에 있는 테스트 케이스를 실행하고 결과를 표시한다.
package	실행 가능한 바이너리 파일들을 jar, war 형태의 배포형 압축파일로 만든다
install	압축 파일을 로컬 Maven Repository에 추가한다.
deploy	압축파일을 Nexus와 같은 원격 Maven Repository에 추가한다

Maven을 실행하기 위해서는 프로젝트 폴더나 pom.xml을 선택한 후에 마우스 오른쪽 버튼을 클릭하고 Run>As를 선택하면 된다. 앞에서 설명된 생명주기 단계에 따라 빌드를 수행할 수 있다. 일반적으로 Maven install을 디폴트로 수행한다. 빌드 생명주기가 선택되어 실행될 경우 앞 단계에 있는 것들이 순차적으로 실행된다. 예를 들어 Maven test를 실행하게 되면 validate>generate-sources>compile>test의 순서로 실행된다.



[그림 3-15] Maven 빌드 생명주기

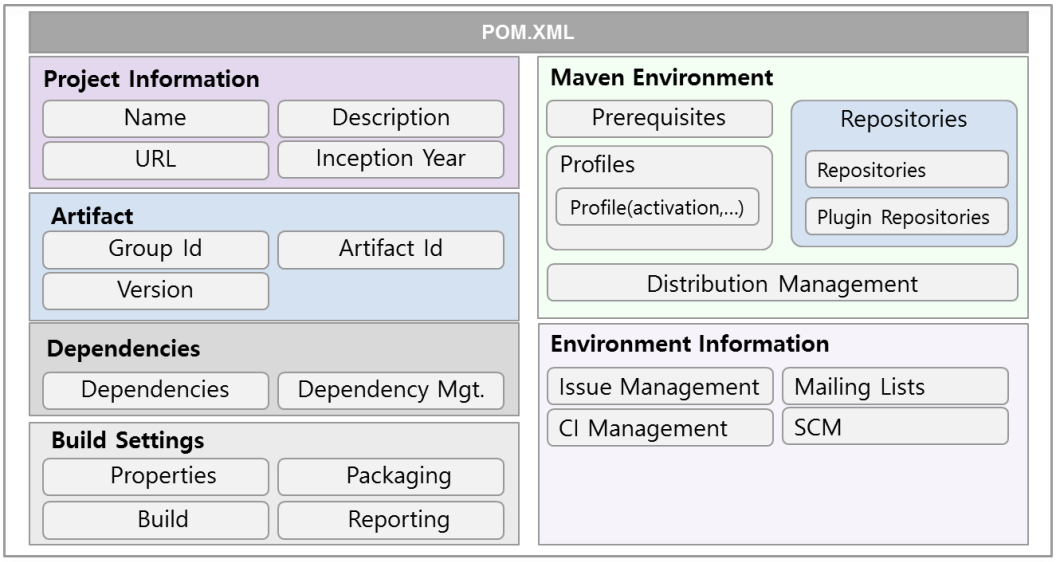
표준프레임워크 개발환경이 아닌 Eclipse에서는 Run Configuration을 통해서 Maven 실행을 설정할 수 있다. 왼쪽 위의 생성버튼을 누르면 화면이 나타나게 된다. 빌드를 수행할 프로젝트를 선택하고 Goals를 입력한다. “clean install”의 경우는 maven clean과 maven install이 순차적으로 실행되는 형태이다.



[그림 3-16] Maven 빌드 설정하기

3.2.3 Maven 설정

pom.xml 은 프로젝트의 정보, 의존성, 빌드 세팅(플러그인) 정보를 포함하고 있다. Maven 프로젝트는 빌드 되어 로컬 또는 원격 Repository 에 배포될 수 있기 때문에 자체적으로 Group Id, Artifact Id, 버전 정보를 가진다. 앞에서 설명했던 라이브러리 관리를 위한 의존성 관리를 할 수 있다. 컴파일, 패키징 뿐만 아니라 테스트 결과 리포트, java doc 생성과 같은 작업들을 플러그인 설정을 통해서 수행 할 수 있다.



[그림 3-17] pom.xml 구조

pom.xml 예제는 다음과 같다.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>egovframework.dev.com</groupId>
  <artifactId>egovframework-dev-com</artifactId>
  <version>1.0</version>
  <packaging>war</packaging>
  <name>egovframework-dev-com Maven Webapp</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.4</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
  </plugins>
</project>
```

Artifact 정보

의존성 관리

플러그인 선언

[소스 3-3] pom.xml 예제

Maven 은 빌드 작업시에 플러그인을 활용하여 생명주기 단계별로 플러그인을 바인딩하여 작업을 처리하게 된다. 플러그인의 설정은 pom.xml 에서 pluginManagement 에 기술된다. 현재 표준프레임워크에서 설정하여 제공하고 있는 주요 Maven 플러그인 정보는 다음과 같다.

[표 3-2] Maven 플러그인

단계	plugin	설명
compile	maven-compiler-plugin	소스코드 컴파일
test	maven-test-plugin maven-surefire-plugin	단위테스트 (Junit) 실행과 리포트 생성
package	maven-jar-plugin maven-war-plugin	압축파일 생성
emma	maven-emma-plugin	Code coverage report 생성

Maven install로 빌드를 수행하게 되면 다음 화면과 같이 compile, test, package, install의 순서로 실행되는 것을 확인할 수 있다. 처음 compile 단계에서 resources 플러그인이 실행되어 소스코드가 copy 되고, compiler 플러그인이 동작하여 컴파일을 수행한다. Test단계에서 test 플러그인이 실행되어 단위테스트를 수행하고, surefire 플러그인을 통해 테스트결과 리포트를 생성한다. Package 단계에서 war 플러그인이 동작되어 war 배포파일이 생성된다. Install단계에서 로컬 Repository에 배포되게 된다.

```
[INFO] ------------------------------------------------------------------------
[INFO] Building FirstExample 1.0.0
```

```
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ FirstExample ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] Copying 18 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ FirstExample ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @
FirstExample ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ FirstExample --
-
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ FirstExample ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ FirstExample ---
[INFO] Packaging webapp
[INFO] Assembling webapp [FirstExample] in
[C:\workspaces\workspace\FirstExample\target\FirstExample-1.0.0]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\workspaces\workspace\FirstExample\src\main\webapp]
[INFO] Webapp assembled in [1000 msecs]
[INFO] Building war: C:\workspaces\workspace\FirstExample\target\FirstExample-1.0.0.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ FirstExample ---
[INFO] Installing C:\workspaces\workspace\FirstExample\target\FirstExample-1.0.0.war to
C:\Users\imich_000\.m2\repository\egovframework\FirstExample\1.0.0\FirstExample-
1.0.0.war
[INFO] Installing C:\workspaces\workspace\FirstExample\pom.xml to
C:\Users\imich_000\.m2\repository\egovframework\FirstExample\1.0.0\FirstExample-
1.0.0.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 7.248 s
[INFO] Finished at: 2017-04-04T18:40:31+09:00
[INFO] Final Memory: 11M/126M
[INFO] ------------------------------------------------------------------------
```

[그림 3-18] Maven install 수행 결과

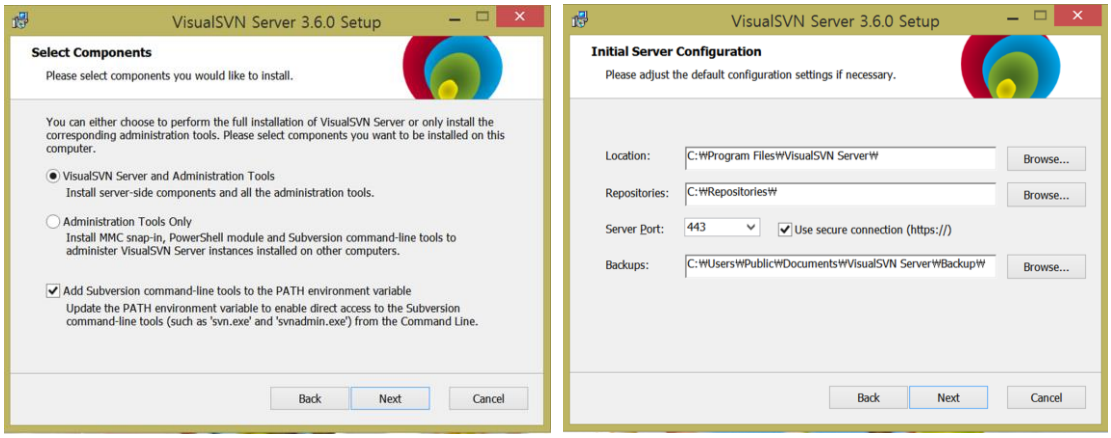
4. 서버 개발환경 활용

표준프레임워크 형상관리 도구인 Subversion을 활용하여 팀프로젝트를 통합하고, 배포도구인 Jenkins를 활용하여 계속적인 테스트와 통합을 수행 한다.

4.1 형상관리 도구 활용

4.1.1 Subversion 개요 및 설치

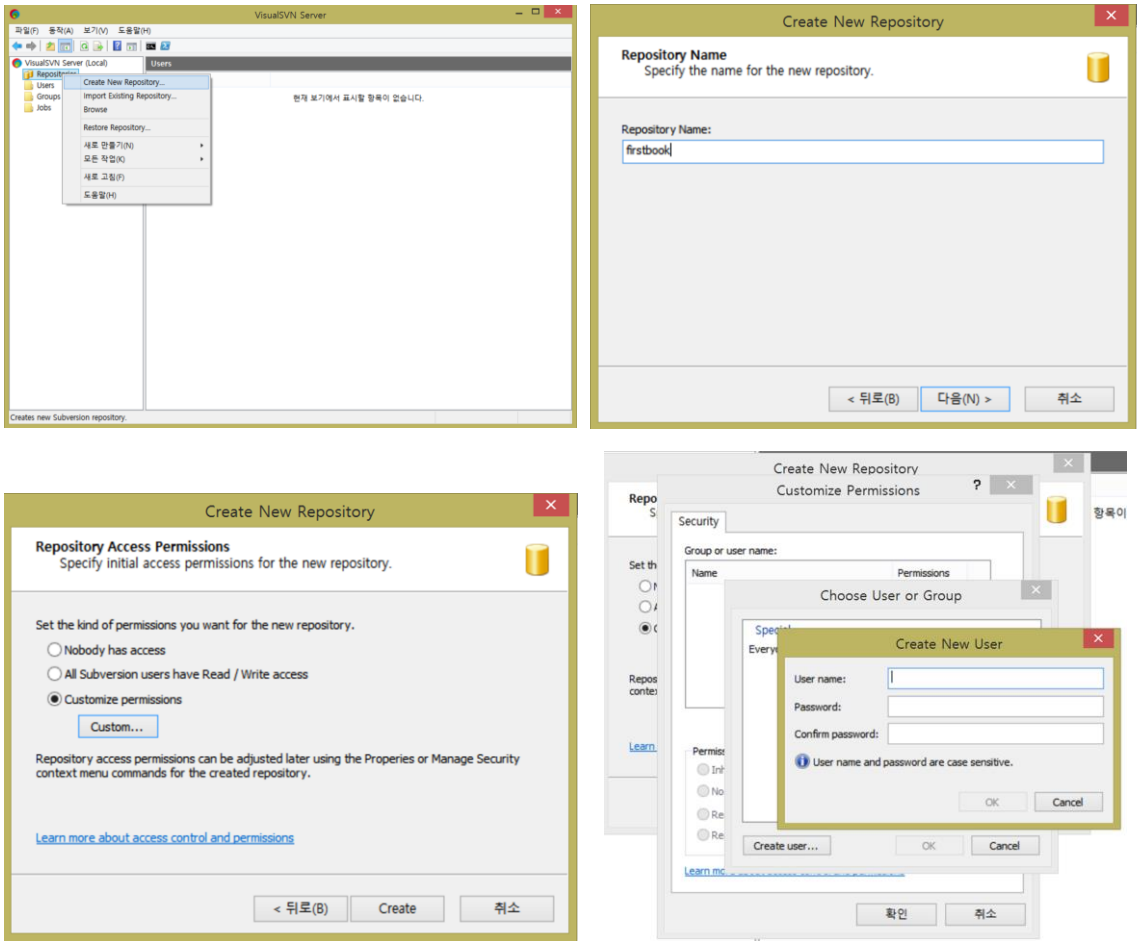
형상관리 도구는 팀프로젝트 수행시에 개인 개발자들이 작성한 소스코드를 통합하여 하나의 프로젝트로 통합하고 변경된 내용을 버전관리 하여 변경 히스토리를 관리하고 필요한 경우 이전 버전으로 변경할 수 있도록 기능을 지원한다. 표준프레임워크 형상관리 도구인 Subversion은 파일 뿐 아니라 디렉터리도 버전관리를 지원하며, 변경관리를 변화된 부분만 관리하여 성능이 우수하다. 윈도우 환경의 서버인 VisualSVN을 활용하면 Subversion을 쉽게 설치할 수 있다. VisualSVN 사이트(https://www.visualsvn.com)에서 VisualSVNServer를 다운로드 받는다. 32bit와 64bit중에서 사용 중인 PC환경에 맞도록 선택한다. 설치파일을 실행하고 약관 동의를 한다. VisualSVN Server and Administration Tools를 선택하고 Next를 누른다. Standard Edition을 선택하고 설치위치, Repository 위치, 포트 등 정보를 입력한다. 테스트 용도로 설치하는 경우라면 디폴트 값을 그대로 활용해도 무방하다.



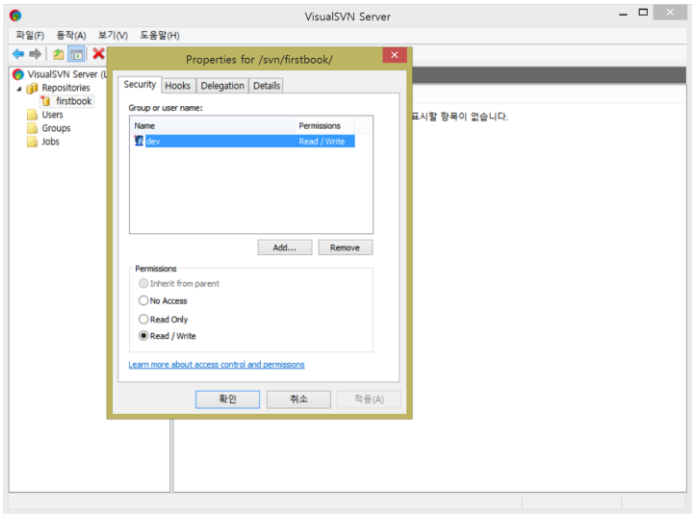
[그림 4-1] VisualSVN 설치화면

설치가 되었으면 VisualSVN Server Manager 를 실행하고, Repositories 를 선택하고 마우스 오른쪽 버튼을 누르고 Create New Repository 를 선택한다. Regular FSFS repository 를 선택하고, Repository 이름을 입력한다. 아무 이름이나 입력 가능하며, firstbook 으로 입력 하였다. empty Repository 를 선택하고 Access 권한 설정을 위해서 Customize permissions 를 선택한다. Add 를 선택, Create User 를 선택하고 User name 과 password 를 입력한다. 편의상 User name 을 dev 로 password 도 동일하게 dev 로 입력하였다. Users 에서 dev 를 선택하고 OK 를 누르고, 확인을 누른다. 권한이 Read/Write 로 되어 있는지 다시 한번 확인한다. 설정된 Repository 정보를 확

인하면 설정이 끝난다. 설정이 완료된 후 상단에 표시되는 URL 정보는 나중에 Eclipse 에 연동할 때 필요한 정보이므로 잘 기억해야 한다. (https://호스트명/svn/firstbook)



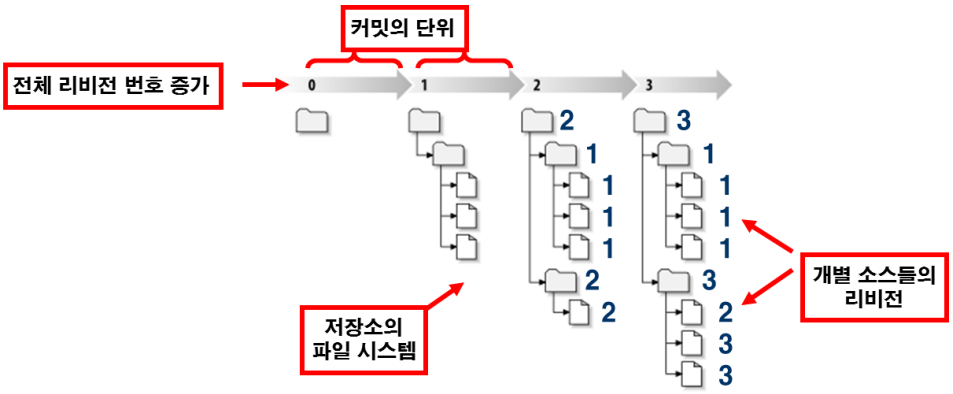
[그림 4-2] VisualSVN Repository 설정화면



[그림 4-3] VisualSVN Repository 설정 완료 화면

4.1.2 Revisions

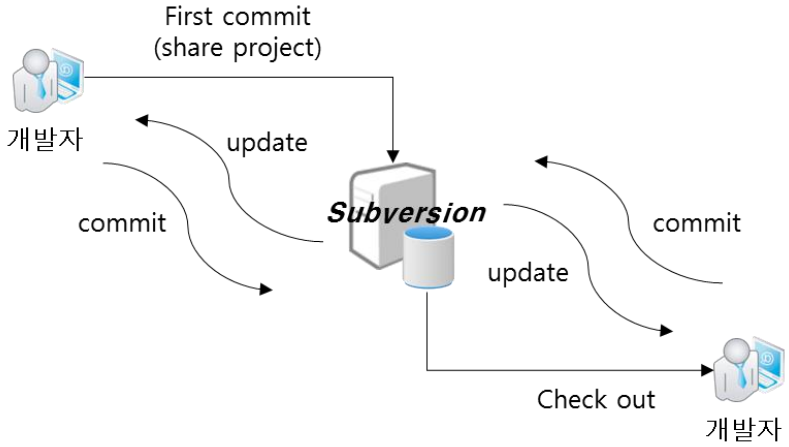
Revision 은 저장소에 저장된 각각의 파일 버전이라고 할 수 있다. 소스파일 등을 수정하여 저장(Commit)하게 되면 일정한 규칙에 의해서 숫자가 증가하게 된다. Subversion 은 변경 발생 시에 전체 단위로 Revision 을 갱신하여 관리한다. 개별 파일 단위로 Revision 을 관리하게 되면 전체적으로 소스코드에 변경이 일어난 순서 정보를 알기 어렵게 된다.



[그림 4-4] Subversion의 Revision 관리

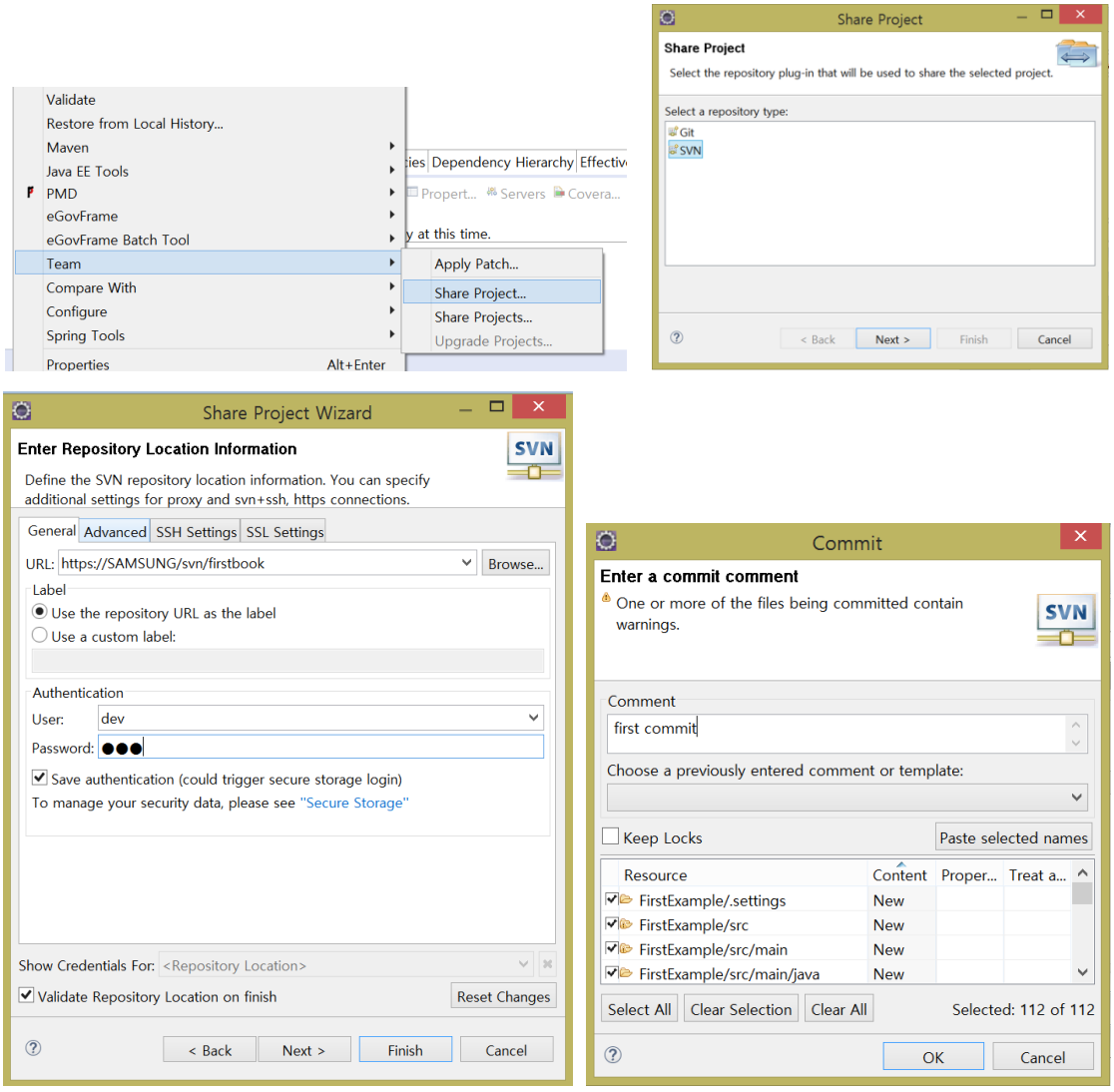
4.1.3 Eclipse 와 연동하기

팀 프로젝트에 참여하는 개발자들은 Eclipse 를 활용하여 소스코드를 공유하며 공동작업이 가능하다. Subversion 을 활용하기 위해 제일 먼저 해야 하는 일은 프로젝트를 최초 commit (Eclipse 에서 share project)하는 것이다. 최초 commit 이 되어 Subversion repository 에 저장이 되면 다른 개발자들이 자신의 workspace 로 내려 받아 활용할 수 있으며, 이를 check out 이라고 한다. Check out 을 수행하면 프로젝트가 생성되면서 Subversion 에 연결된 상태가 된다. 최초 check out 은 한번만 수행하면 되며, 그 이후에는 소스코드를 수정하여 commit 하거나 다른 개발자가 변경한 내역을 update 하여 소스코드를 통합 관리하게 된다.



[그림 4-5] Subversion을 활용한 팀프로젝트 적용

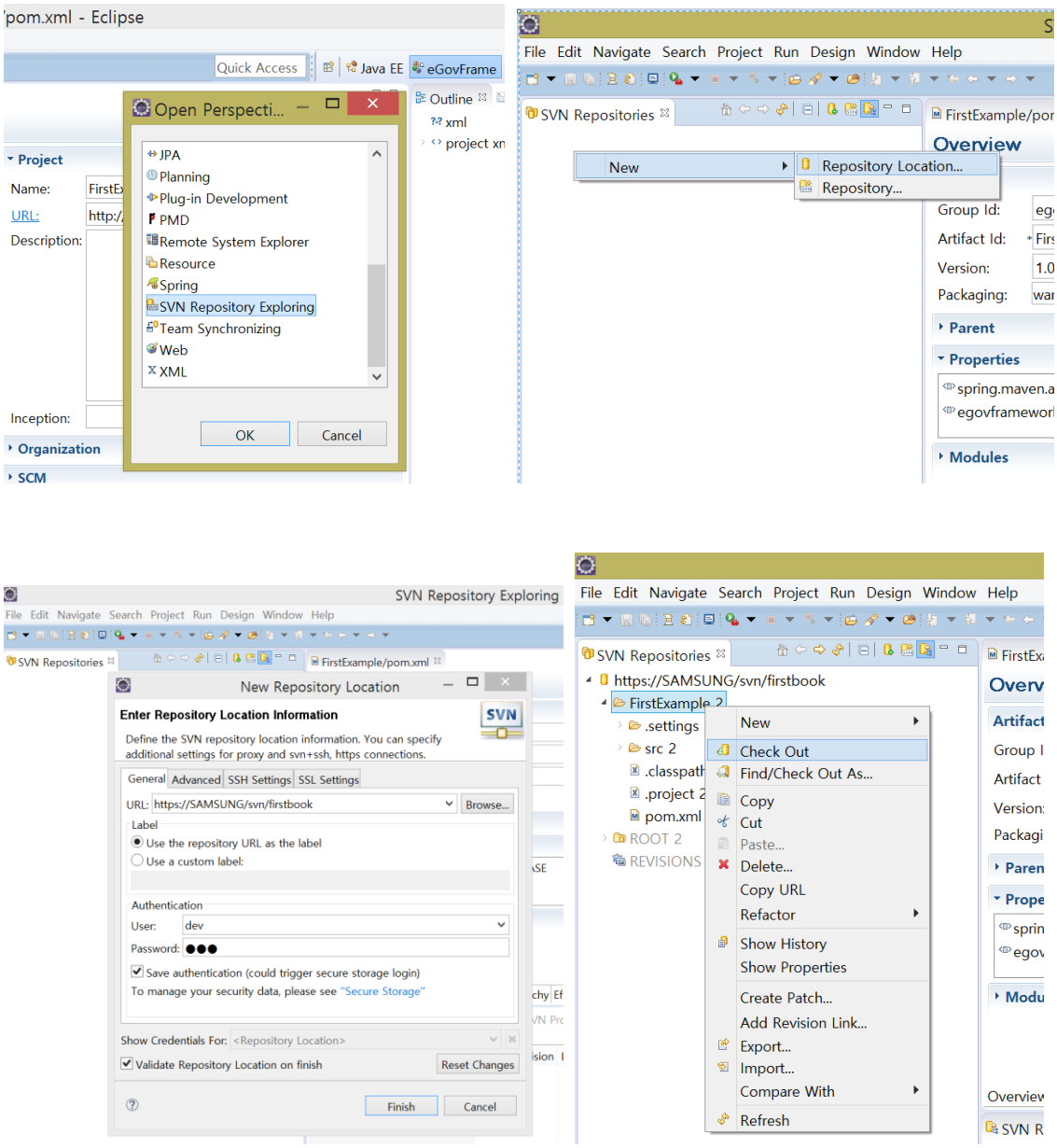
개발자가 처음 프로젝트를 Subversion 에 등록하기 위해서는 프로젝트를 선택하고 마우스 오른쪽 버튼을 눌러 Team>Share Project 를 선택하고, 형상관리 도구로 SVN 을 선택한다. Repository 정보를 입력하는 화면에서 URL 은 형상관리 설치 후에 VisualSVN Server Manager 에서 확인했던 정보를 입력한다. 호스트명은 IP 로 대체 가능하다. User 와 Password 를 입력하고, Finish 를 누르면 Commit 내용을 확인 할 수 있는 창이 보여지게 되며, comment 입력 후에 OK 를 누르면 First Commit 이 완료된다.



[그림 4-6] Subversion에 First Commit

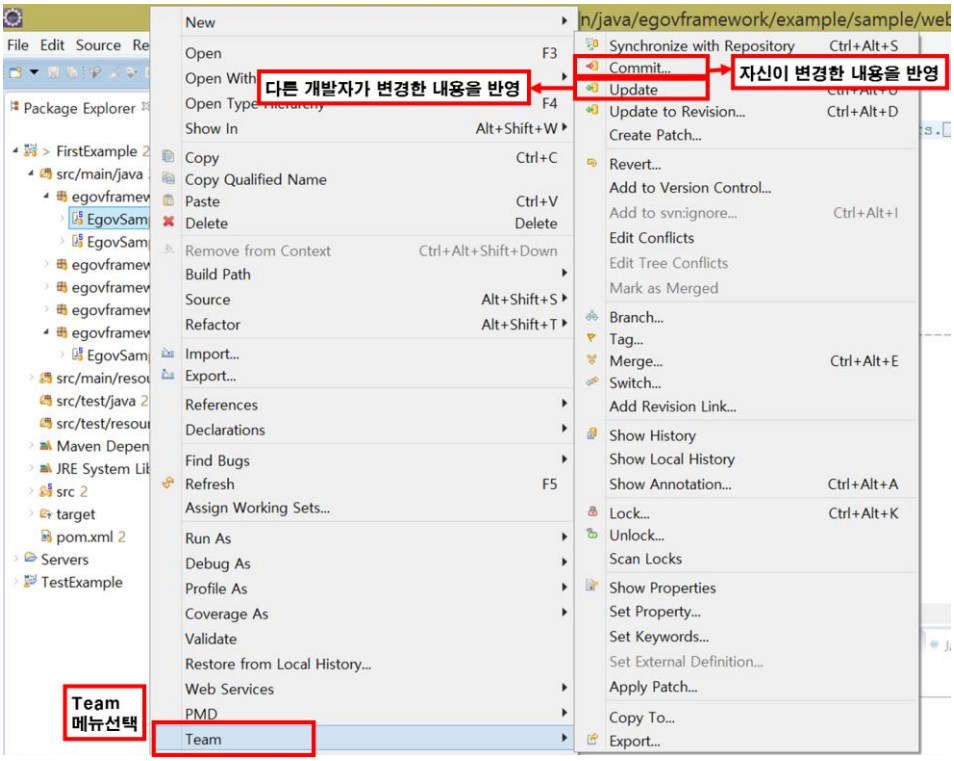
Subversion 에 저장되어 있는 프로젝트를 활용하기 위해서 개발자가 처음으로 해야 되는 일은 Subversion 으로 연결하여 해당 소스코드를 자신의 Workspace 로 내려 받는 것(Check out)이다. Check out 후에는 프로젝트가 Subversion 에 연결을 유지하여 자신이 수정한 내용을 반영(commit)하거나 다른 개발자가 수정한 내용을 내려 받기(update) 할 수 있다. Check out 을 받기 위해 먼저 SVN Repository Exploring 으로 퍼스펙티브를 변경한다. New>Repository

Location 을 선택하면, First commit 에서와 동일하게 Subversion 위치 및 계정 정보를 입력하는 화면이 나오게 된다. 정보를 입력하면 Subversion Repository 가 등록되게 되면 프로젝트를 선택한 후에 마우스 오른쪽 버튼을 눌러서 Check Out 을 선택하게 되면 현재의 workspace 로 해당 프로젝트를 가져온다.



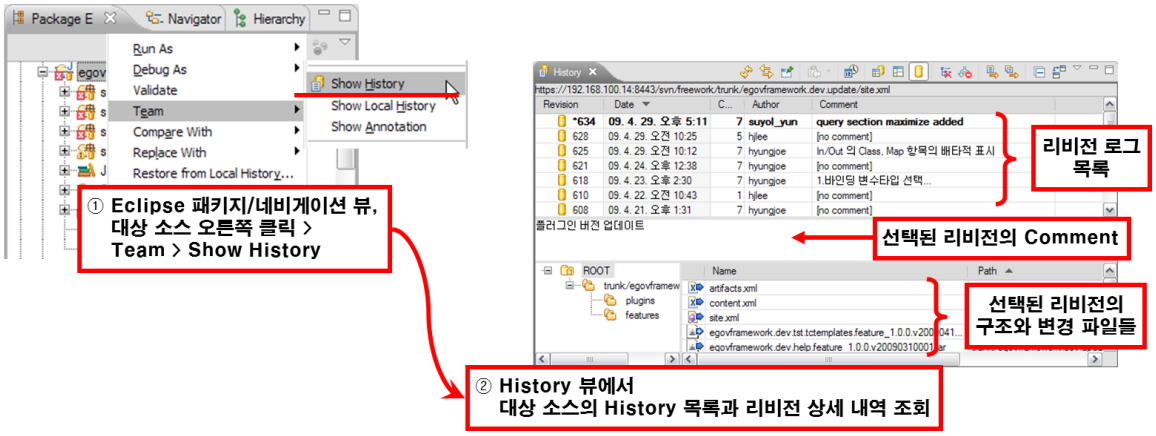
[그림 4-7] Subversion에서 프로젝트 Check out

처음 프로젝트를 share 하거나 Check out 을 한 후에 개발자는 자신이 수정하거나 추가한 소스코드를 Repository 에 commit 하여 변경을 반영할 수 있다. 다른 개발자가 변경한 소스코드의 내용은 update 를 통해서 자신의 소스코드에 변경을 반영하게 된다.



[그림 4-8] Subversion에 commit 또는 update

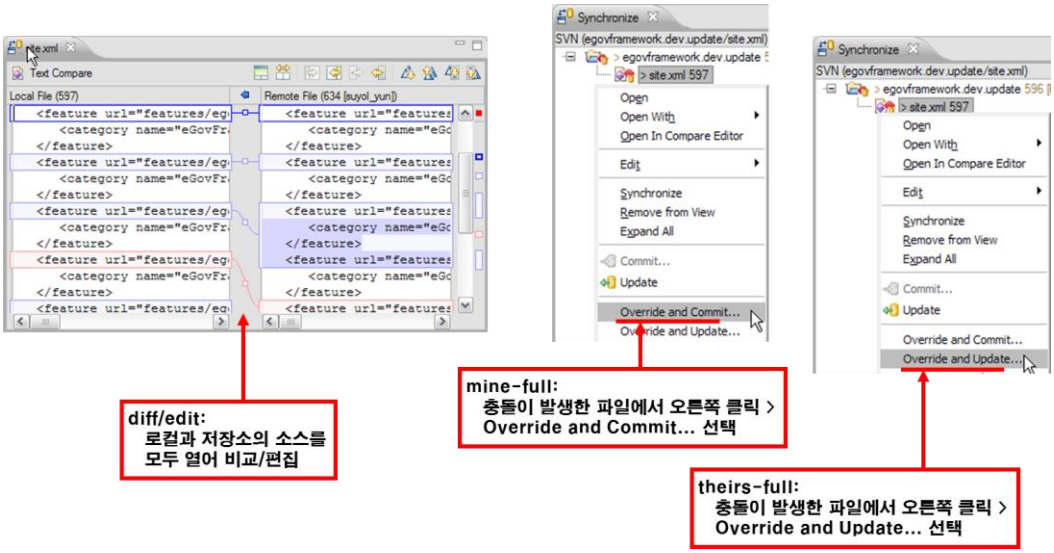
변경된 내역을 조회하기 위해서는 Team>Show History 를 선택하면 변경된 내역을 확인 할 수 있으며, 소스코드를 변경 이전 상태로 복원 할 수 있다.



[그림 4-9] Subversion에서 소스코드 이력조회

Subversion 을 활용하다 보면 종종 충돌이 발생하게 되는 경우가 있다. 하나의 소스코드를 내려 받은 이후에 수정을 한 후 commit 을 하려고 할 때, 다른 개발자가 동일한 소스코드를 변경하여 Repository 에 이미 반영한 경우에 충돌이 발생된다. 양쪽 소스코드 모두 변경이 된 상태이기 때문에 Subversion 은 소스코드 충돌 표시를 한 후 개발자에게 개발자 소스코드를 강제로 commit 할 것인지 변경된 내역을 다시 update 받을 것인지를 선택하도록 한다. 주로

공동작업이 이루어지는 공통모듈의 경우 충돌이 발생될 여지가 많다. 다른 개발자가 변경한 내역을 무시하고 반영할 경우 문제 발생여지가 많기 때문에 충돌의 해결은 개발자간 커뮤니케이션을 통해 변경 내역을 확인 한 후에 Override and Commit 이나 Override and update 로 해결 한다.

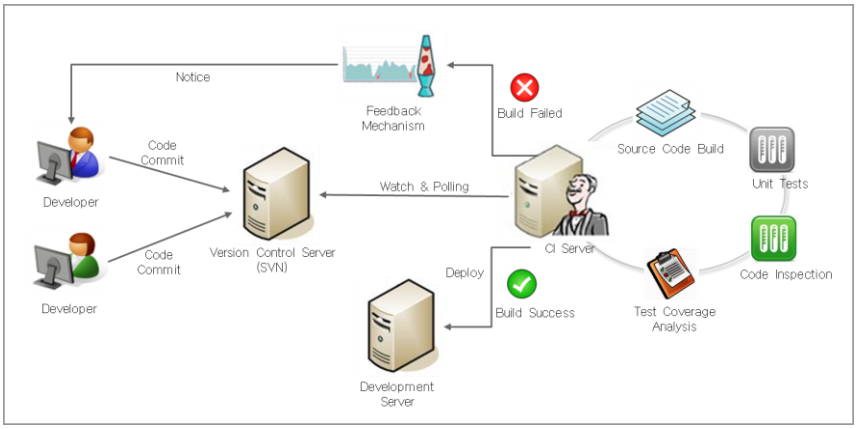


[그림 4-10] Subversion에서 충돌해결

4.2 배포 도구 활용

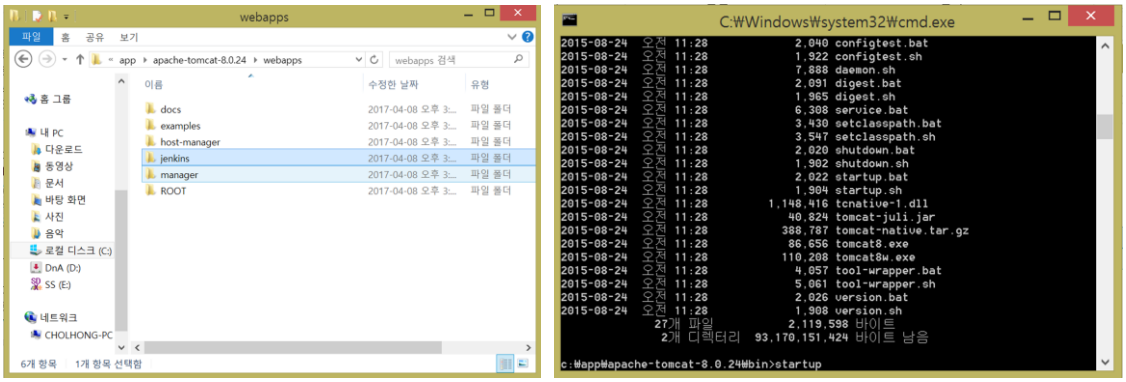
4.2.1 CI(Continuous Integration) 개요 및 설치

전통적으로 SW 개발은 분석, 설계, 개발, 테스트의 순서로 진행이 된다. 일반적인 프로젝트에서 가장 큰 위험 요소는 에러, 미개발, 요구사항 미 충족 등의 결함이 프로젝트 후반 특히 오픈 직전에 발견되는 것이다. 이러한 문제를 해결하기 위해서 소스코드의 통합과 테스트를 상시적으로 수행하는 개념이 CI 이다. 많은 개발자로 구성된 팀이 작업한 것을 짧은 주기로 통합 및 테스트를 수행하는 소프트웨어 개발 방식이다.



[그림 4-11] CI 개념

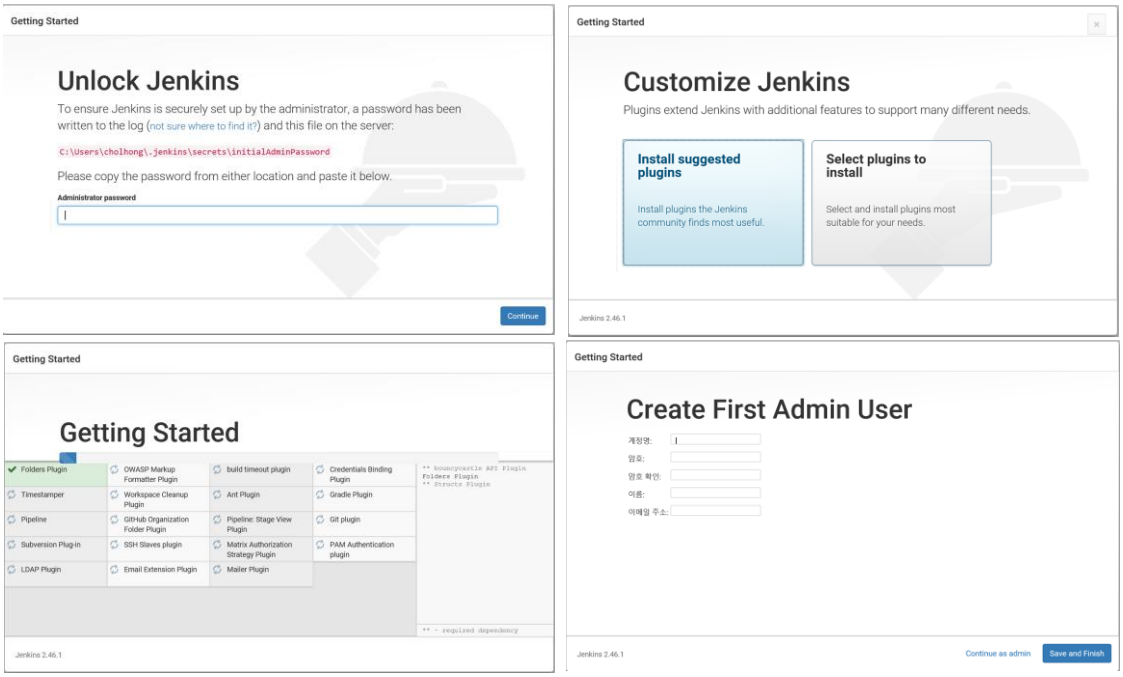
개발자들이 개발한 소스코드를 commit 하면 형상관리 서버에서 통합이 된다. CI 서버는 통합된 소스코드를 자체 디렉터리로 check out 받아 Maven 등의 도구를 활용하여 테스트 및 빌드를 수행한다. 주로 Maven package 를 수행하여 테스트 및 배포파일 생성을 수행하고, 개발서버로 배포하여 개발된 내역을 개발서버에서 확인할 수 있도록 지원한다. 라이브러리의 경우 Maven deploy 를 실행하여 Nexus 등의 원격 Repository 로 배포 할 수 있다. CI 서버로는 Hudson 이 많이 활용되었으나 Sun 이 Oracle 로 흡수 합병에 따라 기존 Hudson 오픈소스 프로젝트에 변경이 생기면서 Jenkins 프로젝트(http://jenkins-ci.org)가 시작 되었다. Hudson 도 Eclipse 에서 오픈소스 프로젝트(http://hudson-ci.org)로 개발이 진행 중에 있으나 Jenkins 보다 업데이트 등이 활발하지 못한 상태이다. 두 개의 CI 서버는 거의 동일한 기능을 가지고 있기 때문에 굳이 구별을 할 필요는 없지만, 사용성 및 향후 발전 측면에서는 Jenkins 가 더 유리한 상태이므로 Jenkins 를 기준으로 설명을 하도록 하겠다. 설치를 하기 위해서 먼저 WAR 형태(또는 zip)의 배포파일을 다운 받는다. 기존에 설치된 Tomcat 등의 WAS 의 webapp 밑에 압축을 풀어 주는 것으로 설치가 완료 된다. Hudson 의 경우도 마찬가지로 설치 할 수 있다. Jenkins 의 경우 WAS 까지 패키징하여 윈도우, 맥 등의 실행파일 형태로 설치할 수 있도록 지원하고 있다. 이 경우 별도 WAS 의 설치가 필요하지 않아 편리한 점도 있으나 윈도우 등 OS 에서 자동 실행되는 서비스로 등록이 되기 때문에 개발 및 테스트 용도로는 관리가 어려울 수 있다. Jenkins 사이트 (http://jenkins-ci.org)에서 다운로드 받은 배포파일을 Tomcat> webapp 에 압축을 풀고 커맨드창에서 Tomcat>bin 디렉터리에서 startup 으로 Tomcat 을 시작한다.



[그림 4-12] Jenkins 설치 및 시작

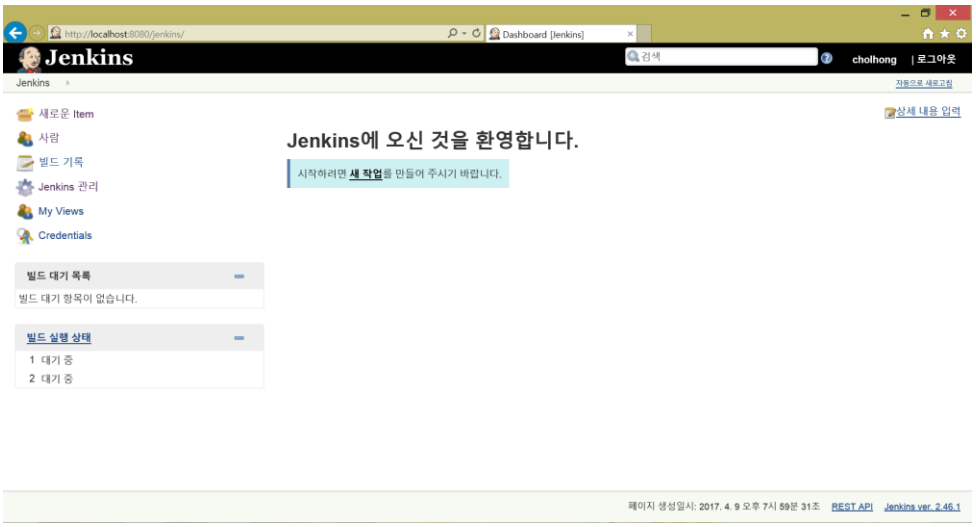
WAS 가 기동되면 설치된 Tomcat 의 포트를 확인(디폴트 8080)하고 주소를 입력하여 웹 브라우저로 접속한다. (http://localhost:8080/jenkins)

맨 처음 접속을 하게 되면 간단한 설정 화면이 나오게 된다. 먼저 Admin 권한을 가진 사용자에 의해 설치가 된 것인지를 확인하기 위해 Password 입력을 요구한다. 경로에 있는 파일을 노트패드 등으로 열어서 Password 를 Copy&Paste 한다. 그 다음 Install Suggested Plugins 를 선택하여 Plugin 을 설치한다. Ant, Maven, Git 등 주요 Plugin 이 설치되는 것을 확인 할 수 있다. 마지막으로 사용자계정을 생성하여 설정이 완료된다.



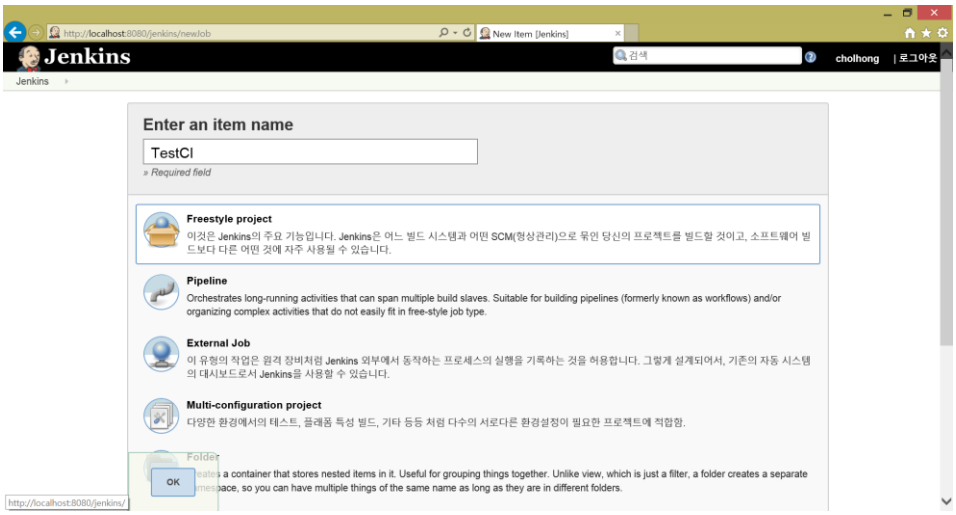
[그림 4-13] Jenkins 설정

설정이 완료 되었으면 다시 접속을 한다. (http://localhost:8080/jenkins) 앞에서 설정한 ID 와 비밀번호로 로그인을 한다.



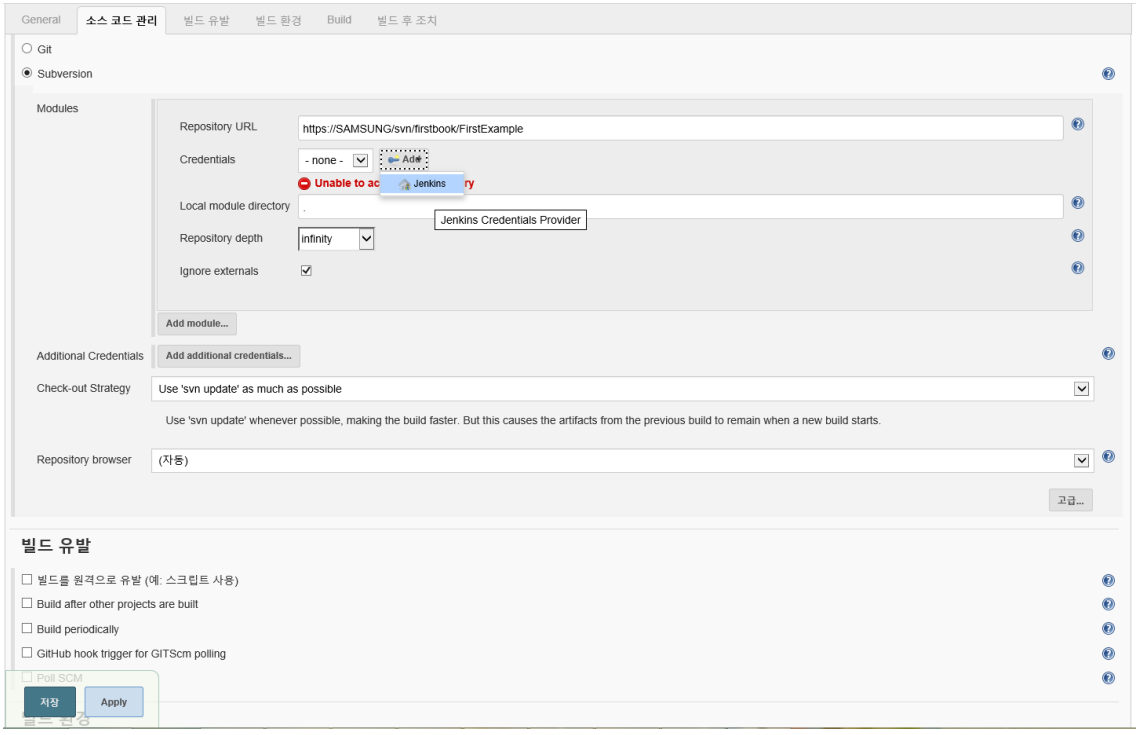
[그림 4-14] Jenkins 시작화면

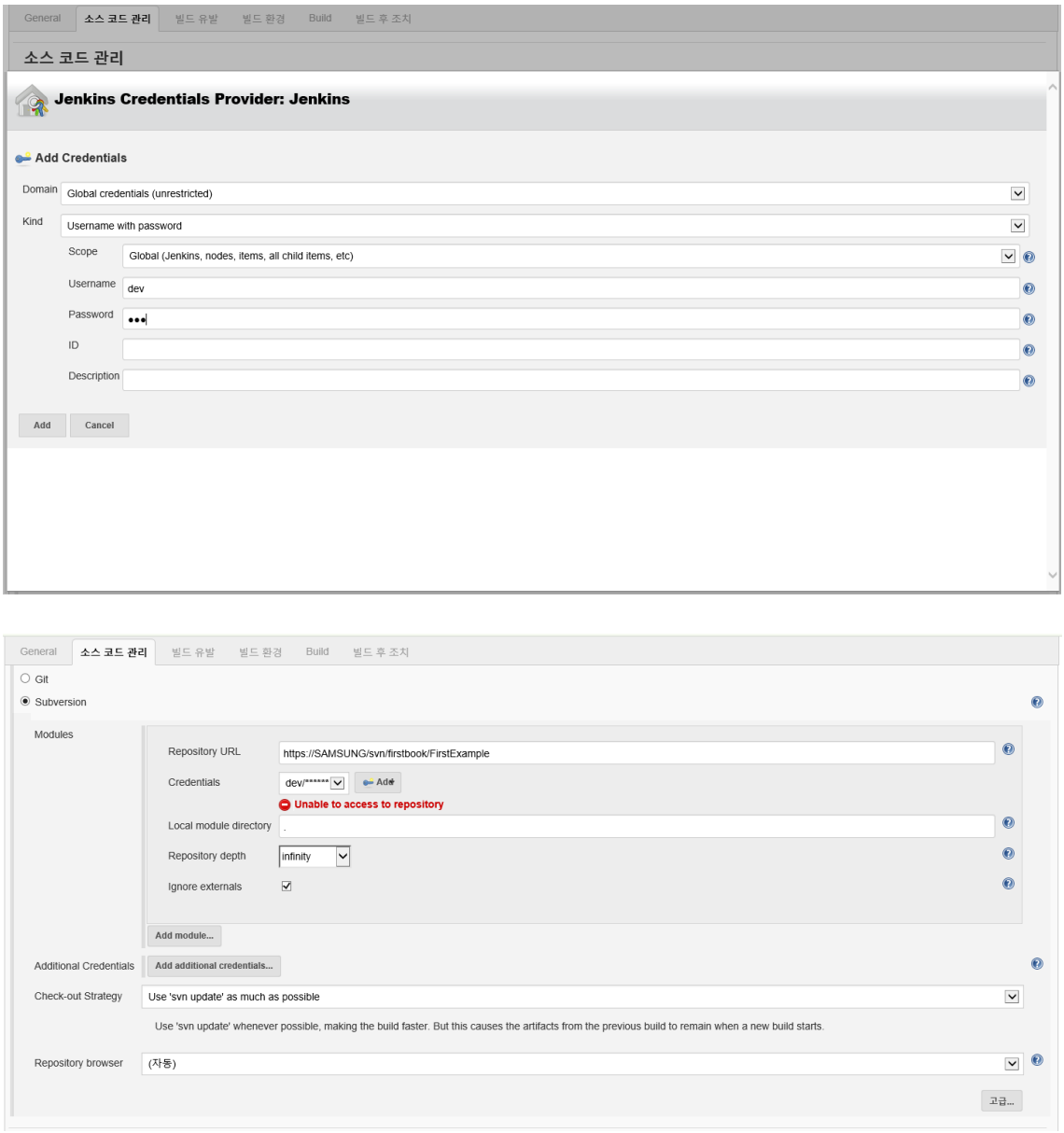
새 작업을 선택하면 빌드 설정을 위한 화면이 보여진다. 작업명을 입력한 다음 Freestyle project 를 선택하고, OK 를 누른다. 작업명은 TestCI 로 입력하도록 하겠다.



[그림 4-15] 작업명 입력 및 작업종류 선택

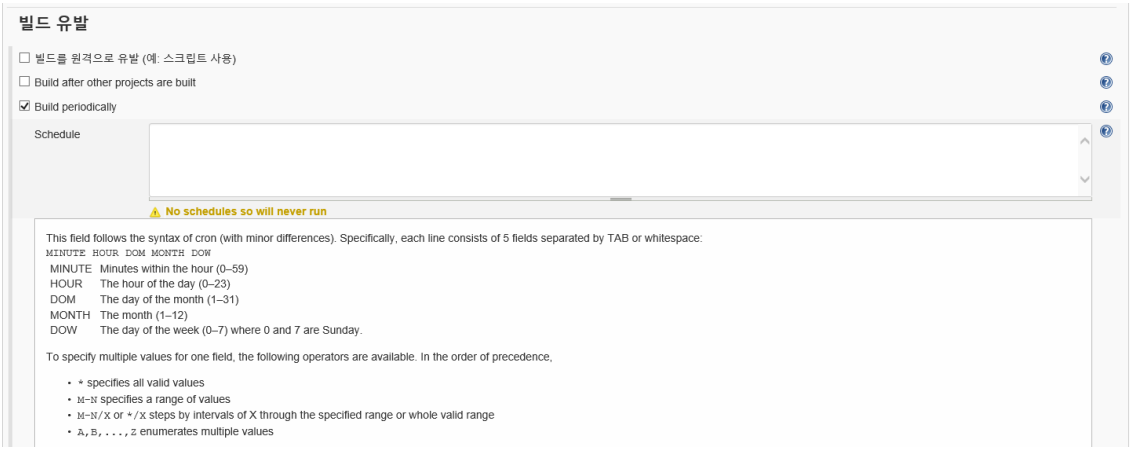
소스코드 관리에서 Subversion 을 선택하고 Repository URL 을 입력한다. 앞에서 Subversion 에서 활용했던 Repository URL 을 입력하고 프로젝트 이름을 입력한다. Eclipse 에서 Subversion 을 활용하기 위해 아이디와 패스워드를 입력했던 것과 마찬가지로 인증을 추가해야 한다. Credential 에서 Add 를 선택하고 Username 을 dev 로 Password 도 dev 를 입력한다. 입력이 된 후에 추가된 Credential 을 선택한다. 화면에서 dev/******로 표시된 내용이다.





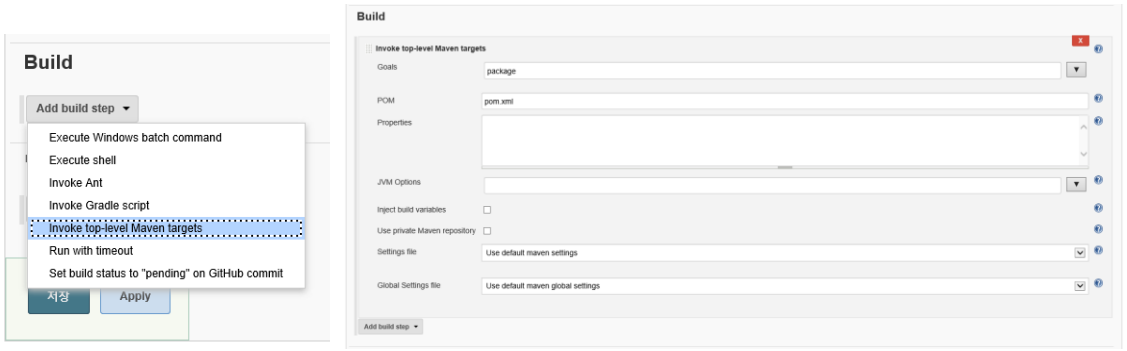
[그림 4-16] Subversion 연동 설정

주기 적인 실행을 위해서는 빌드 유발에서 Build periodically 를 선택하고 Schedule 을 입력한다. MINUTE HOUR DOM(The day of the month) MONTH DOW(The day of the week)의 5 가지 항목으로 구성된다. 예를 들어 하루에 한번 이라면 0 0 * * *의 형태, 매시간 마다의 경우는 0 * * * *의 형태가 된다. 특정 날짜나 특정 요일을 정해서도 실행이 가능하다. 매일 오후 2 시에 실행을 하려면 0 14 * * * 의 형태가 된다.



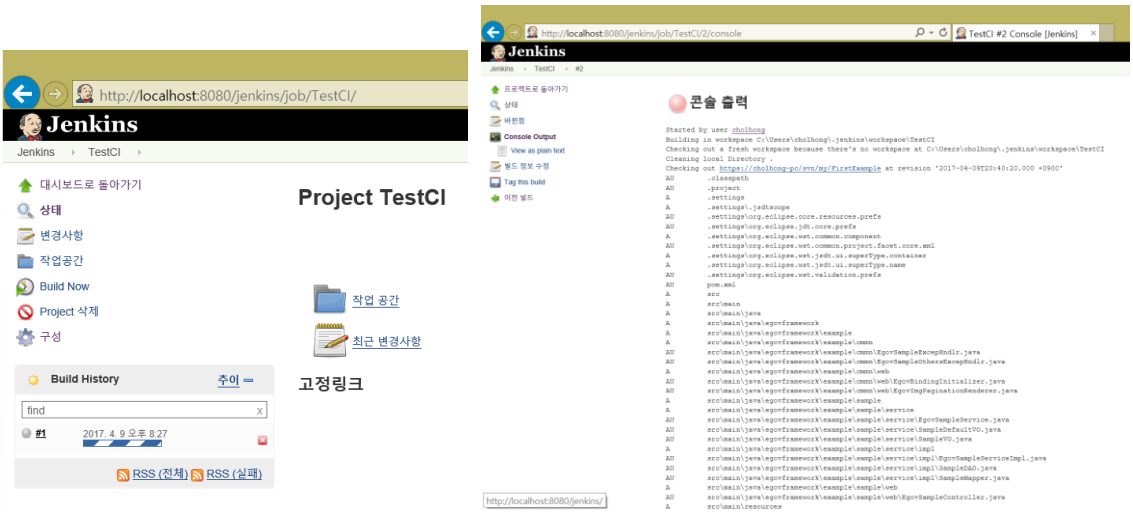
[그림 4-17] 빌드 주기 설정

빌드 설정을 위해서는 Build 탭에서 Invoke top-level Maven targets 를 선택한다. 고급 설정을 눌러서 Goals 는 package 또는 Install 을 선택한다. POM 은 pom.xml 을 입력한다. 이제 저장을 눌러서 빌드 설정을 완료한다.



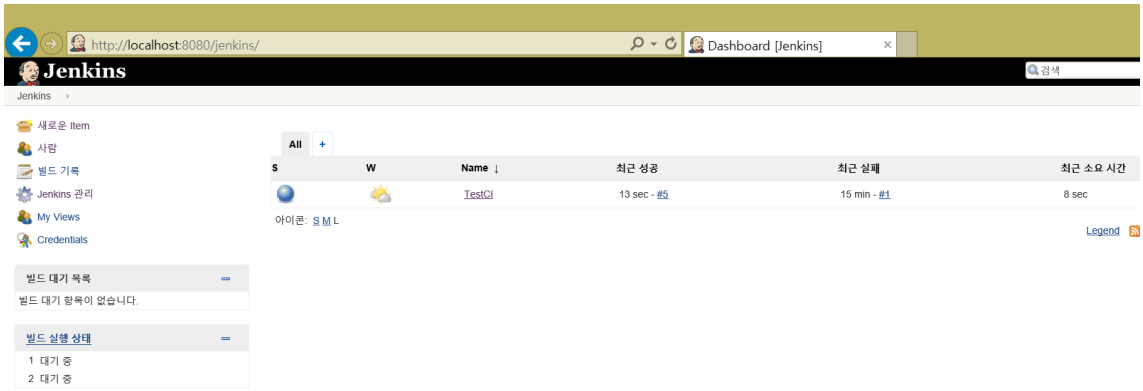
[그림 4-18] Build 설정

Build 를 바로 실행시켜 테스트 하기 위해서 Build Now 를 선택한다. Build History 에서 빌드번호를 선택한다. 빌드 순서에 따라 #1, #2, #3 의 형태로 보여진다. Console Output 을 선택하면 Maven build 의 과정과 결과를 확인 할 수 있다.



[그림 4-19] Build 실행 및 결과확인

대시 보드로 돌아가기를 선택하면 최근 빌드 결과를 보여준다. 빌드가 성공하면 아이콘이 파란색으로 보여지고, 실패하면 빨간색으로 보여진다. 성공횟수가 많아지면 해가 반짝이는 아이콘이 보여지고 실패횟수가 많아지면 구름이 끼고 비가 내리는 아이콘이 보여진다.



[그림 4-20] 대시보드 확인

5. 실행환경 소개

표준프레임워크 실행환경은 소스코드의 재사용성, 유연성을 높이기 위한 표준패턴을 제공하고, 반복적인 구현을 줄일 수 있도록 유용한 유틸리티 기능을 제공한다.

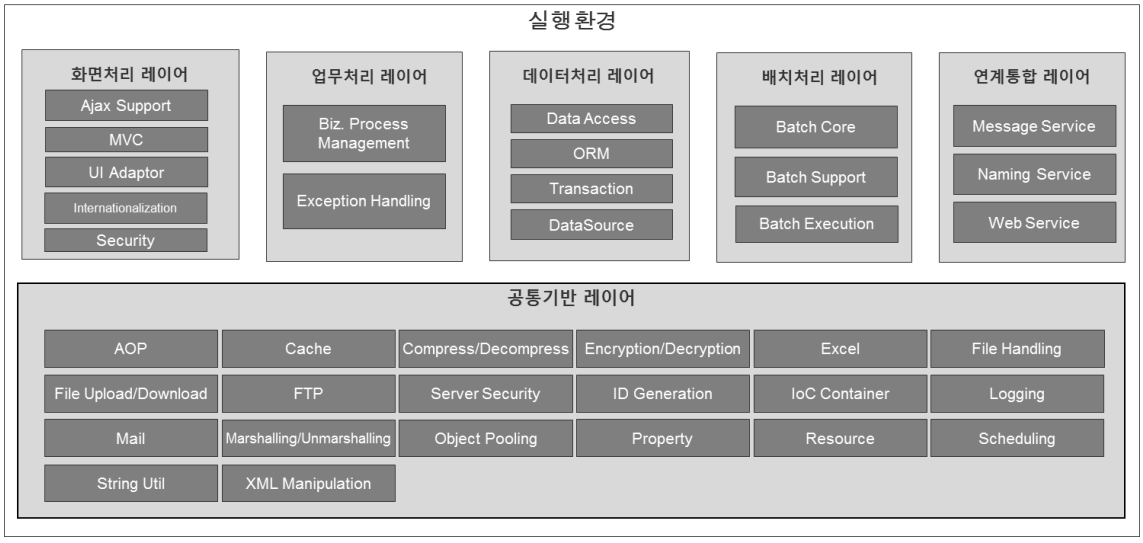
5.1 실행환경 구성

실행환경은 화면처리, 업무처리, 데이터처리, 연계, 공통기반, 배치처리 Layer로 구성된다.

[표 5-1] 실행환경 설명

구분	설명
화면처리 Layer	사용자 인터페이스 및 화면 구현에 필요한 기능과 구조 제공
업무처리 Layer	예외 처리 및 업무흐름 처리 기능을 제공
데이터처리 Layer	개발자가 작성한 소스코드를 통합하여 버전관리 수행
연계 Layer	SOAP기반 웹 서비스 기능 제공
공통기반 Layer	재사용 컴포넌트, 로그 등 개발에 필수적인 기능 제공
배치처리 Layer	일괄처리를 위한 설정 및 실행기능 제공

실행환경 Layer별로 하위 서비스를 제공하고 있다. 화면처리 Layer의 경우에는 기본 패턴인 MVC이외에 국제화, Ajax, 국제화 등 기능을 제공하고 있다. 데이터처리 Layer의 경우에는 DB접속을 위한 DataSource, SQL을 XML설정파일에서 관리할 수 있도록 지원하는 Data Access, 자바코드로 SQL을 대신할 수 있는 ORM 서비스 등으로 구성된다. 공통기반 레이어에는 표준 패턴을 제공하는 IoC, AOP와 Logging 등 서비스를 제공하고 있다.



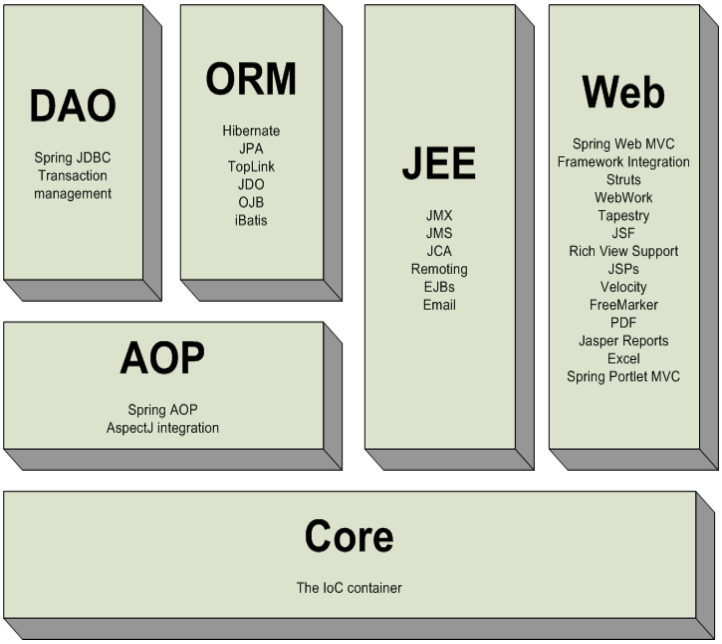
[그림 5-1] 실행환경 구성 서비스

표준프레임워크 실행환경에서 서비스 별로 채택 활용하고 있는 대표적인 오픈소스는 다음과 같다. 개발환경과 마찬가지로 실행환경을 구성하는 오픈소스 라이선스는 대부분 Apache 2.0 기반으로 구성되었다.

[표 5-2] 실행환경 주요 오픈소스

구분	설명
화면처리 Layer	Spring MVC, Apache commons validator, jQuery
업무처리 Layer	Spring
데이터처리 Layer	DBCP, iBatis/MyBatis, Hibernate
연계 Layer	Apache CXF
공통기반 Layer	Spring, Log4j, Spring Security
배치처리 Layer	Spring Batch

표준프레임워크의 주요 패턴과 기능은 Spring 프레임워크를 기반으로 구성된다. Spring 프레임워크는 Java 웹 어플리케이션 개발을 쉽게 해주는 오픈소스 SW 프레임워크이다. 인터페이스와 구현으로 구성된 자바 객체는 Spring 컨테이너에 빈 형태로 등록되어 활용된다. Spring 프레임워크는 IoC 컨테이너로 구성되는 CORE, DB를 지원하는 DAO, JPA 등 ORM, 메시징 등 Enterprise 환경을 지원하는 JEE, 웹 환경을 지원하는 WEB의 패키지로 구성되며, 주요 설명은 아래와 같다.



[그림 5-2] Spring 프레임워크 주요기능

[표 5-3] Spring 프레임워크 주요기능 설명

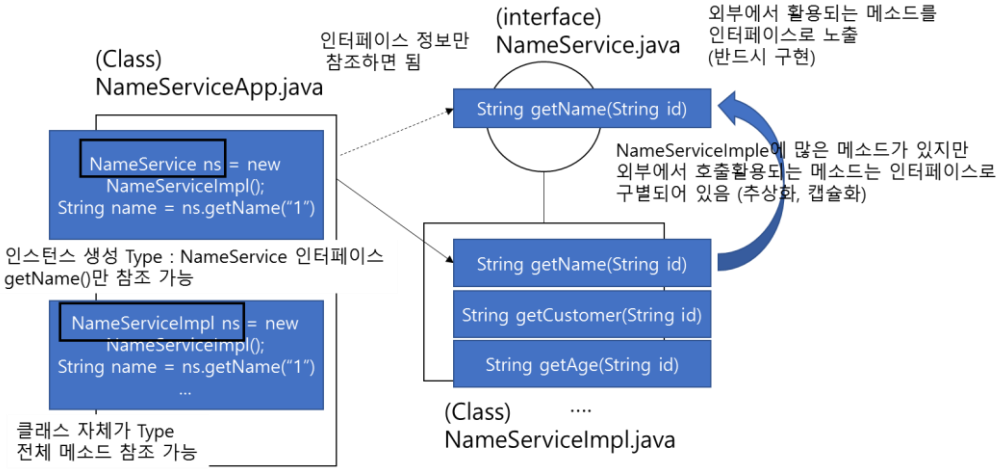
구분	설명
CORE	프레임워크의 가장 기본적인 부분이고 IoC와 의존성 삽입 (Dependency Injection)기능을 제공한다.
DAO	JDBC 코딩과 특정 DB 업체의 에러코드 파싱 등과 같은 작업의 필요성을 제거한 JDBC 추상화 계층을 제공한다. 또한 프로그램적인 트랜잭션 관리 관리 뿐 아니라 선언적인 트랜잭션 관리 기능을 제공한다.
ORM	JPA, JDO, Hibernate와 같은 객체 관계 맵핑(Object-Relational Mapping)을 위한 통합 계층을 제공한다.
AOP	Aspect 지향 프로그램 방식을 지원한다.
WEB	멀티파트 파일업로드 기능, 서블릿 리스너를 사용한 IoC 컨테이너의 초기화, 웹 기반 어플리케이션 컨텍스트 등과 같은 기본적인 웹 기반 통합 기능들을 제공한다.
JEE	Spring 환경에서 다양한 Java EE 기술을 사용할 수 있도록 지원한다.

6. DI (Dependency Injection)

자바프로그래밍 시에 구현된 기능을 다른 소스코드에서 활용 할 때 설정되어 있는 소스코드 간의 종속성 부분은 재사용에 큰 어려움을 주게 된다. 이러한 문제를 해결하기 위해 DI를 활용하여 소스코드간에 직접적으로 종속성을 가지지 않고, 참조 정보를 XML설정파일이나 annotation을 활용하도록 한다. 이를 통해 소스코드를 직접고치지 않고 다른 소스코드를 참조할 수 있도록 지원한다.

6.1 자바 인터페이스 활용

자바 인터페이스는 클래스들이 구현해야 하는 동작을 지정하는데 사용되는 추상형으로, 클래스들이 지켜야 하는 형식을 정의한다. 인터페이스는 클래스에서 외부로 노출되어 활용되는 변수나 메소드 정보를 제공하여 다른 클래스에서 쉽게 참조할 수 있도록 한다. 그리고, 현재의 클래스를 다른 클래스로 대체하기 위해 반드시 구현이 필요한 메소드 정보를 제공한다. 인터페이스에 있는 메소드만 구현이 되면 클래스를 다른 클래스로 교체하여 사용할 수 있다. 일반적으로 컴포넌트를 구성할 때 인터페이스와 클래스를 묶어서 구성하게 된다. 컴포넌트는 특정한 기능을 다른 기능으로 쉽게 대체할 수 있도록 구성이 되어야 하기 때문에 인터페이스 정보는 필수적이다. 외부의 다른 클래스에서 활용가능한 메소드는 인터페이스에 기술되어 있기 때문에 인터페이스에 있는 정보만 참조하면 된다. 인터페이스를 활용하여 클래스 정보의 추상화, 캡슐화가 가능하다. 그림에서 NameServiceImpl 클래스에 많은 메소드들이 있지만 NameService 인터페이스를 Type으로 활용하여 클래스 인스턴스를 생성하게 되면 다른 메소드들은 참조할 수 없고, 인터페이스에 기술된 getName()만 참조가 가능하다. 자바에서 클래스 인스턴스를 생성할 때 별도 인터페이스를 생성하지 않고 클래스 자체를 Type으로 생성 할 수 있으나, 인터페이스를 활용하는 경우 다른 클래스에서 참조 가능한 메소드만 표현하고, 다른 메소드들은 노출시키지 않아도 되기 때문에 활용이 편리하다.



[그림 6-1] 자바 인터페이스 활용

인터페이스는 동작에 관련된 처리 로직 등의 코드가 포함되어 있지 않고, 구현을 위한 메소드명, 입출력 Type 등 형식만 정의 한다. 인터페이스는 다음과 같은 형식으로 작성된다.

```
interface 인터페이스이름 {
        public static final 타입 상수이름 = 값;
        public abstract 메서드이름 (매개변수목록);
}
```

[소스 6-1] 자바 인터페이스 형식

앞의 설명에서 String값의 id를 입력 변수로 가지고, 결과값을 String 값으로 반환하는 getName() 메소드를 가지는 NameService 인터페이스는 다음과 같이 작성 된다.

```
Interface NameService{
  public String getName(String id);
}
```

[소스 6-2] NameService 인터페이스 작성

클래스는 인터페이스 정보를 참조하여 정의되어 있는 메소드는 반드시 구현해야 한다. 만약 구현하지 않으면 에러가 발생된다. NameService 인터페이스를 참조(implements)하여 구현한 NameServiceImpl 클래스는 다음과 같이 작성된다.

```
public class NameServiceImpl implements NameService {
          public String getName(String id) {
          String name = id+ " eGovFrame";
          return name;
          }

          public String getCustomer(String id) {
          return "Customer";
          }

          public String getAge(String id) {
          return "40";
          }
}
```

[소스 6-3] NameServiceImpl 클래스 작성

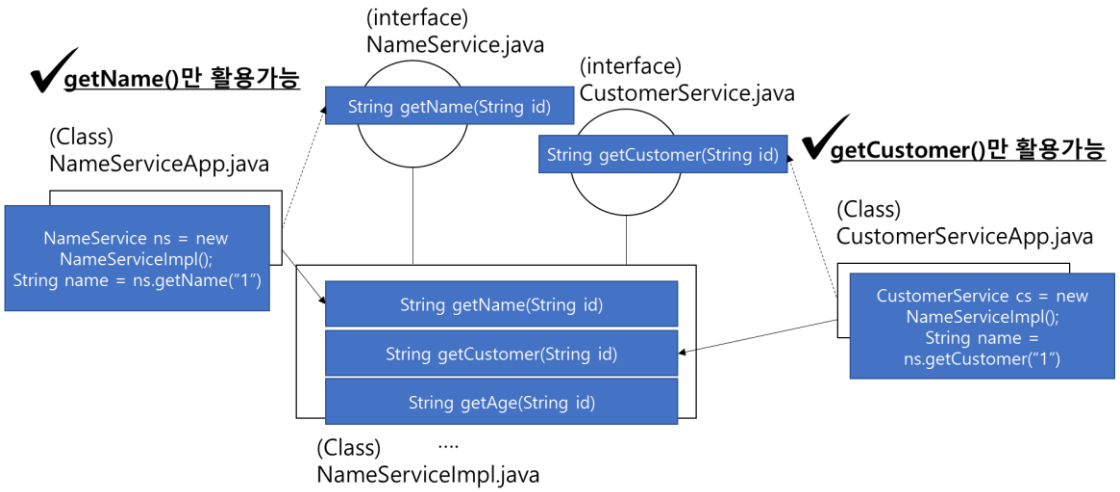
NameServiceImpl의 메소드를 다른 클래스인 NameServiceApp에서 참조하기 위해서 NameService 인터페이스를 Type으로 가지는 클래스 인스턴스와 클래스자체를 Type으로 가지는 클래스 인스턴스를 생성하면 다음과 같다.

```
public class NameServiceApp {
          public static void main (String[] args) {
                    String id = "100";
          // 인터페이스를 Type으로 클래스 인스턴스 생성하는 경우
          // 인터페이스에 정의되어 있는 메소드 getName()만 호출 가능
                    NameService nameservice = new NameServiceImpl();
                    String if_name = nameservice.getName(id);
```

```
                System.out.println("name = "+ if_name);

        // 클래스를 Type으로 클래스 인스턴스 생성하는 경우
        // 모든 메소드 호출 가능
                NameServiceImpl nameserviceimpl = new NameServiceImpl();
                String impl_name = nameserviceimpl.getName(id);
                String impl_customer = nameserviceimpl.geCustoemr(id);
                String impl_age = nameserviceimpl.geAge(id);
        }
}
```

[소스 6-4] NameServiceApp 클래스 작성

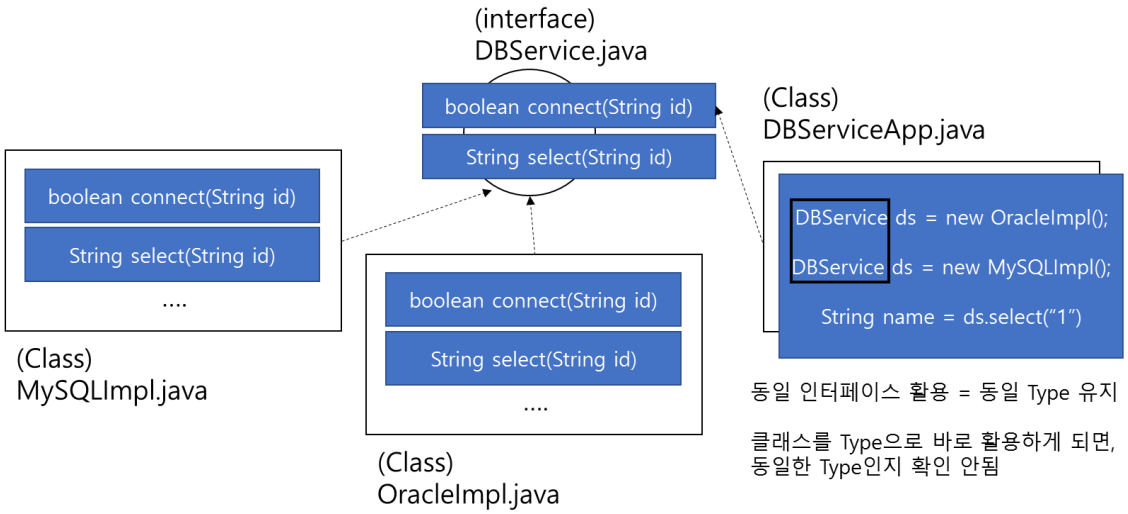
하나의 클래스에 여러 개의 인터페이스를 활용하는 경우 클래스의 메소드를 호출하는 클래스에 따라 다르게 노출시킬 수 있다. 이 경우에는 어떤 인터페이스를 Type으로 활용하여 클래스 인스턴스를 생성하는지에 따라 외부에 어떤 메소드가 노출되는지 결정된다. 그림은 하나의 클래스 NameServiceImpl에 두개의 인터페이스를 활용하는 형태이다. CustomerService 인터페이스에서 getCustomer()만 메소드로 가지고 있기 때문에 CustomerService를 Type으로 생성된 클래스 인스턴스에서는 getCustomer()만 호출하여 활용이 가능하다. 이 경우 Class에서 제공하는 메소드를 호출 활용하는 클래스에 따라 다르게 노출시켜 활용할 수 있도록 캡슐화 하게 된다.



[그림 6-2] 자바 인터페이스를 두 개 활용 사례

자바 인터페이스를 활용하는 다른 사례로는 하나의 인터페이스에 클래스가 다수 존재하는 경우이다. 흔히 컴포넌트를 만드는 가장 중요한 이유로는 현재 구현된 기능을 다른 기능으로 쉽게 교체하여 활용할 수 있도록 하는 것이다. 인터페이스를 같은 것으로 활용하는 클래스가 있다면 기존 코드의 수정 없이 그대로 교체하여 활용 할 수 있다. 컴포넌트 기반의 개발에서는 인터페이스를 잘 설계하고 이를 활용하여 개발하는 것이 중요하다. 인터페이스가 잘 설계되어 있으면 특정한 기능의 모듈을 상황에 따라 교체하여 활용하는 것이 가능해진다.

대표적인 예로 JDBC 드라이버의 경우 Oracle, MySQL 등 활용하는 DB가 달라져도 같은 역할을 수행해야 하므로 인터페이스를 준수하는 것이 필수적이다. 다음과 같이 두 개의 클래스 OracleImpl과 MySQLImpl가 하나의 인터페이스 DBService를 참조하고 있고, DB에 따라 다른 클래스를 활용한다고 가정해보자. DBService 인터페이스에 정의된 두 개의 함수 connect와 select는 반드시 구현이 되어야 한다. 하나의 인터페이스를 참조하고 있기 때문에 두개의 클래스는 서로 교체가 되어도 다른 코드에는 영향을 미치지 않게 된다. 만약 인터페이스를 Type으로 선언하지 않고, 클래스로 Type을 선언하였다면 클래스를 교체할 때 두개의 클래스 Type이 동일해야만 하는지를 먼저 확인해야 한다. 클래스 Type이 일치하는 경우는 클래스 전체 함수나 변수 등이 일치해야 되기 때문에 새로운 동일 Type 클래스를 작성하거나 Type이 같은지를 확인하는 작업은 인터페이스를 활용하는 경우에 비해 매우 어렵다.

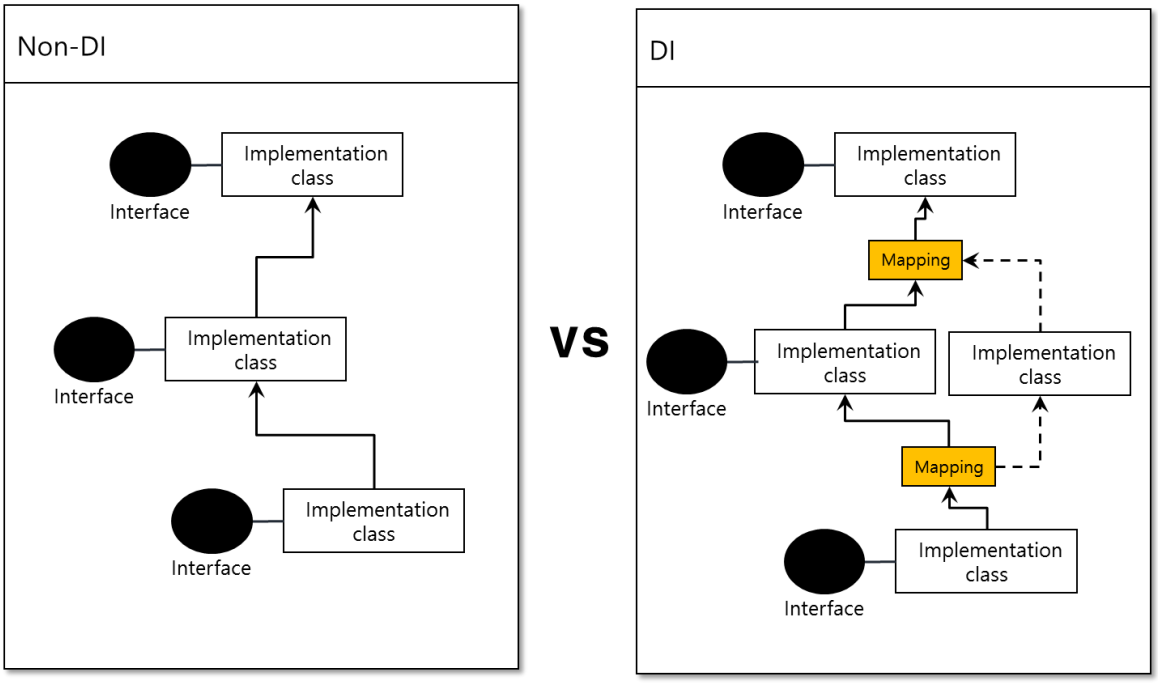


[그림 6-3] 하나의 자바 인터페이스를 두 개 클래스가 공유하는 사례

6.2 DI 개념

인터페이스를 활용하는 경우 동일한 Type을 유지 할 수 있어 컴포넌트 구현과 활용에 유리하다. 특정한 클래스를 활용하다가 비즈니스 로직 변경 필요 등의 이유로 다른 클래스로 교체해야 하는 경우가 발생될 수 있다. 보편적인 방법은 new로 생성된 클래스를 직접 변경하는 것이다. 그러나, 만약 참조하고 있는 클래스가 수십개, 수백개가 되는 경우 굉장히 어려운 작업이 될 것이다. 만약 클래스의 소스코드가 없는 경우는 어떻게 해야 될 것인가? 이러한 문제를 해결할 수 있는 개념이 DI(Dependency Injection)이다. DI는 클래스 인스턴스를 생성할 클래스를 직접 소스코드에 지정하지 않고 인터페이스를 활용하여 Type만 지정한다. XML설정파일이나 annotation을 활용하여 실제 인스턴스를 생성할 클래스를 지정하게 된다. 만약 다른 클래스로 교체를 해야 되는 경우에는 소스코드를 수정할 필요가 없이 XML 설정파일이나 annotation정보를 수정하여 적용이 가능하다. Spring을 활용하지 않는 일반

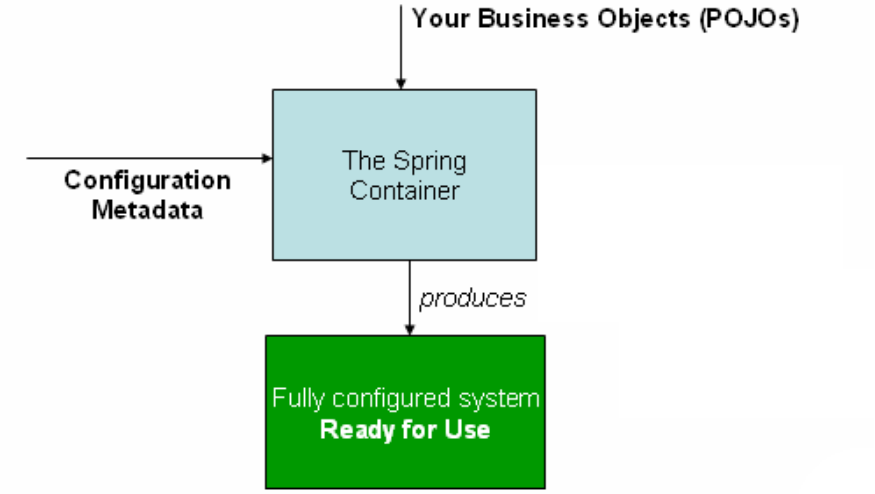
자바환경이라면 컴파일시에는 에러가 발생되지 않으나, 실행시에 Class Not Found 에러가 발생하게 된다. Spring이 동작하면서 설정파일이나 Annotation에 등록된 클래스 정보를 읽어서 동작시점에 Dependency를 생성하게 된다. DI를 활용해서 소스코드에서 new를 해주는 것과 같은 효과를 어플리케이션이 동작하는 시점에 동적으로 생성할 수 있다. 그림에서 DI를 활용하지 않을 때는 new의 형태로 클래스 인스턴스를 생성하게 되어 있어 다른 클래스로 교체하기 위해서는 소스코드 수정이 필요하다. DI를 활용하는 경우에는 XML설정이나 annotation을 변경하여 Mapping정보를 변경하는 형태로 소스코드 변경이 없이 다른 클래스로 교체가 가능하다. 비즈니스 로직 변화가 많은 환경에서 소스코드의 수정없이 변경에 유연 해진다. 클래스가 다른 클래스들 간의 의존성을 소스코드 레벨에서 가지지 않고 설정파일을 변경하여 의존성 변경이 가능해진다. 특정한 기능을 컴포넌트 형태로 만들어서 쉽게 교체해야 되는 경우 효율적으로 활용될 수 있다.



[그림 6-4] DI개념

Spring Container는 객체를 생성하고, 객체 간의 의존성을 이어줄 수 있도록 필요한 정보를 제공한다. 이러한 객체를 Bean이라고 부르며, Spring 프레임워크에서 동작하는 모든 객체는 Bean단위로 활용된다. 어플리케이션이 동작하는 시점에 XML이나 annotation으로 설정된 정보를 Spring Container에 저장하고, 설정 정보를 활용하여 DI를 수행하여 결과적으로 Spring에서 활용가능한 객체 형태인 bean을 생성하게 된다. 소스코드 레벨에서는 클래스가 어떤 클래스를 참조하고 있는지 알 수 없고, 어플리케이션이 동작하는 시점에 설정파일을 통해서 클래스에 대한 의존성이 결정되게 되므로 이를 IoC (Inversion of Control)이라고 한다. 클래스 의존성의 결정이 소스코드에 있지 않고 외부에서 결정되어 활용되는 형태로 일

반적인 어플리케이션이 동작과 반대되는 개념으로 IoC(제어의 역전)라고 불린다.



[그림 6-5] Spring Container 개념

Spring Container의 정보는 ApplicationContext에 의해 관리된다. ApplicationContext는 Bean 생성 및 DI, 생명주기를 관리하는 BeanFactory의 기능과 Spring AOP, 메시지 리소스 처리, 이벤트 처리 등의 기능을 제공한다. ApplicationContext는 BeanFactory의 기능을 모두 제공하므로 일반적으로 ApplicationContext를 활용한다. Spring Container 정보를 생성하고 “foo”라는 bean을 찾는 방법은 다음과 같다. 다수의 Spring 설정파일(Bean 설정, AOP설정, 트랜잭션 설정, Data source 설정 등)을 읽어서 context를 생성하여 활용한다.

```
ApplicationContext context = new ClassPathXmlApplicationContext(
        new String[] {"services.xml", "daos.xml"});
Foo foo = (Foo)context.getBean(“foo”);
```

[소스 6-5] CustomerService 인터페이스 작성

6.3 DI 활용

DI를 설정하기 위해 활용하는 방법으로 XML을 활용하는 방법과 annotation을 활용하는 방법이 있다. XML을 활용하는 방법은 설정이 불편하고 어려운 단점이 있는 반면, XML 파일만 분석하면 모든 클래스들 간의 의존성을 파악할 수 있는 장점이 있다. Annotation의 경우 소스코드에 annotation이 산재되어 있어 의존성 파악은 어려운 점이 있으나, 편리하게 설정 활용할 수 있는 장점이 있다. XML 설정의 경우 Spring 기반으로 구현된 솔루션, 플러그인, 프레임워크 등을 활용할 때 많이 활용되고, 일반 비즈니스 로직 구현은 annotation을 활용하는 것이 일반적이다. 다음의 예제를 통해 DI활용을 설명한다. 서비스 클래스는 비즈니스 로직을 처리하는 클래스이다. DAO는 DB와 연동하여 정보를 처리하는 클래스로 표준프레임워크에서는 MyBatis(이전 버전 iBatis)를 활용하여 DB작업을 수행한다. 본 예제에서는 DI 설명을 위해 String값을 반환하는 간단한 형태로 DAO를 구성하였다. DI설정파일 context-customer.xml과 annotation 설정 파일 context-common.xml을 활용한다.

[표 6-1] DI활용 예제

역할		파일명
인터페이스		CustomerService.java
XML 방식	서비스 클래스	CustomerXMLServiceImpl.java
	DAO 클래스	CustomerXMLDAO.java
	DI설정	context-customer.xml
Annotation 방식	서비스 클래스	CustomerAnnotationServiceImpl.java
	DAO 클래스	CustomerAnnotationDAO.java
	Annotation 설정	context-common.xml
어플리케이션		CustomerSerivceApp.java
테스트코드 (JUnit)		CustomerServiceTest.java

비즈니스 로직인 웹 어플리케이션의 경우 Controller클래스에서 호출 된다. 본 예제에서는 일반 자바 어플리케이션 CustomerServiceApp클래스에서 호출하도록 하였다. 제일 먼저 인터페이스를 구현해야 한다. CustomerService 인터페이스는 XML이나 annotation 방식에 상관 없이 구현 클래스에 공동 활용되며, String을 입력 값을 전달 받고 String으로 결과 값을 반환하는 두 개의 메소드getCustName()과 getCustGrade()를 가진다.

```
public interface CustomerService {
        String getCustName(String id);
        String getCustGrade(String id);
}
```

[소스 6-6] CustomerService 인터페이스 작성

6.3.1 XML 설정을 활용한 DI

XML 설정파일은 <beans/>를 root로 가지며, <bean/>을 사용하여 의존성을 설정한다. Spring Container에 여러 설정파일을 읽어 활용할 수 있기 때문에 설정 내용이 너무 많은 경우에는 XML설정을 여러 개의 파일로 구성한다. 또한, DI, AOP, 트랜잭션, 메시지 등 설정 내용에 따라 개별 파일을 구성하는 것이 일반적이다. 다음의 XML설정(context-customer.xml)은 두 개의 bean을 설정하고 있다. Bean은 id를 가지고 실제 클래스를 Mapping 하고 있다. 클래스 정보는 Spring Container에 id와 함께 저장되어 어플리케이션이나 다른 bean에서 bean의 id로 해당 클래스 정보를 요청할 때 정보를 제공하게 된다. customerXML의 id를 가지는 bean은 실제 lab.CustomerXMLServiceImpl 클래스이며, property속성(custXMLDAO)이 정의 되어 bean id가 customerXMLDAO bean을 레퍼런스(ref)하고 있으며, 서비스 클래스에서 DAO클래스를 참조하기 위한 DI설정에 활용된다. 레퍼런스 되는 Bean은 id가 custoemrXMLDAO이고, 클래스는 lab.CustomerXMLDAO로 Mapping 되어 있다. Property이름은 클래스에서 클래스 정보를 받아오기 위해 정의하는 setter method에서 첫 글자를 대문자로 바꾸어 메소드 명으로 활용된다. 설정에서 customerXML의 경우 CustomerXMLServiceImpl 클래스의 setter 메소드 명으로 setCustXMLDAO으로 정의한다.

```
<?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-4.0.xsd">

        <bean id="customerXML" class="lab.CustomerXMLServiceImpl">
            <property name="custXMLDAO" ref="customerXMLDAO" />
        </bean>
                                        Bean을 레퍼런스 (이름 동일)
        <bean id="customerXMLDAO" class="lab.CustomerXMLDAO"/>

</beans>
```

[소스 6-7] XML DI설정 context-customer.xml

XML로 DI를 설정하는 클래스 CustomerXMLServiceImpl은 인터페이스를 참조하여 구현되며, 인터페이스에 설정된 두 개의 메소드 getCustName과 getCustGrade를 반드시 구현해야 한다. DAO를 호출하여 전달 받은 결과 값을 반환하도록 구현 되었다.
private CustoemrXMLDAO xmlDAO; 로 Type만 선언되어 있고, new로 인스턴스 생성이 되지 않았는데 xmlDAO.getCustName(id)의 형태로 메소드를 호출하고 있다. Spring을 쓰지 않는 환경이라면 실행시점에 Class Not Found 에러가 발생되는데, DI를 활용하여 설정에 있는 클래스 정보를 setter method에서 치환하여 new를 실행한 것과 같이 인스턴스를 생성하게 된다. setCustXMLDAO에서 cmlDAO의 변수로 XML설정파일에 property로 설정 되어 있는 클래스 정보 (lab.CustomerXMLDAO)를 읽어와서 매핑하게 된다.

```
public class CustomerXMLServiceImpl implements CustomerService {
                                                    인터페이스 참조 구현
        private CustomerXMLDAO xmlDAO;          Type만 정의되고 new는 없음

        public void setCustXMLDAO(CustomerXMLDAO cxmlDAO) {
                this.xmlDAO = cxmlDAO;
        }                                       설정파일의 property custXMLDAO
                                                에서 앞자를 대문자로 바꾸어
        public String getCustName(String id){   setCustXMLDAO로 정의
                return xmlDAO.getCustName(id);  cxmlDAO에 클래스 정보를 전달받아
        }                                       xmlDAO에 치환하여 DI수행

        public String getCustGrade(String id) {
                return xmlDAO.getCustGrade(id);
        }
}
```

[소스 6-8] CustomerXMLServiceImpl 서비스 클래스 작성

DAO는 서비스에서 호출되어 전달 받은 문자 값에 “ eGovFrame XML”과 “ S XML”을 추가

하여 반환하도록 되어 있다. XML설정에는 bean id가 customerXMLDAO로 설정되어 있고, 실제 클래스는 lab.CustomerXMLDAO이며 DI를 위해 서비스 클래스에 property 속성으로 레퍼런스 되어 있다.

```
public class CustomerXMLDAO {

	public String getCustName(String id) {
		return id + " eGovFrame XML";
	}

	public String getCustGrade(String id) {
		return id + " S XML";
	}
}
```

[소스 6-9] CustomerXMLDAO 클래스 작성

6.3.2 Annotation 설정을 활용한 DI

Annotation은 소스코드에 직접 코딩을 하는 것이 아니라 @의 형태로 메타데이터를 삽입하는 것을 의미한다. 비즈니스로 로직에는 영향을 주지 않고 소스코드에 대한 실행이나 설정 정보를 추가한다. Annotation을 활용하여 DI를 설정하기 위해서는 Spring에서 annotation 활용을 위한 component-scan 정보를 설정해주어야 한다. Spring Container가 실행 될 때 base package 하위 소스코드의 Annotation 정보를 읽어서 Spring Container에 저장한다.
아래 설정(context-common.xml)에서는 패키지 lab 이하의 모든 소스코드를 스캔하여 annotation 정보를 획득하여 Spring Container에 저장하게 된다.

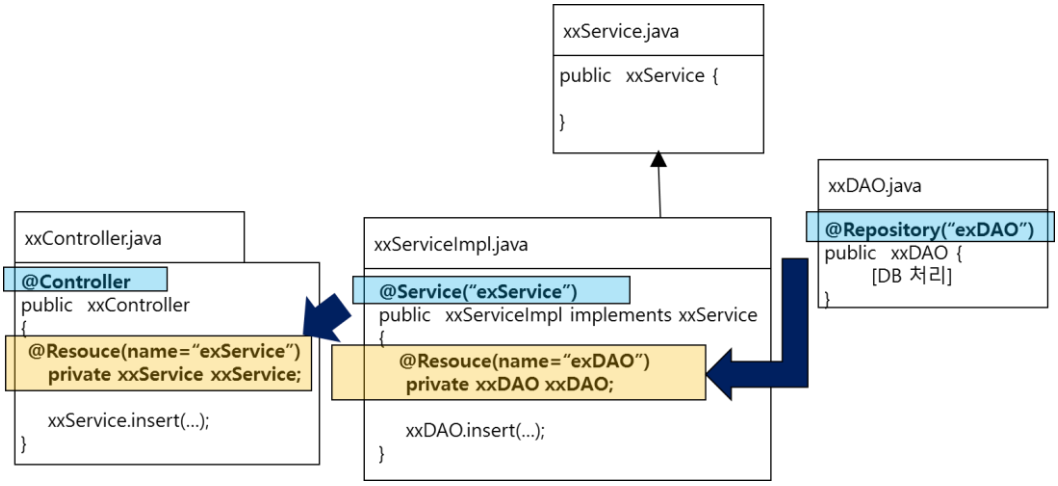
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
	    http://www.springframework.org/schema/context
	    http://www.springframework.org/schema/context/spring-context-4.0.xsd">

	<context:component-scan base-package="lab" />

</beans>
```

소스코드 스캔할 상위 패키지 지정

[소스 6-10] context-common.xml 작성

Bean을 설정하기 위해 클래스 위에 역할별로 annotation을 기술한다. 비즈니스 로직을 구현하는 서비스 클래스는 @Service, DAO는 @Repository로 정의 한다. MVC에서 소개하게 되는 Controller 클래스에는 @Controller를 정의한다. Bean 이름은 @Service("customer")와 같이 정의하며, 이름이 지정되지 않으면 클래스 이름의 첫 글자를 소문자로 바꾸어 자동으로 지정한다. 설정된 bean 정보를 활용하여 DI를 설정하기 위해서는 @Resource나

@Autowired를 활용한다. Private CustomerAnnotationDAO customerdao;의 형태로 Type이 지정되어 있으나 new로 인스턴스를 생성하지 않고, @Resource(name=”custoerdao”)의 형태로 DI를 수행하였다. @Resource의 경우에는 bean의 이름을 지정하여 DI가 가능하며, @Autowired의 경우에는 같은 Type (Interface)를 활용하는 class를 자동으로 매핑하여 DI를 수행한다. 같은 Type이 다수 존재하는 경우는 에러가 발생되므로 유의하여야 한다.
XML설정과 annotation설정을 통한 DI는 서로 교차도 가능하다. 즉 XML로 설정된 bean을 annotation을 활용하여 DI를 설정할 수 있고, 반대로 annotation으로 설정된 bean을 XML을 활용하여 DI 설정할 수 있다. Annotation을 활용한 DI 설정은 다음 그림과 같다. @Controller, @Service, @Repository로 bean이 설정되어 있고, 이를 DI로 활용하기 위해 @Resource로 참조 설정 되었다.



[그림 6-6] Annotation을 활용한 DI 설정

@Service를 활용하여 customer 이름으로 bean이 설정되어 있다. Customerdao 이름으로 설정된 DAO bean을 활용하기 위해 @Resource로 참조 설정 하였다.

```
@Service("customer")                    customer 서비스 bean 설정
public class CustomerAnnotationServiceImpl implements CustomerService {

        @Resource(name="customerdao")
        private CustomerAnnotationDAO customerdao;

        public String getCustName(String id) {
                return customerdao.getCustName(id);
        }

        public String getCustGrade(String id) {
                return customerdao.getCustGrade(id);
        }
}
```

Type만 정의되고 new 없음
customerdao 이름의 bean을
@Resource를 통해 DI 수행

[소스 6-11] CustomerXMLDAO 클래스 작성

@Repository를 활용하여 customerdao 이름으로 bean이 설정되어 있다. 서비스 클래스에서 @Resource로 참조 설정되고 메소드 호출 된다.

```
@Repository("customerdao")        customerdao DAO bean 설정
public class CustomerAnnotationDAO implements CustomerService {

        public String getCustName(String id) {
                return id+" eGovFrame Annotation";
        }

        public String getCustGrade(String id) {
                return id +  " S Annotation";
        }
}
```

[소스 6-12] CustomerXMLDAO 클래스 작성

6.3.3 Client 어플리케이션 실행

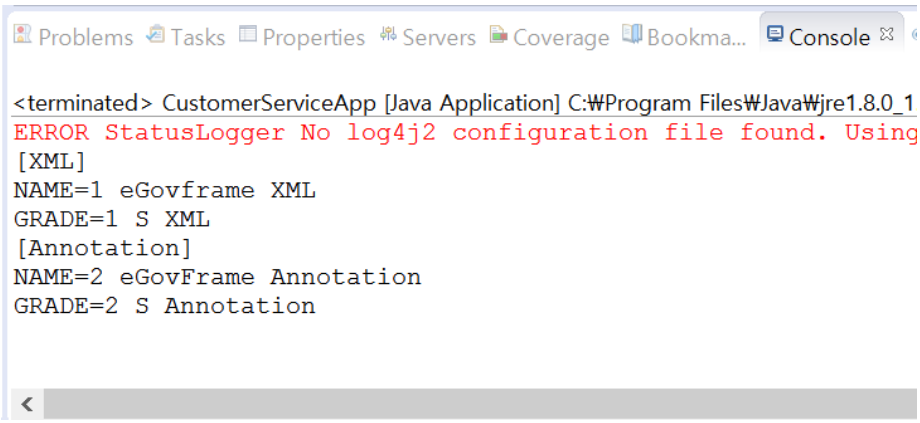
CustomerXMLServiceImpl이 동작하는지 확인하기 위해서 getCustName과 getCustGrade를 호출하는 자바 어플리케이션을 작성 한다. 자바 어플리케이션은 만들어진 서비스를 활용하는 Client 역할을 수행하기 때문에 XML이나 Annotation을 통한 DI 설정을 하지 않고, Spring Container에서 bean을 검색하여 정보를 얻어오는 형태로 활용하게 된다. XML이나 Annotation을 활용하여 DI설정을 하는 경우는 클래스간 연관이 높으면서 변경 가능성이 있는 경우에 변경 유연성을 높이기 위해서이다. 다른 클래스에서 기능만 호출하여 사용하는 경우는 연관성을 없애는 것이 중요하다. 이러한 경우 CustomerServiceApp 클래스 에서와 같이 Spring Container에서 bean id로 검색하고 bean 정보를 얻어오게 지원하면 기능 제공자와 사용자 간의 loosely coupling 된 형태로 의존성을 없애고 유연하게 활용할 수 있다.

```
public class CustomerServiceApp {
        public static void main(String[] args) {
                String configLocation = "classpath*:META-INF/spring/context-*.xml";
                                        Spring Container에 읽어들일 설정파일 위치
                ApplicationContext context
                                = new ClassPathXmlApplicationContext(configLocation);
                                        설정파일 이용하여 Application Context 생성
                                        annotation으로 설정된 정보도 함께 생성
                CustomerService CustomerXML
                                =(CustomerService)context.getBean("customerXML");
                                        bean이름 “customerXML”검색, 의존성 생성
                System.out.println("[XML]");
                System.out.println("NAME="+ CustomerXML.getCustName("1"));
```

```
                System.out.println("GRADE="+ CustomerXML.getCustGrade("1"));


                CustomerService CustomerAnnotation
                        = (CustomerService)context.getBean("customer");
                                        bean이름 "customer"검색, 의존성 생성

                System.out.println("[Annotation]");
                System.out.println("NAME="+ CustomerAnnotation.getCustName("2"));
                System.out.println("GRADE="+ CustomerAnnotation.getCustGrade("2"));

        }
}
```

[소스 6-13] CustomerServiceApp 클래스 작성

어플리케이션을 실행하면 다음과 같이 결과가 출력되는 것을 확인할 수 있다. XML에서는 id를 "1"로 전달 했고, annotation에서는 id를 "2"로 전달하여 서비스 클래스와 DAO 클래스를 거쳐서 결과 값을 전달 받았음을 확인 할 수 있다. 빨간색으로 에러 표시가 되는 부분은 로그를 생성할 log4j파일이 없어서 발생되는 것으로 제7장에서 학습하게 되며, 실행에는 지장이 없다.



[그림 6-7] 어플리케이션 실행결과

6.3.4 Test 코드(JUnit) 실행

단위테스트는 소스코드의 중요 로직의 메소드를 실행시켜서 결과를 확인 할 수 있도록 한다. 메소드의 실행결과가 특정한 값으로 예상되는 경우 일치하는지 여부를 확인하여 일치하면 테스트가 Pass 되어 아무런 메시지가 출력되지 않지만, 일치하지 않는 경우는 에러메시지를 출력하게 된다. 이를 위해 표준프레임워크에서는 JUnit을 활용하여 테스트 케이스를 작성하여 실행한다. 먼저, spring-test와 JUnit 라이브러리 설정을 확인하여야 한다. pom.xml에 다음과 같이 의존성 설정이 되어 있는지 확인한다.

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.0.9.RELEASE</version>
        <scope>test</scope>
        <exclusions>
                <exclusion>
                        <artifactId>commons-logging</artifactId>
                        <groupId>commons-logging</groupId>
                </exclusion>
        </exclusions>
</dependency>
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
</dependency>
```

표준프레임워크 3.6.0의 경우 4.1.2.RELEASE

[소스 6-14] JUnit활용을 위한 pom.xml 설정

Maven 기반 디렉터리 구조에서 테스트를 위해 src/test/java와 src/test/resources가 정의되어 있다. Test 코드는 src/test/java 밑에 위치하게 되며, 테스트에서 활용되는 설정파일은 src/test/resources 밑에 정의되어야 한다. 따라서 src/main/resources에 있는 설정파일을 src/test/resources 밑으로 copy해야 한다. Spring 프레임워크에서 JUnit을 지원하기 위해 Spring Container를 실행할 수 있도록 annotation 기반 기능을 제공하고 있다.
@RunWith은 Spring 기반으로 JUnit을 실행하도록 Runner를 지정하는 역할을 하고 있다. @ContextConfiguration은 경로에 해당되는 설정파일을 읽어서 Spring Container를 생성하여 JUnit 테스트 프로그램에서 접근하여 활용될 수 있도록 한다. 테스트케이스들은 @Test의 형식으로 설정되어 메소드를 직접 호출하지 않아도 JUnit이 실행될 때 자동으로 실행된다. 테스트 결과를 판별하기 위해 수행이 정상적일 때 결과값과 실제 수행된 결과값을 비교하여 두 개가 같을 경우에는 아무런 동작도 하지 않으나, 다를 경우 테스트 실패라는 에러메시지를 출력하게 된다. assertEquals의 경우 예상 결과 값과 실제 수행된 결과 값을 비교하여 두 값이 다를 경우 에러메시지를 출력한다. 이외에도 assertNotEquals, assertTrue, assertFalse, assertNull, assertNotNull 등 많은 메소드를 제공하고 있다. 테스트를 수행하기 전후에 데이터의 준비, 결과값 처리 등 필요한 작업이 있을 경우 테스트케이스 수행 전후에 자동으로 실행 될 수 있도록 @BeforeClass, @AfterCalss, @Before, @After의 annotation을 제공하고 있다. JUnit에서 활용되는 annotation들은 다음 표와 같다.

[표 6-3] JUnit annotation

구분	Annotation	설명
테스트 환경 설정	@RunWith	Spring 활용 Runner 지정 (SpringJUnit)
	@ContextConfiguration	Spring Container 생성
테스트케이스	@Before	테스트케이스 실행 전에 매번 실행 됨

실행 전 수행	@BeforeClass	테스트케이스 실행 전에 한 번만 실행 됨
테스트케이스 실행 후 수행	@After	테스트케이스 실행 후에 매번 실행 됨
	@AfterClass	테스트케이스 실행 후에 한 번만 실행 됨
테스트케이스	@Test	테스트케이스 실행

앞에서 어플리케이션으로 실행되었던 Customer서비스 실행을 JUnit으로 실행하게 되면 다음과 같다. @ContextConfiguration에서는 test 환경에서 활용될 설정 파일 위치를 지정하게 되며, src/test/resources 밑의 설정 정보를 읽어오게 된다. JUnit이 실행되게 되면 @Test로 설정되어 있는 테스트케이스가 실행되게 되며, assertEquals가 실행되어 예상 값과 결과 값을 비교하여 결과를 출력하게 된다.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {classpath*:META-INF/spring/context-*.xml"})
public class CustomerServicetest {
        @Resource(name="customerxml")
        CustomerXMLServiceImpl xmlcustomer;

        @Resource(name="customer")
        CustomerAnnotationServiceImpl customer;

        @Test
        public void testMain() {
            assertEquals(customer.getCustName("1"),"1 eGovFrame Annotation");
            assertEquals(customer.getCustGrade("1"),"1 S Annotation");

            assertEquals(xmlcustomer.getCustName("1"),"1 eGovFrame XML");
            assertEquals(xmlcustomer.getCustGrade("1"),"1 S XML");     }
}
```

spring에서 JUnit을 활용하기 위해 runner설정
spring container를 로드하도록 설정

XML 방식 DI 적용

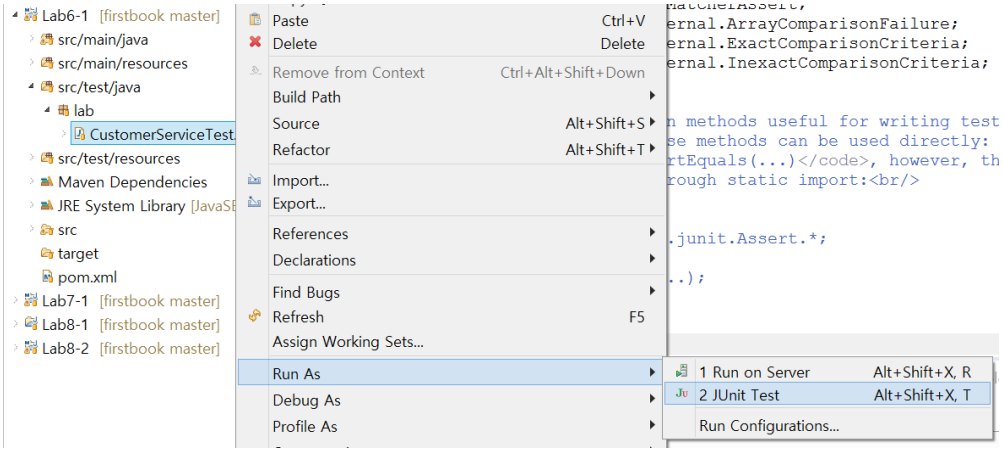
Annotation 방식 DI 적용

테스트 케이스 작성

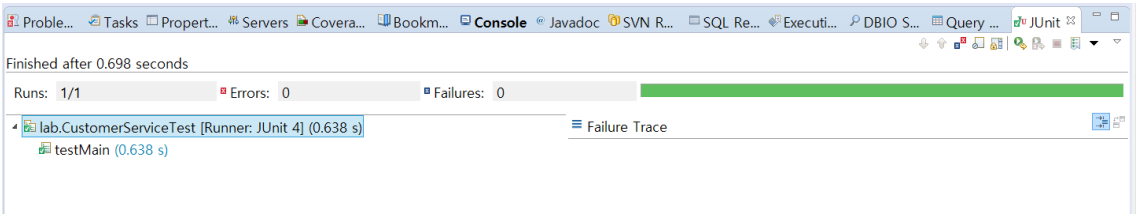
테스트 예상 값과 실제 결과값을 비교

[소스 6-15] JUnit활용 테스트코드 CustomerServiceTest 클래스

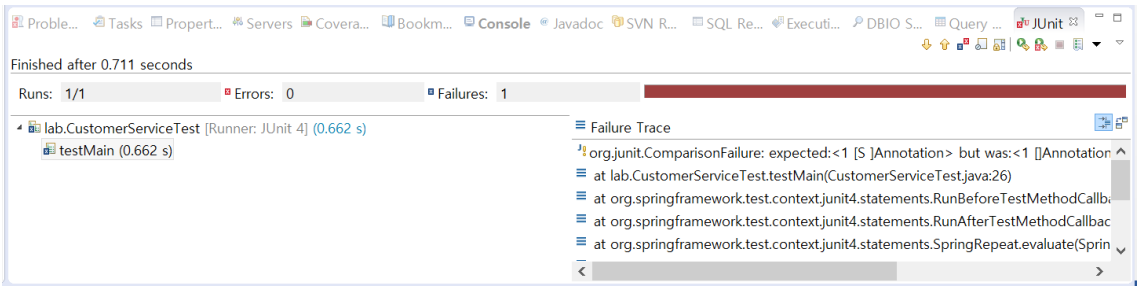
테스트 코드의 실행은 CustomerServiceTest.java를 선택하고 마우스 오른쪽 버튼을 눌러서 Run As > JUnit Test를 선택하여 실행한다. 테스트가 성공한 경우에는 아무런 출력이 없이 녹색 막대가 보여지게 된다. 테스트가 실패한 경우에는 실패 메시지와 함께 빨간색 막대를 보게 된다. 테스트 실패의 경우는 에러가 발생된 경우와 정상적으로 실행은 되었으나 assert 문의 조건을 만족하지 못하는 경우 발생된다. (assertEquals의 경우 예상 값과 수행 결과 값이 다른 경우 실패)



[그림 6-8] JUnit 테스트코드 실행



[그림 6-9] JUnit 테스트 성공



[그림 6-10] JUnit 테스트 실패

7. Logging

표준프레임워크에서 Log정보 참조를 위해 콘솔화면, File, DB를 활용 할 수 있도록 log4j 기반으로 기능을 제공하고 있다. Log4j는 설정을 통해 개발자에게 필요한 정보를 출력할 수 있도록 설정할 수 있으며, Log레벨 조정을 통해서 필요한 정보가 전달 될 수 있도록 지원한다. 개발 시에는 DEBUG레벨로 상세한 정보를 참조하고 운영 시에는 불필요한 메시지가 출력되지 않도록 INFO, ERROR레벨로 조정하여 소스코드 수정 없이 설정파일의 수정만으로 제어가 가능하다.

7.1 Logging 개요

Logging은 시스템의 개발이나 운영 시에 발생할 수 있는 어플리케이션 내부 정보에 대해서 시스템의 외부 저장소에 기록하거나 화면 등에 출력하여 디버그 또는 시스템의 상황 파악을 쉽게 할 수 있도록 지원하는 서비스이다. 디버그를 위해 System.out.println()을 사용하기도 하는데 예제 프로그램 등 간단한 것이 아닌 실제 프로젝트 환경에서는 사용하지 말아야 한다. System.out.println()을 호출하게 되면 디스크 I/O 동기화 처리가 되기 때문에 전체적인 시스템의 성능이 저하 될 수 있고, System.out.println()으로 디버그 처리한 부분을 일일이 주석처리, 해제를 하는 것은 개발 및 운영의 효율성을 떨어트릴 수 있다. 표준프레임워크에서는 Apache log4j를 활용하여 logging을 지원하며 주요 특징은 다음과 같다.

- 서브시스템 (패키지) 별로 상세한 Log 정책 부여
- 날짜형식, 시간형식 등 다양한 형식의 Log 메시지 형태 지정
- 다양한 매체(File, DB, Mail 등)에 대한 Log 관리 가능
- 레벨(debug, info, warn, error 등) 별로 로그를 기록

[표 7-1] Log4j 중요 컴포넌트

컴포넌트	설 명
Logger	Log파일을 작성하는 클래스로 Log를 기록할 대상을 패키지 또는 클래스 이름으로 지정하고 이에 대한 로그레벨과 Appender를 지정
Appender	Log를 출력하는 위치로 화면, 파일, DB등 지정. 화면출력의 경우 console 지정
Layout	Appender의 출력 포맷으로 날짜, 시간, 클래스명 등의 정보를 Log 내용으로 지정

Logger에서 Log 레벨을 지정할 수 있으며, 지정된 Log 레벨보다 높은 메시지도 함께 출력되게 된다. 예를 들어 Log 레벨이 ERROR이면 ERROR이외 메시지는 출력되지 않는다. 레벨이 DEBUG로 지정되면 레벨이 높은 INFO, WARN, ERROR 메시지도 함께 출력된다. 개발 시에는 Log 레벨을 DEBUG로 두고 메소드 실행결과 등 확인이 필요한 Log를 출력하고, 실제 운영시에는 Log 레벨을 INFO로 두고 중요 정보확인 및 에러메시지 등을 관리하는 것이 일반적이다.

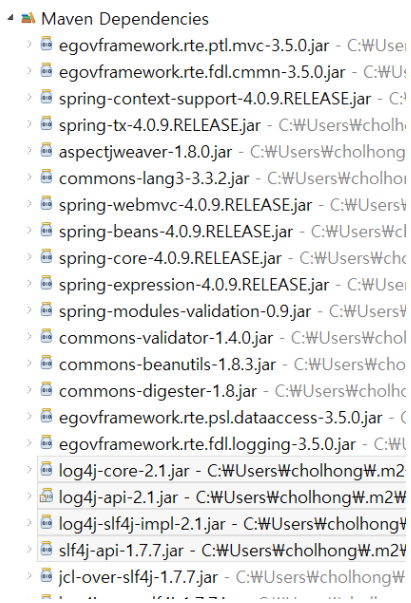
[표 7-2] Log레벨

Log레벨	설 명
Error	처리 중 문제가 발생한 상태
Warn	처리 가능한 문제이지만, 향후 시스템 에러의 원인이 될 수 있는 경고성 메시지
Info	로그인, 상태 변경 등 정보성 메시지
Debug	개발시 디버그 용도로 사용할 메시지

7.2 Log4j 활용

Log4j를 활용하기 위해서는 egovframe.rte.fdl.logging 참조 가능하도록 설정이 필요하다. pom.xm에 직접 설정을 하지 않아도 전자정부 표준프레임워크에서 반드시 설정이 필요한 egovfrmae.rte.psl.dataaccess와 egovframe.rte.ptl.mvc 참조설정을 하면 의존성 설정에 의해서 자동으로 설정이 된다. Log4j-core-2.1.jar 및 log4j-api-2.1.jar 등 필요 라이브러리 의존성이 설정되어 있는 것을 확인 할 수 있다. 아래 설정은 egovframe.rte.fdl.logging에 설정되어 있는 내용과 개발환경에서 설정된 라이브러리 화면이다.

```
<dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
                <version>2.5</version>
        </dependency>
<dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j-impl</artifactId>
                <version>2.5</version>
        </dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
                <version>1.7.18</version>
        </dependency>
<dependency>
          <groupId>org.slf4j</groupId>
          <artifactId>log4j-over-slf4j</artifactId>
          <version>1.7.18</version>
</dependency>
```



[그림 7-1] Log4j 참조설정 및 라이브러리 화면

Log4j 활용을 위해서는 Logger, Appender를 설정하는 설정파일(log4j2.xml)이 필요하다. 설정파일은 src/main/resources 밑에 위치한다. 설정파일에 Default로 설정되는 logger는 Root level로 표현된다. 아래 설정파일에서는 ERROR가 default 레벨로 표시되어 logger가 지정되지 않으면 Error 레벨의 Log만 출력되게 된다. Spring 프레임워크에서 출력하는 메시지는 logger명이 “org.springframework”로 되어 있고 INFO 레벨로 지정되어 있다. 소스코드에 해당되는 logger이름은 “lab”으로 되어 있고 레벨은 INFO로 설정되어 있다.

DEBUG Log를 활용하려면 레벨을 DEBUG로 바꾸고 활용한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d %5p [%c] %m%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="java.sql" level="DEBUG" additivity="false">
            <AppenderRef ref="console" />
        </Logger>
        <Logger name="lab" level="INFO" additivity="false">
            <AppenderRef ref="console" />
        </Logger>
       <Logger name="jdbc.sqltiming" level="INFO" additivity="false">
           <AppenderRef ref="console" />
       </Logger>
      <Logger name="org.springframework" level="INFO" additivity="false">
           <AppenderRef ref="console" />
      </Logger>
        <Root level="ERROR">
            <AppenderRef ref="console" />
        </Root>
    </Loggers>
</Configuration>
```

[소스 7-1] log4j2.xml 설정

Log를 출력하기 위해서는 먼저 LogManager의 getLogger() 메소드 호출을 통해 Log출력에 활용할 logger를 설정해주어야 한다. getLogger()는 static 메소드이므로 인스턴스 생성을 하지 않고 호출이 가능하다. getLogger()에 string 값이 입력되면 입력 값에 해당되는 Logger를 활용할 수 있게 된다. Null 값이 입력되면 Class 이름이 입력된다. 앞에서 logger의 이름을 "lab"으로 입력하였기 때문에 getLogger("lab")이라고 입력을 해도 되지만, getLogger(.class.getName())의 형태로 패키지와 클래스네임을 입력하도록 하면 패키지 이름이 입력되어 logger "lab"을 입력한 것과 같게 된다. Logger가 설정되면 출력하고자 하는 Log레벨을 지정하고 출력하고자 하는 메시지를 Log로 출력한다.

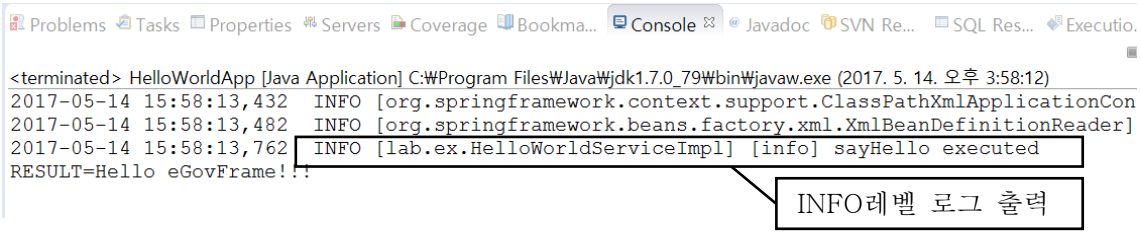
```
package lab.ex;

import org.apache.log4j.Level;
import org.apache.logging.log4j.Logger;

import org.apache.logging.log4j.LogManager;

public class HelloWorldServiceImpl implements HelloWorldService{

    Logger logger = LogManager.getLogger(HelloWorldServiceImpl.class.getName());

        private String name;

        public void setName(String name) {
                this.name = name;
        }

        public String sayHello() {
                logger.debug("[debug] sayHello executed");
                logger.info("[info] sayHello executed");
                return "Hello " + name + "!!!" ;
        }
}
```

클래스 이름으로 logger지정
lab.HelloWorldServiceImpl.class
lo4j2.xml에서 lab으로 설정된 logger 매핑

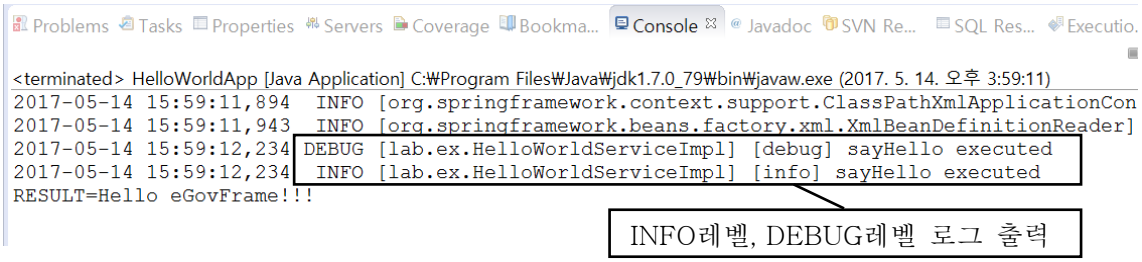
로그를 DEBUG레벨과 INFO레벨로 각각 출력

[소스 7-2] logger 활용

log4j2.xml에서 Log레벨이 INFO로 지정되어 있으므로 예제를 실행하면 debug 메시지는 출력 되지 않는다. Log 레벨을 DEBUG로 수정하면 debug와 info 두 개의 로그 메시지가 출력되는 것을 확인 할 수 있다.



[그림 7-2] INFO레벨 로그



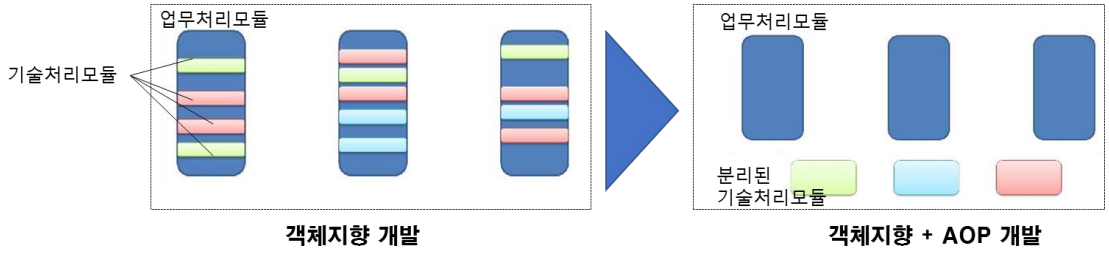
[그림 7-3] DEBUG레벨 로그

8. AOP(Aspect Oriented Programming)

프로그램을 개발할 때 로깅, 트랜잭션, 예외처리 등 반복적으로 처리가 되어야 하고, 업무 로직 코드와 혼재되어 전체적으로 소스코드가 복잡해지고 특정 라이브러리에 의존성도 발생되게 된다. 이러한 문제를 해결하기 위해 기술적인 처리가 필요한 영역을 별로 공통모듈로 분리하고 대상과 발생시키는 시점을 정의하여 자동적으로 실행될 수 있도록 하여 변경이 용이하고 재사용성을 높일 수 있도록 한다.

8.1 AOP 개요

AOP는 객체지향 프로그래밍을 보완하는 개념으로 어플리케이션을 객체지향적으로 모듈화하여 작성하더라도 다수의 객체들에 분산되어 중복적으로 존재하는 공통모듈이 여전히 존재한다. 이를 AOP에서는 횡단관심(cross concern)이라고 부른다. 업무 처리를 하는 모듈(주된 관심, primary concern)에서 로깅, 보안, 트랜잭션, 예외처리 등의 기술적인 처리를 수행하기 위한 소스코드가 포함이 되어 업무 로직에 집중하기가 어렵다. 또한, 기술적 처리를 위한 모듈 들과의 의존성이 생겨 어플리케이션 변경 시에 유연성이 떨어진다. 일반 어플리케이션 환경과 AOP가 적용된 환경을 비교해 보면, AOP가 적용되지 않으면 모듈안에 주된 관심과 횡단 관심이 섞여 존재하게 된다. 그림에서 처럼 업무 모듈에 기술적 처리를 위한 cross concern이 섞여서 존재한다. AOP를 활용하면 기술적 처리를 위한 모듈을 advice로 분리하여 별도로 존재하게 한다. 별도로 존재하는 advice가 필요한 시점에 동작할 수 있도록 동작 하는 대상(point cut)과 동작 되는 시점(joint point)을 규칙형태로 정의한다.



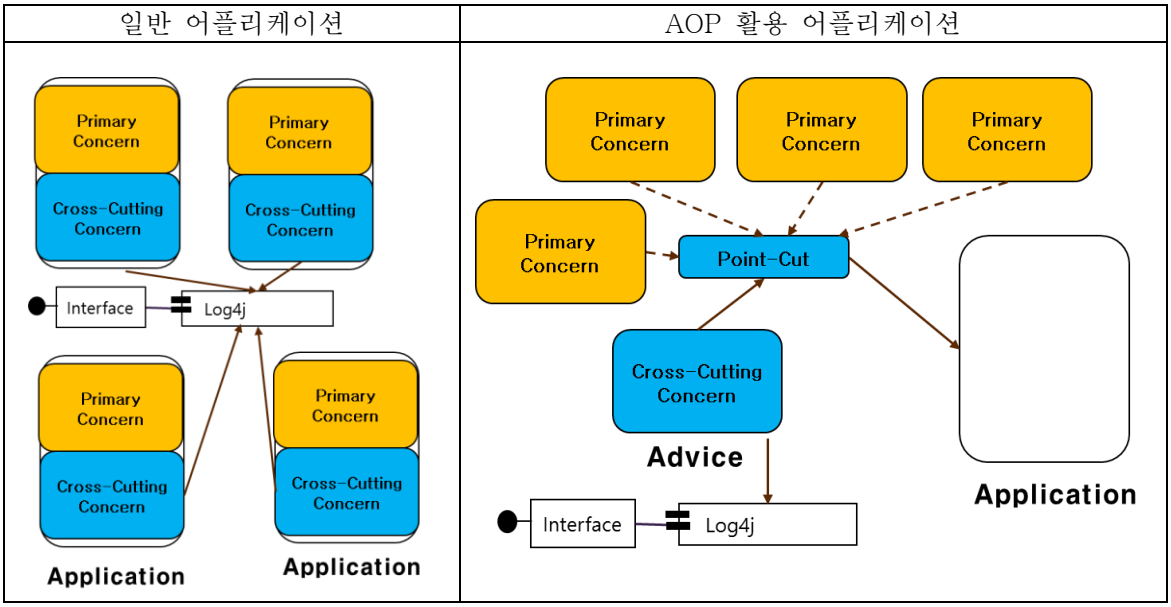
[그림 8-1] AOP 개념

예를 들어 로그를 처리하기 위해 log4j를 활용한다면 log4j 모듈에 대한 의존성이 생기게 되며, 업무 처리 모듈에서 로그를 출력하기 위한 소스코드가 포함되게 된다. 이러한 문제를 해결하기 위해서는 기술적 처리를 위한 cross concern을 advice로 별도로 분리하고, 업무 처리 모듈 중에 어느 메소드(point cut)에서 어느 시점(joint point)에 처리를 해야 되는지를 규칙형태로 정의 한다. 정해진 규칙에 해당되게 되면 advice의 정해진 메소드가 실행(Weaving)된다. AOP에서 활용되는 주요 개념은 다음과 같다.

[표 8-1] 주요개념

<table>
<tr><td>개념</td><td colspan="2">설 명</td></tr>
<tr><td rowspan="6">Joint point</td><td colspan="2">횡단 관심 모듈이 삽입되어 동작할 수 있는 실행 가능한 시점을 결정
특정 메소드 실행 전, 후, 결과 값 리턴 등</td></tr>
<tr><td>Advice</td><td>설명</td></tr>
<tr><td>Before</td><td>메소드 호출 전에 수행</td></tr>
<tr><td>After returning</td><td>메소드가 실행되어 결과값이 리턴 된 후 수행</td></tr>
<tr><td>After throwing</td><td>예외가 발생되었을 때 수행</td></tr>
<tr><td>After advice</td><td>메소드가 실행된 후에 수행 (정상,예외에 상관없음)</td></tr>
<tr><td>Point cut</td><td colspan="2">어느 모듈 및 메소드를 대상으로 실행이 되는지를 결정
예를 들어 특정 클래스에 있는 모든 메소드가 호출 될 때</td></tr>
<tr><td>Advice</td><td colspan="2">공통활용을 위해 분리된 횡단 관심 모듈 (cross concern)</td></tr>
<tr><td>Aspect</td><td colspan="2">Advice와 Point cut의 조합</td></tr>
<tr><td>Weaving</td><td colspan="2">Joint point와 point cut에 의해 결정된 대상 및 시점에 advice를 실행하는 과정</td></tr>
</table>

그림에서 일반 어플리케이션은 모든 모듈이 log4j에 의존성을 가지고 있으나, AOP를 활용하게 되면 분리된 advice에만 log4j의존성을 가지게 되며, 주된 관심인 업무 모듈에서 cross concern인 log4j를 호출하는 모듈을 포함할 필요가 없어진다. 정해진 규칙에 해당하게 되면 자동으로 실행되도록 한다.



[그림 8-2] AOP활용 어플리케이션과 일반 어플리케이션

AOP를 활용하여 얻게 되는 효과는 다음과 같다.

[표 8-2] AOP 활용 효과

활용효과	설 명
중복코드 제거	횡단관심이 여러 모듈에 반복적으로 기술되는 문제를 방지
비즈니스 로직 가독성 향상	핵심기능 코드로부터 횡단관심코드를 분리하여 가독성 향상
생산성 향상	비즈니스 로직에 집중하여 개발 진행
재사용성 향상	횡단관심코드는 여러 모듈(Advice)에서 호출활용
변경용이성 증대	횡단관심코드가 하나의 모듈(Advice)로 관리 되기 때문에 변경 발생시 용이하게 수정가능

8.2 XML을 활용한 AOP 설정

AOP활용을 위해 설정파일에 advice, point cut, joint point를 설정한다. 파일명은 임의로 정의를 해도 되지만 구분을 위해 주로 context-advice.xml로 명명한다. 설정은 aop라는 prefix를 활용하여 설정을 한다. 먼저 pint cut 설정은 id와 대상을 expression을 활용하여 지정한다. *lab..*Impl.*(..)의 표현은 lab 패키지 밑에 Impl로 되어 있는 모든 클래스의 모든 메소드에 대해서 AOP를 적용한다는 것이다. Joint point의 설정은 before, after-returning, after-throwing, after, around에 대해서 각각 해당되는 point cut id를 지정하고 실행할 advice의 메소드를 지정한다. Before joint point의 경우 point cut은 “targetMethod”이고 메소드가 실행되기 전에 “beforeTargetMethod”가 실행되게 된다. After-returning의 경우 “afterReturningTargetMethod”가 실행되며 결과값을 “retVal” 변수로 받아오게 된다. 로그 등을 출력하는 등 결과 값을 활용할 경우 사용된다.

```
<bean id="adviceUsingXML" class="lab.ex.AdviceUsingXML" />
<aop:config>
  <aop:pointcut id="targetMethod" expression="execution(* lab..*Impl.*(..))" />
  <aop:aspect ref="adviceUsingXML">
    <aop:before pointcut-ref="targetMethod" method="beforeTargetMethod" />
    <aop:after-returning pointcut-ref="targetMethod"
        method="afterReturningTargetMethod" returning="retVal" />
    <aop:after-throwing pointcut-ref="targetMethod"
        method="afterThrowingTargetMethod" throwing="exception" />
    <aop:after pointcut-ref="targetMethod" method="afterTargetMethod" />
    <aop:around pointcut-ref="targetMethod" method="aroundTargetMethod" />
  </aop:aspect>
</aop:config>
```

Advice bean 설정

AOP 대상 클래스 및 메소드 설정

Advice bean 설정

Joint point 설정

[소스 8-1] AOP활용 위한 context-advice.xml 설정

Point cut을 설정하기 위해 다음과 같은 지정자를 활용할 수 있다.

[표 8-3] Point cut 지정자

지정자	설명
execution	메소드에 대해 joint point 정의
within	특정 타입 속하는 joint point 정의
this	빈 참조가 주어진 타입의 인스턴스를 갖는 joint point 정의
target	대상 객체가 주어진 타입을 갖는 joint point 정의
args	인자가 주어진 타입의 인스턴스인 joint point 정의
@target	수행중인 객체의 클래스가 주어진 타입의 annotation을 갖는 joint point 정의
@args	전달된 인자의 런타임 타입이 주어진 타입의 annotation을 갖는 joint point 정의
@within	주어진 annotation을 갖는 타입 내 joint point 정의
@annotation	대상 객체가 주어진 annotation을 갖는 joint point 정의

앞에서 설명된 여러 지정자를 활용하여 Point cut을 설정하는 예제를 보면 다음 표와 같다. 정의 예제에서 execution은 메소드를 지정하고 within은 타입을 기준으로 설정하는 차이가 있다. Execution을 사용하게 되면 메소드별로 AOP를 설정하게 되고, within은 인터페이스 별로 설정하게 된다. Execution에서 모든 메소드를 선택하게 되면, within으로 설정한 것과 같은 형태로 동작하게 된다. 또한, 두 개 이상의 표현식이 조합이 되어 활용 될 수 있다. IF문처럼 and (&&), or (||), not (!)의 연산자를 활용하여 point cut을 정의 할 수 있다.

[표 8-4] Point cut 정의예제

Point cult	Joint points 설정
execution(public * *(..))	public 메소드 실행
execution(* set*(..))	이름이 set으로 시작하는 모든 메소드명 실행
execution(* xyz.service.*.*(..))	service 패키지의 모든 메소드 실행
execution(* xyz.service..*.*(..))	service 패키지와 하위 패키지의 모든 메소드 실행
execution(* xyz.service..*Impl.*(..))	service 패키지와 하위 패키지의 Impl로끝나는 모든 클래스 모든 메소드 실행
within(com.xyz.service.*)	service 패키지 내의 모든 결합점
within(com.xyz.service..*)	service 패키지 및 하위 패키지의 모든 결합점
within(com.xyz.service.AccountService)	AccountService 인터페이스의 모든 메소드 실행
this(com.xyz.service.AccountService)	AccountService 인터페이스를 구현하는 프록시 개체의 모든 결합점
target(com.xyz.service.AccountService)	AccountService 인터페이스를 구현하는 대상 객체의 모든 결합점
args(java.io.Serializable)	하나의 파라미터를 갖고 전달된 인자가 Serializable인 모든 결합점

@target(org.springframework .transaction.annotation.Transactional)	대상 객체가 @Transactional 어노테이션을 갖는 모든 결합점
@within(org.springframework .transaction.annotation.Transactional)	대상 객체의 선언 타입이 @Transactional 어노테이션을 갖는 모든 결합점
@annotation(org.springframework .transaction.annotation.Transactional)	실행 메소드가 @Transactional 어노테이션을 갖는 모든 결합점
@args(com.xyz.security.Classified)	단일 파라미터를 받고, 전달된 인자 타입이 @Classified 어노테이션을 갖는 모든 결합점
bean(accountRepository)	“accountRepository” 빈
!bean(accountRepository)	“accountRepository” 빈을 제외한 모든 빈
bean(*)	모든 빈
bean(account*)	이름이 'account'로 시작되는 모든 빈
bean(*Repository)	이름이 “Repository”로 끝나는 모든 빈
bean(accounting/*)	이름이 “accounting/“로 시작하는 모든 빈
bean(*dataSource) || bean(*DataSource)	이름이 “dataSource” 나 “DataSource” 으로 끝나는 모든 빈

8.3 Advice 설정

AOP설정을 위해 AdviceUsingXML advice를 설정하였다. Advice에서 before, After-returning 등의 Joint point에서 실행될 메소드를 정의한다. Before의 경우 메소드가 수행되기 전에 실행되며, 메소드가 호출되어 전달 되는 인수(argument)의 값을 호출하여 받아 올 수 있다. 소스코드에서 before는 해당 클래스 및 메소드 이름을 출력하고, 전달 받은 인수 값을 출력한다. Advice가 실행되면 “Advice Using XML beforeTargetMethod” 가 출력되고 클래스명과 메소드명이 출력된다. 전달 받은 인수가 있는 경우에 인수 값도 함께 출력된다.

```
public class AdviceUsingXML {
        private static final Logger LOGGER   =
                LogManager.getLogger(AdviceUsingXML.class);

    public void beforeTargetMethod(JoinPoint thisJoinPoint) {

    String className = thisJoinPoint.getTarget().getClass().getSimpleName();
    String methodName = thisJoinPoint.getSignature().getName();

    StringBuffer buf = new StringBuffer();
    buf.append("AdviceUsingXML.beforeTargetMethod : [" + className
        + "." + methodName + "()]");

    Object[] arguments = thisJoinPoint.getArgs();

    int argCount = 0;
```

Before Advice 정의

Joint point에서 클래스명, 메소드명 정보 가져옴

Joint point에서 인수정보 가져옴

```
        for (Object obj : arguments) {
            buf.append("₩n - arg ");
            buf.append(argCount++);
            buf.append(" : ");
            buf.append(ToStringBuilder.reflectionToString(obj));
        }
        LOGGER.debug(buf.toString());
    }
```

인수 값을 문자열로 변환

[소스 8-2] Before advice

After returning의 경우 메소드 실행이 정상적으로 실행 된 후에 수행 된다. 결과 값은 XML설정에 returning으로 명시된 변수로 전달 받게 된다. 소스 8-1에서 retVal로 정의 되었다.

```
public void afterReturningTargetMethod(JoinPoint thisJoinPoint, Object retVal) {
        String className = thisJoinPoint.getTarget().getClass().getSimpleName();
        String methodName = thisJoinPoint.getSignature().getName();

        StringBuffer buf = new StringBuffer();
        buf.append("AdviceUsingXML.afterReturningTargetMethod : ["
            + className + "." + methodName + "()]");
        buf.append("₩n");
        if (retVal instanceof List) {
            List<?> resultList = (List<?>) retVal;
            buf.append("resultList size : " + resultList.size() + "₩n");
            for (Object oneRow : resultList) {
                buf.append(ToStringBuilder.reflectionToString(oneRow));
                buf.append("₩n");
            }
        } else {
            buf.append(ToStringBuilder.reflectionToString(retVal));
        }
        LOGGER.debug(buf.toString());
    }
```

결과 값 전달 받음

결과값이 다수일 때 출력

결과값이 단 건일 때 출력

[소스 8-3] After returning advice

After throwing advice는 메소드 실행 중 에러 등의 예외 사항이 발생되었을 때 수행되며, exception 내용을 전달 받는다.

```
public void afterThrowingTargetMethod(JoinPoint thisJoinPoint,
            Exception exception) throws Exception {
        LOGGER.debug("AdviceUsingXML.afterThrowingTargetMethod executed.");
        LOGGER.error("에러가 발생했습니다. {}", exception);
```

예외 발생 시에 내역 전달 받음

```
        throw new Exception("에러가 발생했습니다.", exception);
    }
```

[소스 8-4] After throwing advice

After는 메소드 실행 후에 정상종료 및 예외발생 등 결과에 상관없이 무조건 수행된다. If 문에서 finally와 비슷한 역할을 하며, 주로 리소스 해제 작업과 같은 마무리 작업을 위해 활용한다.

```
public void afterTargetMethod(JoinPoint thisJoinPoint) {
        LOGGER.debug("AdviceUsingXML.afterTargetMethod executed.");
    }
```

[소스 8-5] After advice

Around는 메소드 실행 전후에 수행 된다. 결과 값을 변환 해야 될 경우 활용한다. 결과 값을 전달 받아서 가공한 후 리턴 할 수 있다. Proceed() 앞은 메소드 실행 전에 수행이 되고, 뒤는 메소드가 실행된 다음에 수행된다.

```
public Object aroundTargetMethod(ProceedingJoinPoint thisJoinPoint)
            throws Throwable {
        LOGGER.debug("AdviceUsingXML.aroundTargetMethod start.");

        Object retVal = thisJoinPoint.proceed();

        LOGGER.debug("AdviceUsingXML.aroundTargetMethod end.");
        return retVal;
    }
```

메소드 수행 전에 실행

메소드 수행 후에 실행

[소스 8-6] Around advice

Point cut의 메소드가 실행될 때 joint point에 따른 advice 실행순서를 정리해보면 다음과 같다. 정상실행과 예외발생의 경우 모두 before, around (실행 전)의 순서로 수행이 되며, 대상 메소드가 실행된 후에 정상실행의 경우 around(실행 후), after, after-returning의 순서로 실행이 된다. 예외 발생의 경우는 after를 실행하고 after-throwing을 실행하고 종료하게 된다.

[표 8-5] Advice 실행 순서

정상실행	예외발생
1. before 2. around (대상 메소드 실행 전) 3. 대상 메소드 4. around (대상 메소드 실행 후) 5. after(finally) 6. after-returning	1. before 2. around (대상 메소드 실행 전) 3. 대상 메소드 (Exception 예외발생) 4. after(finally) 5. after throwing

8.4 AOP 활용 및 실행결과

sayHello()와 sayError()의 두 개의 메소드를 가지는 HelloWorldServiceImpl 서비스를 작성하였다. HelloWorldApp에서 호출되어 실행되는 형태이며, sayHello()는 message를 인수로 받아서 문자열을 생성하여 결과를 출력하게 된다. Before advice에서 전달 받는 인수를 출력하게 되어 있어 내용 확인이 가능하며, After returning에서 결과 값을 리턴 할 때 결과를 출력하게 되어 있어 결과 확인이 가능하다. sayError()는 After throwing이 제대로 수행되는지를 확인하기 위해 고의로 0으로 나누어 에러를 발생시켰다.

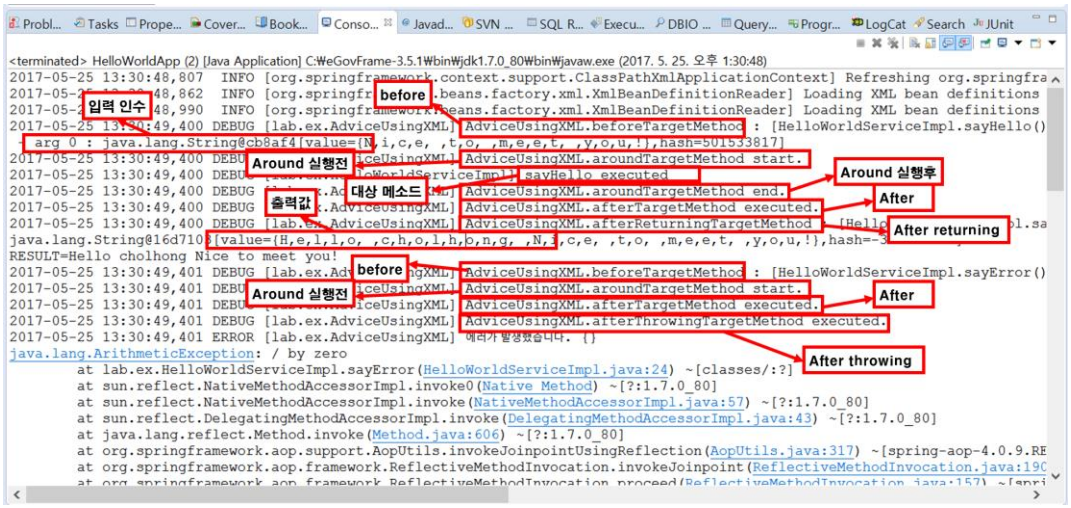
```
public class HelloWorldServiceImpl implements HelloWorldService{

    Logger logger1 = LogManager.getLogger(HelloWorldServiceImpl.class.getName());

        private String name;
        public void setName(String name) {
                this.name = name;
        }
        public String sayHello(String message) {
                logger1.debug("sayHello executed");
                return "Hello " + name + " "+ message ;
        }
        public void sayError() {
        double i = 100/0;
        }
}
```

정상 실행 AOP 동작확인

고의로 에러를 발생시켜 예외처리 AOP 동작 확인

[소스 8-7] HelloWorldService Implementation

실행하기 전에 주의할 점은 예제에서 해당 advice에서 로그를 출력하도록 되어 있는 debug 레벨로 설정되어 있어 로그레벨이 Debug로 되어야 확인이 가능하다. HelloWorldApp을 선택하고 Run As > Java Application을 선택하여 실행하면 다음과 같은 결과를 얻는다.



[그림 8-3] AOP 실행결과

8.5 표준프레임워크에서 Exception 처리 AOP 활용

표준프레임워크에서 예외 처리를 위해서 AOP를 활용하고 있다. AOP를 활용하지 않으면 모든 메소드에서 예외처리를 중복적으로 구현해야 하는 불편함이 있다. 특히, 예외 종류에 따라서 처리가 다르게 수행해야 할 경우에는 메소드별로 중복적으로 관련로직을 개별 구현해야 한다. 예외 처리를 위한 advice에서는 예외가 BizException, Runtime Exception, 실행환경 Exception인지 예외 종류에 따라서 분기처리가 되어 다른 처리를 수행하게 된다.

```
<bean id="exceptionTransfer"
 class="egovframework.rte.fdl.cmmn.aspect.ExceptionTransfer">
...
</bean>
<aop:config>                                          예외처리가 될 대상 (point cut) 설정
  <aop:pointcut id="serviceMethod"
  expression="execution(* egovframework.rte.sample..impl.*Impl.*(..))" />
  <aop:aspect ref="exceptionTransfer">
       <aop:after-throwing throwing="exception“
        pointcut-ref="serviceMethod" method="transfer" />
  </aop:aspect>
</aop:config>
```

[소스 8-8] 표준프레임워크 예외처리 AOP 설정

```
public class ExceptionTransfer {

  public void transfer(JoinPoint thisJoinPoint, Exception exception) throws Exception {
    ...
    if (exception instanceof EgovBizException) {                    BizException 처리
      log.debug("Exception case :: EgovBizException ");
      EgovBizException be = (EgovBizException) exception;
      getLog(clazz).error(be.getMessage(), be.getCause());
      processHandling(clazz, exception, pm, exceptionHandlerServices, false);
      throw be;
    } else if (exception instanceof RuntimeException) {             Runtime Exception 처리
      log.debug("RuntimeException case :: RuntimeException ");
      RuntimeException be = (RuntimeException) exception;
      getLog(clazz).error(be.getMessage(), be.getCause());
      processHandling(clazz, exception, pm, exceptionHandlerServices, true);
      ...
      throw be;
    } else if (exception instanceof FdlException) {                 실행환경 Exception 처리
      ...
      throw fe;
    } else {
      ...
      throw processException(clazz, "fail.common.msg", new String[] {}, exception,
locale);
    }
```

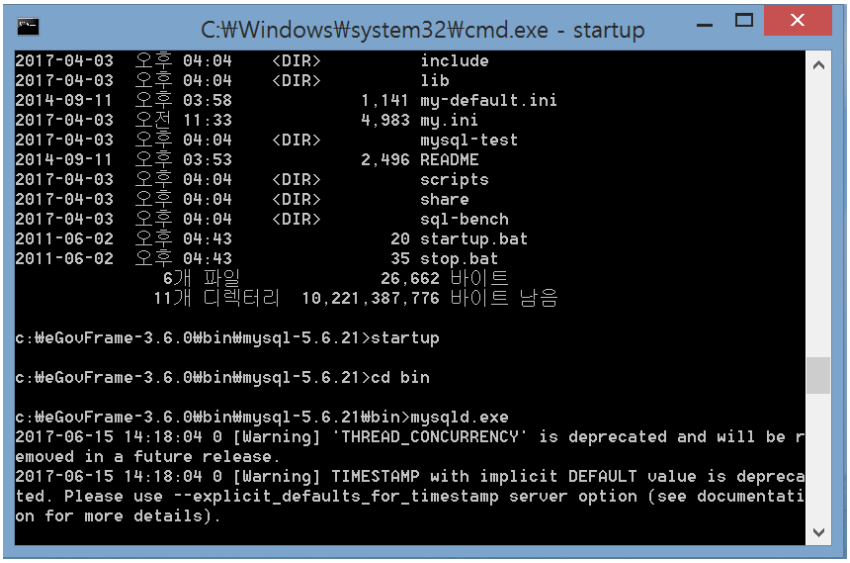
[소스 8-9] 예외처리를 위한 Advice인 Exception Transfer

9. Data Access(MyBatis)

DB를 활용하여 데이터를 연계하는 것은 정보시스템에서 필수적이고, 아주 중요한 작업이다. Java에서 제공하는 SQL 처리 기능을 활용하면 불필요하고 반복적인 코드 작성이 필요하며, SQL이 소스코드에 존재하여 변경이 유연하지 않고 관리가 어렵다. MyBatis는 SQL을 별도 XML파일에 위치하여, 간단하게 SQL을 실행할 수 있도록 지원한다.

9.1 MySQL DB 활용

DB프로그래밍을 위해서는 MySQL 설치와 활용이 필요하다. 표준프레임워크 교육 실습교재에 MySQL을 배포하고 있어 이를 활용하면 계정생성, DB생성 등 별도 작업없이 바로 쉽게 활용할 수 있다. 실습교재에 mysql-5.6.21 디렉터리에 포함이 되어 있으며, 이를 실행시키기 위해서는 아래와 같이 startup을 하면 DB가 구동 되게 된다.



[그림 9-1] MySQL 구동

실습교재에 포함된 MySQL에는 예제에서 활용할 DB(com)와 사용자계정(com)이 만들어져 있다. 별도로 MySQL을 설치하여 활용하는 경우에는 root계정으로 접속한 후에 아래의 명령을 활용하여 DB와 계정을 생성하고 진행한다. DB이름은 com, 사용자 계정 com, 비밀번호는 com01로 설정된다. MySQL이 설치된 bin 디렉터리에서 “mysql -uroot”를 실행시키고 입력한다. 패스워드를 설정한 경우에는 “mysql -uroot -p*패스워드*”를 입력하여 실행할 수 있다.

```
create database com;
create user 'com'@'%'   identified by 'com01';
GRANT ALL PRIVILEGES ON com.* TO 'com'@'%' WITH GRANT OPTION;
```

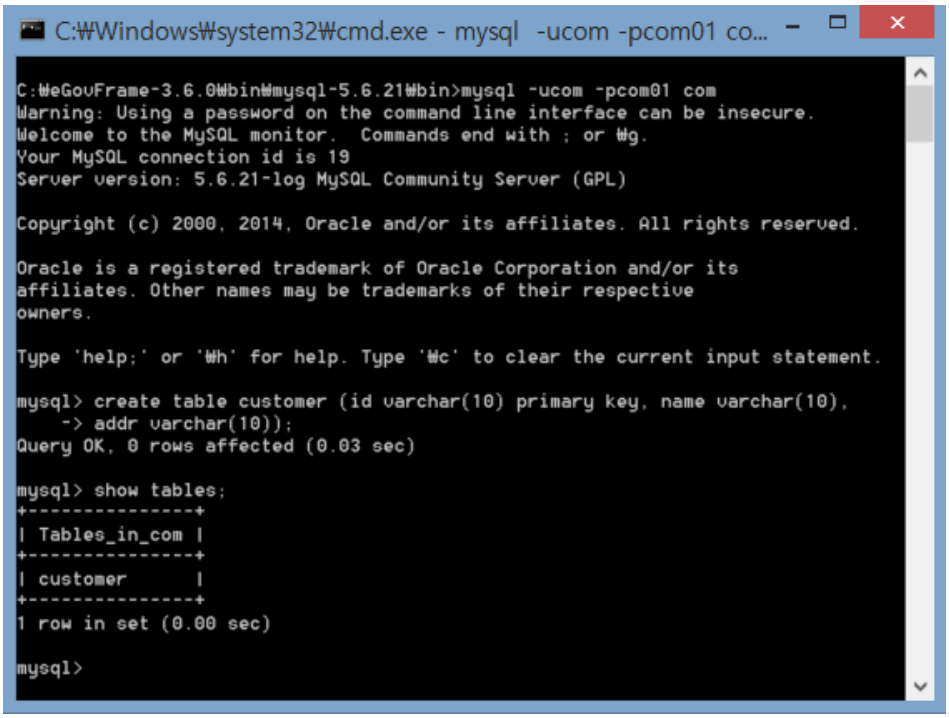
[소스 9-1] MySQL계정과 DB생성 (MySQL을 직접 설치한 경우)

DB에 테이블을 생성하기 위해 새로 생성한 계정으로 mysql을 실행하기 위해서는 mysql [옵션] [DB명]을 입력해서 실행해야 한다. “mysql -ucom -pcom01 com”을 입력하여 com 사용자 계정으로 com DB로 접속한다. 테이블 생성을 위한 스크립트를 입력하고 실행하면 테이블이 생성된다. 테이블은 id, name, addr의 3개 컬럼을 가지고 있고, id가 primary 키이며, 생성 SQL은 다음과 같다.

```
create table customer (
id varchar(10) primary key,
name varchar(10),
addr varchar(10));
```

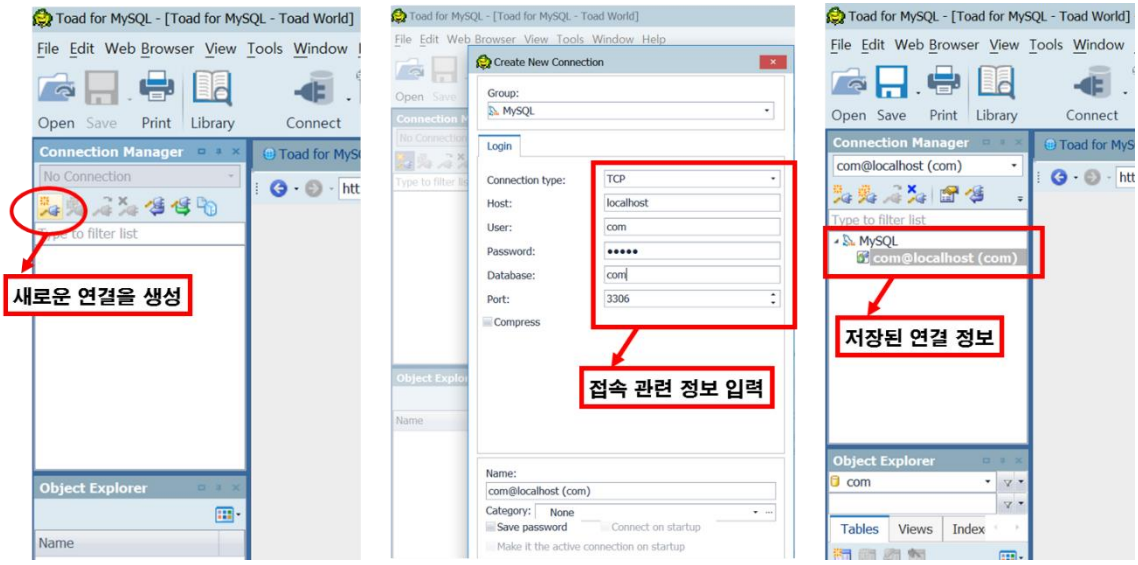
[소스 9-2] MySQL계정과 DB생성 (MySQL을 직접 설치한 경우)

테이블이 생성되었는지 확인하기 위해 “show tables;”를 실행하여 결과를 확인한다. customer 테이블이 성공적으로 생성되었음을 확인할 수 있다.



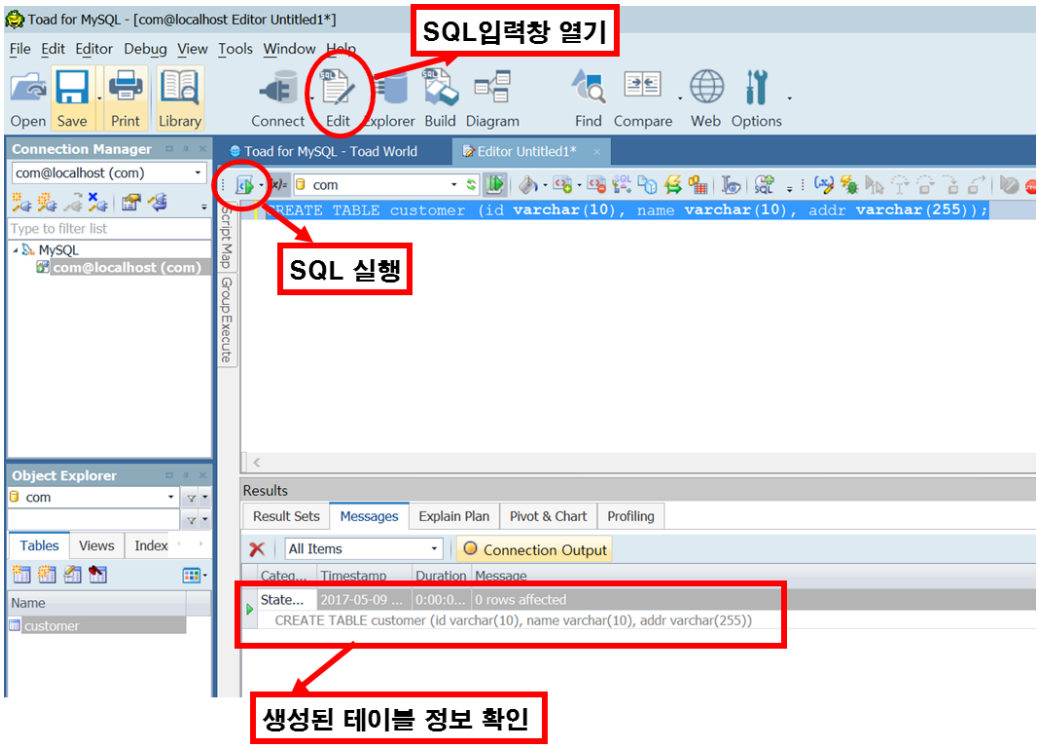
[그림 9-2] 테이블 생성

커맨드라인 형태 보다 편리하게 활용가능한 전문 DB관리 툴도 활용할 수 있다. TOAD for MySQL은 Freeware로 별도 비용없이 자유롭게 활용할 수 있는 툴이며, 편리하게 DB 관리가 가능하다. 생성된 사용자 계정을 활용하여 HOST: localhost User: com Password com01 port 3306을 입력하여 새로운 연결설정을 만들 수 있다. Root로 로그인하여 사용자 계정 및 DB를 먼저 생성할 수도 있다.



[그림 9-3] TOAD를 이용한 새로운 연결 설정

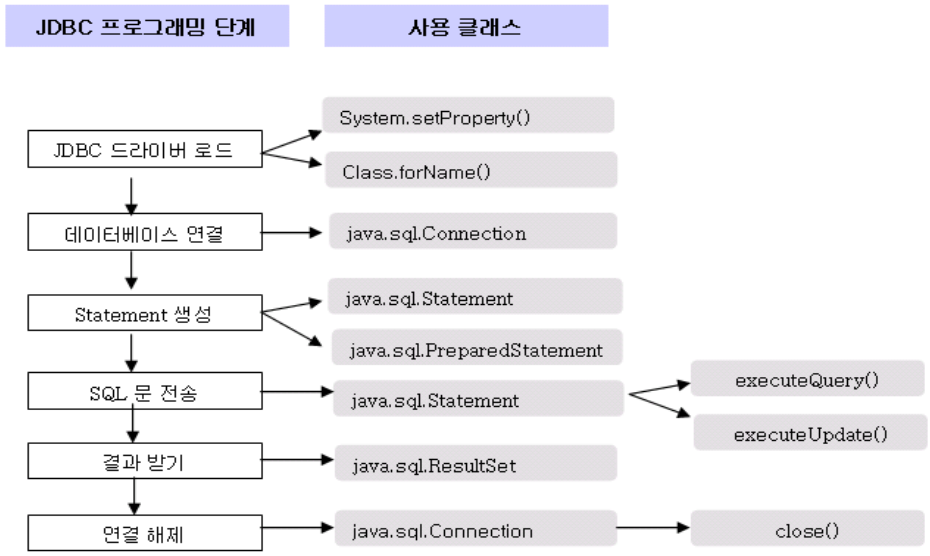
연결이 되었으면 Edit SQL을 이용하여 직접 SQL을 입력하고 실행 시킬 수 있다. 아래 그림은 TOAD를 활용하여 테이블 생성을 위한 SQL을 입력하고, 실행하여 테이블을 생성하는 화면이다.



[그림 9-4] TOAD를 이용하여 테이블 생성

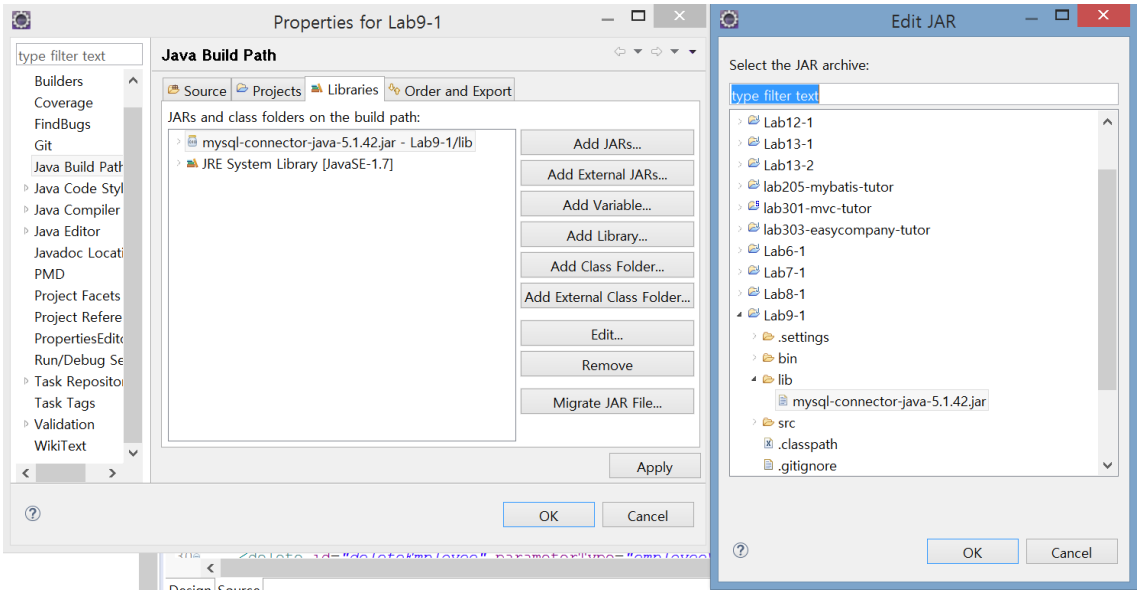
9.2 자바 SQL 프로그래밍

프레임워크를 활용할 때와 활용하지 않을 때 DB 프로그래밍이 달라지는 점을 살펴보기 위해 일반적으로 자바에서 제공하는 SQL 프로그래밍 활용 방법을 살펴보도록 하겠다. 활용 순서는 다음과 같다.



[그림 9-5] 자바 SQL 프로그래밍 과정

제일 먼저 DB연결을 위한 자바드라이버 규격(JDBC)을 활용하여 DB에 연결을 한다. 이를 위해서는 JDBC 드라이버가 클래스패스에 위치해야 한다. JDBC 드라이버는 DB마다 다르기 때문에 활용하고자 하는 DB에 맞는 드라이버가 필요하다. 예제코드는 maven을 활용하지 않기 때문에 직접 파일을 다운로드 받아서 build path를 설정해야 한다. DB드라이버는 활용하는 DB 벤더 사이트 등에서 구할 수 있고, http://repo1.maven.org 등 외부 Maven Repository에서 직접 다운을 받아 활용해도 된다. Build path 설정을 위해서 Project build path 설정을 선택하고 Add JARs를 선택하고 mysql-connector-java-5.1.42.jar를 선택한다. Maven을 활용하는 프로젝트에서는 3장에서 학습했던 것처럼 pom.xml에 dependency를 추가하면 된다.



[그림 9-6] MySQL JDBC 드라이버 Build path 설정

Class.for.Name 명령을 활용하여 드라이버를 지정한 다음에 로드 한다. MySQL 드라이버는 com.mysql.jdbc.Driver이다. 드라이버가 로드 되었으면, DB접속 정보를 가지고 DB에 연결한다. getConnection을 활용하여 DB에 접속하게 되며, 접속 정보는 JDBC:mysql://[서버 도메인 또는 IP]:포트:DB명의 형태로 구성된다. 표준프레임워크 실습교재의 DB를 활용하기 위해서 접속정보는 JDBC:mysql://localhost:3306:com의 형태이다. MySQL의 기본 포트는 3306이며 설정을 통해 포트를 변경할 수 있다. 연결이 된 다음에는 Statement또는 PreparedStatement를 통해 실행 시킬 SQL문을 준비하게 된다. Statement를 활용하는 경우에는 Statement 객체 생성 후에 SQL 문장을 변수 처리부와 함께 문자열로 구현한다.

```
Statement stmt = conn.createStatement();
String sqlString = (“insert into customer values('"
+ request.getParameter("id")+ "','"+ request.getParameter("name")+ "')";
stmt.executeUpdate(sqlString);
```

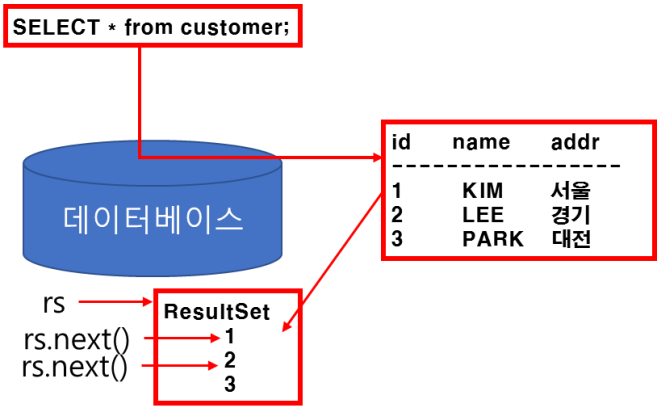
[소스 9-3] Statement를 활용한 SQL문 생성

INSERT나 UPDATE에서 입력 값을 바꾸어 반복적으로 SQL을 실행하게 되는 경우에는 PreparedStatement를 쓰는 것이 성능이 우수하다. Statement를 활용하게 되면 DB에서 매번 새롭게 Parsing 처리를 하기 때문에 성능이 저하된다. PreparedStatement를 활용하는 경우에는 SQL문에는 “?”로 변수 처리를 하고 setString을 활용하여 값을 대입한다.

```
PreparedStatement pstmt = conn.prepareStatement(“insert into customer values(?,?)”);
pstmt.setString(1,request.getParameter("id");
pstmt..setString(2, request.getParameter("name");
pstmt.executeUpdate();
```

[소스 9-4] PreparedStatement를 활용하여 SQL문 생성

INSERT, UPDATE, DELETE 등 SQL을 실행하여 전달받을 결과 값이 없는 경우에는 executeUpdate를 통해 실행하며, SELECT의 경우에는 executeQuery를 실행시키고 결과값인 ResultSet을 전달 받는다. ResultSet은 커서 개념의 연결 포인터로 next() 메소드를 통해 다음 ROW로 이동하여 결과값을 전달 한다.



[그림 9-7] ResultSet처리

ResultSet처리를 위해서 ResultSet에 결과 값이 없을 때까지 루프를 반복하고 getString() 또는 getInt()을 실행하여 값을 전달 받는다.

```
ResultSet rs = pstmt.executeQuery();
while(rs.next()) {
    id = rs.getString(1); // or rs.getString("id");
    name = rs.getString(2); // or rs.getString("name");
}
```

[소스 9-5] executeQuery 실행 및 ResultSet 처리

전체적인 자바 SQL을 활용하여 테이블에 값을 입력하고 출력하는 방법은 다음과 같이 구현 된다. 실행하기 전에 customer 테이블이 생성된 MySQL DB가 구동이 되어 있어야 하며, MySQL JDBC드라이버(mysql-connecor-java-5.1.42.jar)를 빌드패스에 등록해야 한다. 실행을 하게 되면 customer 테이블에 배열로 표현된 3개의 레코드를 테이블에 INSERT하고 다시 SELECT를 수행하여 INSERT 결과를 System.out.println()으로 출력한다.

```
import java.sql.*;

public class JDBCExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) { }
        Connection conn = null;
        try {
            conn = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/com", "com", "com01" );
```

JDBC 드라이버 로드

DB연결

```
                int[] id = { 1, 2, 3 };
                String[] name = { "KIM", "PARK", "LEE" };
                String[] addr = { "A", "B", "C" };

        String sql = "INSERT INTO customer (id, name, addr) VALUES ( ?, ?, ? )";
            PreparedStatement pstmt = conn.prepareStatement(sql);
                for(int i=0; i < id.length; i++) {
                        pstmt.setInt(1, id[i]);
                        pstmt.setString(2, name[i]);
                        pstmt.setString(3, addr[i]);
                        pstmt.executeUpdate();
                }
                        Statement stmt = conn.createStatement();
        ResultSet rset = stmt.executeQuery("SELECT id, name, addr FROM customer");
                while(rset.next()) {
                        System.out.println("id: " + rset.getInt(1));
                        System.out.println("name: " + rset.getString(2));
                        System.out.println("addr: " + rset.getString(3));
                }
                } catch (SQLException e) {
                        System.out.println(e.getMessage());
                } finally {
                        if(conn != null) try
                            { conn.close();
                          } catch (SQLException e) {}
                }
        }
}
```

PreparedStatment를 활용 SQL준비

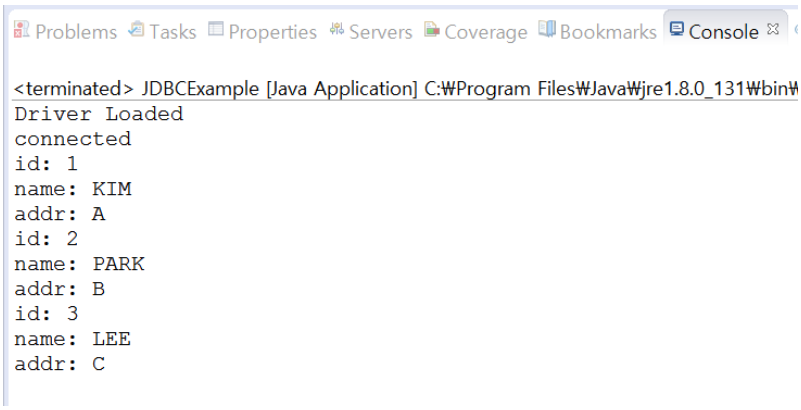
Statment를 활용 SQL준비

결과값 가져오기

DB 연결 해제

[소스 9-6] 자바 SQL을 활용한 DB프로그래밍 예제

Run As를 선택하여 자바 어플리케이션을 실행시키면 Console에 다음과 같이 결과가 출력된다.



[그림 9-8] 예제 실행 결과

9.3 Data Source

DB에 연결하기 위해서 연결정보를 매번 입력하고 getConnection()명령어를 실행하여 connection을 생성하는 것은 매우 번거롭다. Spring 프레임워크에서는 연결에 필요한 정보를 포함하여 DataSource Bean을 독립적인 객체로 생성하여 쉽게 DB연결이 가능하도록 지원한다. DataSource Bean은 다음과 같이 설정한다. 사용하고자 하는 DB에 맞는 드라이버명, 접속정보, 사용자, 패스워드 정보가 필요하며, ${driver}의 형태로 표시된 것은 별도 파일인 properites파일에서 값을 읽어오는 것이다.

```
<bean id="dataSource"
          class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="${driver}" />
    <property name="url" value="${dburl}" />
    <property name="username" value="${username}" />
    <property name="password" value="${password}" />
 </bean>
```

[소스 9-7] Spring DataSource Bean 설정

Spring DataSource의 경우 연결이 필요한 경우 새롭게 세션을 생성하여 연결하게 되는데, 이경우 연결에 소요되는 시간이 많이 소요되기 때문에 전체적인 DB처리 성능이 저하되게 된다. 새롭게 DB연결을 하게 되면 TCP/IP에 의해서 연결에 필요한 정보를 교환하게 되어 결과적으로 SQL에 소요되는 시간보다 새로운 연결을 생성하는데 더 많은 시간이 소요될 수 있다. 이러한 문제점을 해결하기 위해서 DB connection pool을 활용하는 것이 일반적이다. 오픈소스 DB connection pool 중에 Apache의 DBCP가 많이 활용되고 있다. 실제 운영을 위해서 DB 성능은 미리 초기 및 최대 풀의 수와 관련이 있으며, 하드웨어 성능에 따라 최적의 값을 찾아내어 활용하는 것이 필요하다. 아래는 DBCP를 활용하기 위한 dataSource bean설정 방법이다.

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
<property name="driverClassName" value="${driver}" />
    <property name="url" value="${dburl}" />
    <property name="username" value="${username}" />
    <property name="password" value="${password}" />
    <property name="defaultAutoCommit" value="true"/>
    <property name="initialSize" value="5"/>
    <property name="maxActive" value="30"/>
    <property name="maxIdle" value="5"/>
    <property name="maxWait" value="30000"/>
    <property name="validationQuery" value="SELECT 1 FROM DUAL"/>
    <property name="testOnBorrow" value="true"/>
    <property name="testOnReturn" value="false"/>
    <property name="testWhileIdle" value="true"/>
    <property name="timeBetweenEvictionRunsMillis" value="60000"/>
</bean>
```

[소스 9-8] DBCP DataSource Bean 설정

설정된 Spring 또는 DBCP DataSource Bean은 @Resuource나 @Autowired를 활용하여 DI를 통한 의존성을 생성하여 쉽게 활용할 수 있다.

```
@Resource(name = "dataSource")
DataSource dataSource;

@Test
public void testJdbcDataSource() throws Exception {
    Connection con = null;
    try {
        con = dataSource.getConnection();
        stmt = con.createStatement();
        rs = stmt.executeQuery("select 'x' as x from dual");
        while (rs.next()) {
            assertEquals("x", rs.getString(1));   .....
    }
}
```

[소스 9-8] DataSource 활용예제

Weblogic이나 JEUS와 같은 상용 WAS를 활용하는 경우에는 WAS에서 제공하는 connection pool을 활용하는 것이 바람직하다. JNDI(Java Naming Directory Interface)의 형태로 지정된 자원을 접근할 수 있도록 설정할 수 있다.

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="jndiName" value="XACOM"/>
<property name="jndiTemplate" ref="jnditemplate" />
</bean>

<bean id="jnditemplate" class="org.springframework.jndi.JndiTemplate" >
<property name="environment"> <props>
<prop key="java.naming.factory.initial"> jeus.jndi.JNSContextFactory </prop>
<prop key="java.naming.provider.url">localhost:9736</prop>
</props> </property> </bean>
```

[소스 9-9] JEUS에서 JNDI활용 Connection Pool 설정

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="jndiName"> <value>oracle22Jndi</value>
</property> <property name="jndiTemplate"><ref local="dataSourceTemplate" />
</property>
</bean>

<bean id="dataSourceTemplate" class="org.springframework.jndi.JndiTemplate">
<property name="environment"> <props>
<prop key="java.naming.provider.url"> t3://localhost:7001 </prop>
<prop key="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</prop>
</props> </property>
</bean>
```

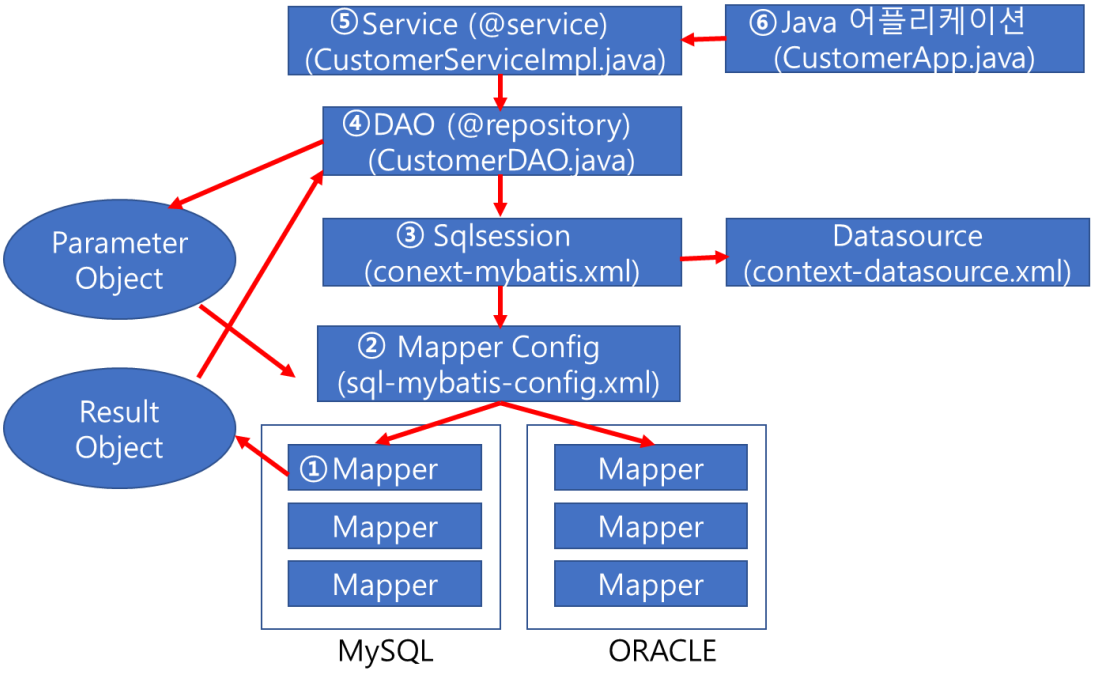
[소스 9-10] Weblogic에서 JNDI활용 Connection Pool 설정

9.4 Data Access (MyBatis)

Java SQL을 활용하여 DB프로그래밍을 할 경우에는 connection 객체 생성과 연결, 수행할 SQL의 준비와 실행, ResultSet을 활용한 결과처리 등 DB 처리가 될 때마다 비슷한 형태의 소스코드를 작성해야 한다. 개발자가 작성하는 소스코드가 많아 지면서 생산성도 떨어지게 되고 중요한 비즈니스 로직에 집중하지 못하는 문제가 생긴다. 그리고, SQL문이 소스코드에 위치하기 때문에 SQL문 수정이 필요한 경우에는 소스코드를 수정하고 빌드와 배포를 다시 해야 하는 번거로움이 있다. 또한 작업 종료시에 자원 반환 등의 처리를 실수로 누락하는 경우 당장 에러 등의 문제가 발생하지 않지만 리소스 손실 등으로 운영 중에 갑자기 서버에 장애가 생길 수도 있다. MyBatis를 활용하게 되면 실행할 SQL문은 XML로 작성된 설정파일 형태로 별도 저장된다. 따라서 SQL문을 수정할 경우 소스코드는 그대로 유지하고 XML파일만 바꾸면 되므로 변경이 수월 해진다. 또한 connection 연결 및 SQL문의 준비와 수행, 입력 파리미터 및 ResultSet의 매핑 등을 모두 MyBatis에서 처리하기 때문에 개발자가 처리해야 할 부분이 적어져서 개발 생산성이 높아지고, 개발자의 실수에 의한 장애 발생 가능성도 줄어들게 된다. 만약 여러 DB 활용을 고려하여 프로그래밍을 해야 할 경우 Java SQL을 활용하는 경우 DB에 따라 별도 프로그램을 작성해서 관리해야 되기 때문에 불편하다. MyBatis의 경우 DB별로 실행 할 SQL문을 XML파일로 작성하고 설정파일에 활용 할 DB정보를 설정해주어 DB를 쉽게 변경 실행 할 수 있다. 소스코드가 특정 DB에 종속되지 않기 때문에 소스코드의 호환성을 높여주고 관리도 쉽게 할 수 있게 된다. MyBatis를 활용하기 필요한 구성요소로는 SQL문을 XML형태로 관리하는 Mapper파일이 있고, MyBatis 설정을 하는 Mapper Config 파일이 있다. Parameter Object는 SQL문의 where절이나 INSERT, UPDATE 등에 필요한 값을 전달 하게 된다. 클래스, Map 모두 활용할 수 있는데 주로 클래스를 활용한다. Result Object는 SELECT 수행 결과 값을 오브젝트에 매핑하여 DAO로 전달하게 된다. 마찬가지로 Parameter Object와 같이 클래스, Map 모두 활용 가능하다. Mybatis와 Spring이 연결되어 동작하기 위해서 Spring의 sqlSessionFactory 빈을 활용하고, 표준프레임워크에서는 context-mybatis.xml의 형태로 설정한다. 연결에 필요한 datasource 정보와 mapper config 파일 위치를 지정한다.

[표 9-1] MyBatis 구성요소

구성요소	설명
MapperConfig XML File	MyBatis 동작을 위한 기본적인 설정을 공통으로 정의
Mapper XML File	실행할 SQL문과 매핑 정보를 XML 방식으로 정의
Parameter Object	SQL문의 조건절에서 값을 비교하거나 INSERT, UPDATE 등에서 필요한 입력값을 받아오기 위한 Object로 멤버 변수와 같은 이름의 변수를 SQL문에 활용하여 매핑
Result Object	SQL 실행결과를 리턴하기 위한 Object



[그림 9-9] MyBatis 구성

MyBatis를 활용하여 개발을 하기 위한 방법은 다음과 같다.

1) JDBC 드라이버 설치

Maven 환경에서 DB 접속을 위해 pom.xml에 DB 드라이버의 dependency 설정이 필요하다. MySQL 드라이버의 dependency를 다음과 같이 설정한다.

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.42</version>
</dependency>
```

[소스 9-11] MySQL JDBC 드라이버 dependency 설정

2) SQL Mapper XML파일

먼저 SQL mapper XML파일을 작성해야 한다. Mapper XML파일에는 SQL마다 고유 id를 가지게 되며, DAO에서 SQL id를 호출하고 입력 값을 위한 Parameter 클래스를 전달하여 실행된다. Parameter Object는 mapper config에서 설정한 alias를 활용하는 것이 편리한다. 입력 값은 SQL안에 "#{id}"의 형태로 설정되게 되며, 매핑을 위해서는 parameter Object의 멤버변수와 같게 이름을 적어야 한다. "id", "name", "address"의 멤버변수를 가지는 customerVO 클래스를 parameter Object로 활용한다면, SQL안에 입력 값을 매핑할 때 #{id}, #{name}, #{address}와 같이 기술하면 된다. 결과 값은 ResultMap으로 설정된 Result Object를 통해서 전달 받게 되며, mapper파일 안에서 SQL실행 결과 컬럼 값과 Result Object 멤버 변수와 매핑 정보가 기술된다. Parameter Object와 마찬가지로 Result

Map은 mapper config에서 설정한 alias를 기술한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="customer">

        <resultMap id="customerResult" type="customerVO">
                <id property="id" column="id" />
                <result property="name" column="name" />
                <result property="address" column="addr" />
        </resultMap>

        <insert id="insertCustomer" parameterType="customerVO">
                        <![CDATA[
                                insert into customer (id, name, addr)
                                values   (#{id}, #{name}, #{address})
                        ]]>
        </insert>

        <select id="selectCustomerList" parameterType="customerVO"
                resultMap="customerResult">
                        <![CDATA[
                                select   id, name, addr
                                from     customer
                        ]]>
        </select>
</mapper>
```

SELECT 실행결과 값을 전달 받을 클래스 매핑

SQL실행결과 컬럼과 클래스 멤버변수 매핑

Id: “insertCustomer”, Parameter: customerVO

Parameter 클래스의 멤버변수를 입력값으로 설정

결과값 Result Map id 명시

[소스 9-12] Mappers (lab-dao-class.xml) 설정 예시

3) SQL Mapper XML파일

MyBatis 설정을 위해 필요한 파일 이며, property, type alias, mappers, settings와 같은 정보를 가진다. 예제에서는 type alias와 mappers 관련 정보만 설정하여 활용한다. Type alias는 mapper XML에서와 같이 Parameter Object 정보를 모두 기술하면 너무 길기 때문에 실제 클래스 정보는 Config 파일에만 기술하여 활용할 수 있도록 지원한다. 또 한가지는 Mappers 파일의 위치를 등록한다. 파일의 위치는 * 등을 쓰지 않고, 정확하게 기술을 해주어야 한다. Sqlsession에서는 mappers 파일의 위치를 * 등을 활용하여 설정 할 수 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-config.dtd">

 <configuration>
    <typeAliases>
        <typeAlias alias="customerVO" type="lab.CustomerVO" />
    </typeAliases>
```

Parameter Object, Result Object에 활용될 Object alias 지정

Mapper 파일 위치 설정

```
    <mappers>
        <mapper resource ="META-INF/sqlmap/mappers/lab-dao-class.xml" />
    </mappers>
</configuration>
```

[소스 9-13] Mapper configuration (sql-mybatis-config.xml) 설정 예시

Settings 정보는 아래와 같이 MyBatis 설정을 위해 활용 된다. 아래 예시에 기술되어 있는 value값은 default 값이며, 설정이 생략되는 경우에는 자동으로 default 값을 가진다.

```
<settings>
  <setting name="cacheEnabled" value="true"/>
  <setting name="lazyLoadingEnabled" value="false"/>
  <setting name="multipleResultSetsEnabled" value="true"/>
  <setting name="useColumnLabel" value="true"/>
  <setting name="useGeneratedKeys" value="false"/>
  <setting name="autoMappingBehavior" value="PARTIAL"/>
  <setting name="autoMappingUnknownColumnBehavior" value="NONE"/>
  <setting name="defaultExecutorType" value="SIMPLE"/>
  <setting name="defaultStatementTimeout" value="null"/>
  <setting name="defaultFetchSize" value="null"/>
  <setting name="safeRowBoundsEnabled" value="false"/>
  <setting name="mapUnderscoreToCamelCase" value="false"/>
  <setting name="localCacheScope" value="SESSION"/>
  <setting name="jdbcTypeForNull" value="OTHER"/>
  <setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>
</settings>
```

[소스 9-14] Mapper configuration settings 예시

4) Spring 설정

Spring과 MyBatis 연동을 위한 설정으로 sqlSession 객체를 생성하고 관리하며, DB연결에 필요한 Datasource, Mapper configuration 파일의 위치를 설정한다. Mapper location을 일괄로 지정할 수도 있다. 이경우 Mapper configuration의 mapper 위치와 중복지정 할 수 없다.

```
<?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-4.0.xsd">

<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation"
            value="classpath:/META-INF/sqlmap/sql-mybatis-config.xml" />
```

DB연결 설정(DBCP 등)

MyBatis 설정

```
    <property name="mapperLocations"
            value="classpath:/META-INF/sqlmap/mappers/lab-*.xml" />
</bean>

</beans>
```

Mapper 파일 위치 (* 활용 가능하며, Matis 설정의 위치 설정과 중복 불가)

[소스 9-15] sqlSessionFactoryBean(context-mybatis.xml) 설정 예시

Annotation방식의 DI설정을 위하여 component scan 설정을 하였고, DB properties를 파 파일에서 읽을 수 있도록 properties를 지정하였다.

```
<?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-4.0.xsd
                                    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:property-placeholder
                        location="classpath:/META-INF/spring/jdbc.properties" />
    <context:component-scan base-package="lab"/>
</beans>
```

[소스 9-16] Spring bean (context-common.xml) 설정 예시

DB연결을 하기 위한 DBCP설정과 jdbc properties 설정은 다음과 같다.

```
<?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:jdbc="http://www.springframework.org/schema/jdbc"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-4.0.xsd">

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
     destroy-method="close">
                <property name="driverClassName" value="${driver}" />
                <property name="url" value="${dburl}" />
                <property name="username" value="${user}" />
                <property name="password" value="${password}" />
        </bean>
</beans>
```

[소스 9-17] DBCP (context-datasource.xml) 설정 예시

DBCP 설정에 변수 처리되어 있는 값은 아래의 값으로 치환되어 실행 된다. 만약 DB를

다르게 설정하였거나 별도 DB를 활용하는 경우 설정을 변경하여 실행한다.

```
driver=com.mysql.jdbc.Driver
dburl=jdbc:mysql://127.0.0.1:3306/com
user=com
password=com01
```

[소스 9-18] DB 연결 속성 (jdbc.properties)

5) DAO 작성

EgovAbstractMapper 클래스를 상속받아 DAO 클래스를 작성한다. EgovAbstractMapper 클래스는 SqlSessionDaoSupport의 하위클래스로 SqlSession 설정과 메소드 호출의 편리함을 제공한다. EgovAbstractMapper를 쓰지 않으면, insert, update, delete, select 등을 실행할 때 앞에 " getSqlSession().insert()"의 형태로 써주어야 되기 때문에 불편하다.

```
@Repository("customerDAO")
public class CustomerDAO extends EgovAbstractMapper {

    public void insertCustomer(CustomerVO vo) {
        insert("customer.insertCustomer", vo);
    }

    public List<CustomerVO> selectCustomerList(CustomerVO vo) {
        return selectList("customer.selectCustomerList", vo);
    }


}
```

Mapper id와 parameter object 명시하여 mapper INSERT 실행

Mapper id와 parameter object 명시하여 mapper SELECT 실행

[소스 9-19] DAO (CustomerDAO.java) 작성

Parameter 및 Result Object로 활용되는 CustomerVO는 다음과 같이 id, name, address 세개의 멤버변수를 가지는 클래스이다.

```
public class CustomerVO {
        String id;
        String name;
        String address;
}
```

[소스 9-20] Parameter, Result Object (CustomerVO.java) 작성

6) Service 및 Java 어플리케이션 작성

Controller, JUnit, Java 어플리케이션에서 실행 시킬 수 있도록 서비스를 작성한다. 일반적으로 트랜잭션 및 비즈니스 로직 처리를 위한 서비스를 정의, 실행 시키는 것이 일반적이며, DAO를 바로 호출하지 않는다. 현재는 비즈니스 로직이 없기 때문에 괜히 한 계층을 더 호출하는 것 같아 불필요해 보일 수 있어도, 실제 복잡한 업무를 구현하는 환경에서 유연한 처리와 모듈의 재사용성을 높이기 위해서 필요하다.

```
@Service("customerService")
public class CustomerServiceImpl implements CustomerService {

    @Resource(name = "customerDAO")
    public CustomerDAO customerDAO;

        public void insertCustomer(CustomerVO customerVO) throws Exception {
                customerDAO.insertCustomer(customerVO);
        }

        public List<CustomerVO> selectCustomerList(CustomerVO customerVO)
          throws Exception {
                return customerDAO.selectCustomerList(customerVO);
        }
}
```

[소스 9-21] Service (CustomerServiceImpl.java) 작성

구현된 DAO와 Service를 테스트하기 위해 값을 INSERT하고 SELECT하는 Java 어플리케이션을 구현하여 실행하였다.

```
public class CustomerApp {
        public static void main(String[] args) throws Exception {
            String configLocation = "classpath*:META-INF/spring/context-*.xml";
            ApplicationContext context
                    = new ClassPathXmlApplicationContext(configLocation);

          CustomerService customer
                    =(CustomerService)context.getBean("customerService");

            vo.id = "1";
            vo.name = "KIM";
            vo.address = "SEOUL";
            customer.insertCustomer(vo);

            vo.id = "2";
            vo.name = "LEE";
            vo.address = "PUSAN";
            customer.insertCustomer(vo);

          List<CustomerVO> resultList= customer.selectCustomerList(vo);
            int num = resultList.size();
            for (int i=0; i<num; i++ ) {
                    CustomerVO resultvo = resultList.get(i);
                    System.out.println("id="+ resultvo.id);
                    System.out.println("name="+ resultvo.name);
                    System.out.println("address="+ resultvo.address);
            }
} }
```

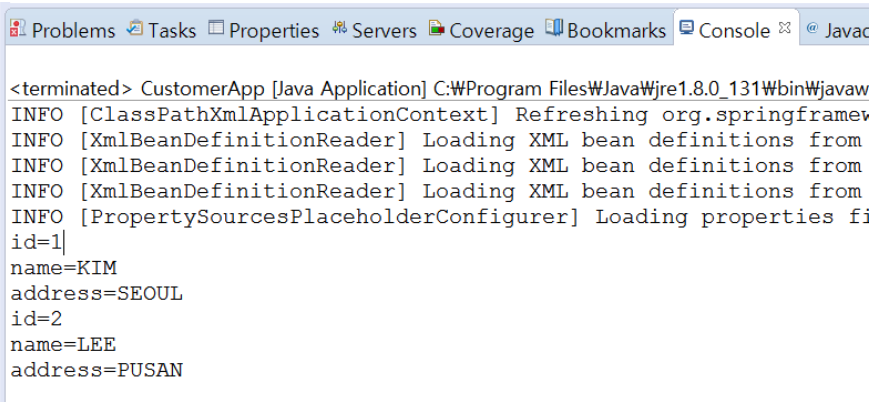
Service를 DI설정 (@Service 설정 매칭)

두개의 값을 INSERT

입력된 값을 SELECT

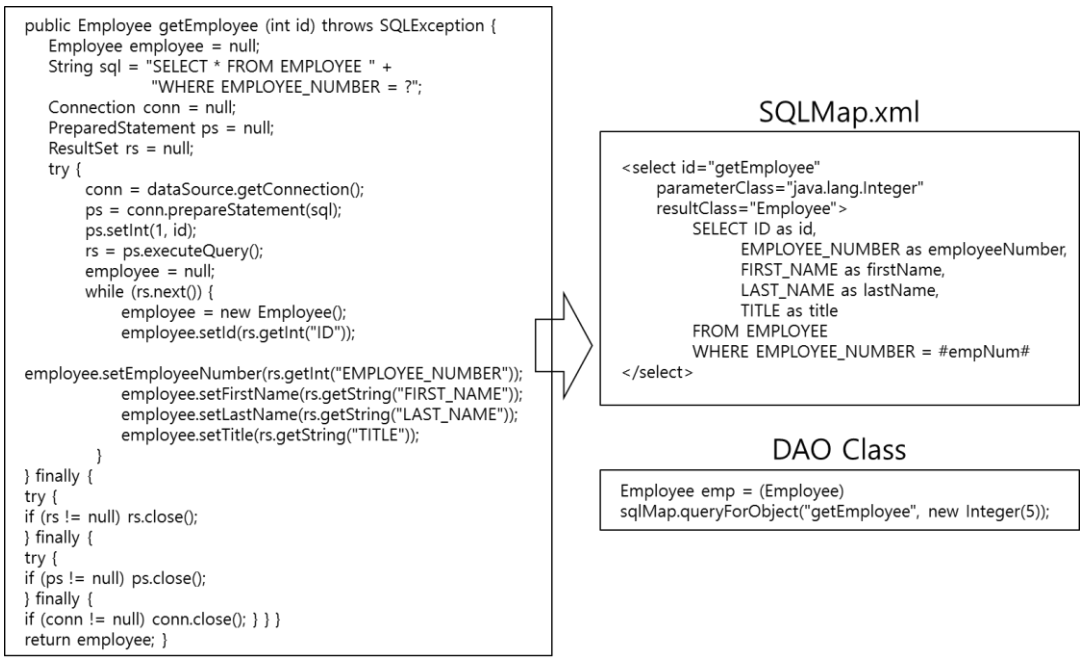
[소스 9-22] Java 어플리케이션 (CustomerApp.java) 작성

구현된 Java 어플리케이션을 Run AS로 실행하면 다음과 같은 결과를 얻을 수 있다.



[그림 9-10] 실행결과

앞에서 살펴보았던 Java SQL을 활용하여 프로그래밍을 하는 것과 MyBatis를 활용하는 것을 비교해보면, DB연결, SQL문 준비, 실행 후에 ResultSet처리, 리소스 반환 등 개발자가 처리해야 될 것이 많고, 프로그램이 늘어날수록 비슷한 형태의 코드가 계속 늘어나게 되고, 반복적으로 개발하게 된다. MyBatis의 경우는 SQL들은 모두 설정파일로 들어가게 되고, 실행시에는 mapper에 정의된 SQL id와 parameter object를 전달하여 실행하면 되기 때문에 간단하게 구현과 실행이 가능하며, 많은 업무를 처리 할 때도 소스코드 수가 아주 많아지지 않고, 코드들이 정형화 되어 유지보수가 편리하다. 다음 그림에서 불필요하고 반복적인 소스코드를 개발할 필요가 없이 SQL을 정의해주고, 실행만 시켜주면 다른 부분들은 MyBatis가 처리하게 된다.



[그림 9-11] Java SQL과 MyBatis 소스코드 비교

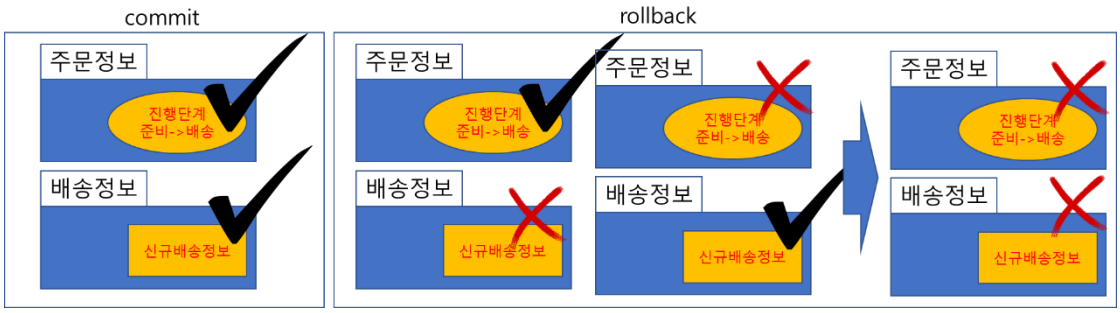
앞의 예제에서 SQL문은 XML에 정적으로 위치하고 있는데, 개발을 하다 보면 조건에 따라 SQL문이 바뀌어야 되는 경우가 있다. 특히 검색 같은 경우 검색 조건에 따라서 SQL문이 바뀌어야 된다. 이러한 처리를 가능하도록 하기 위해 MyBatis에서 Dynamic SQL을 지원한다. If는 가장 많이 사용되는 요소로 조건 결과에 따라서 해당 SQL을 포함하거나 미포함 할 수 있도록 조정할 수 있다. 아래 예제에서는 empNo가 null이 아니거나, empName이 null이 아닌 경우 where절에 조건이 포함되게 된다. 이때 trim을 활용하면, 반복되는 문자를 자동으로 제거 할 수 있다. Where절 다음 앞에 나오는 AND, OR 등의 문자를 제거할 수 있으며, 조건을 모두 만족하지 않으면 where절도 포함되지 않는다. 이외에도 foreach, choose 등을 활용하여 Dynamic SQL을 생성할 수 있다.

```
<select id="selectEmpList" parameterType="empVO" resultType="empVO">
      select EMP_NO as empNo, EMP_NAME as empName
      from EMP
   <trim prefix="WHERE" prefixOverrides="AND|OR ">
     <if test="empNo != null">
          EMP_NO = #{empNo}
     </if>
     <if test="empName != null">
          AND EMP_NAME LIKE '%' || #{empName} || '%'
     </if>
   </trim>
  </select>
```

[소스 9-23] IF를 활용한 Dynamic SQL

9.5 Transaction 처리

Transaction처리는 DB 데이터의 정합성을 위해서 반드시 필요한 개념이다. 예를 들어 쇼핑몰에서 고객이 주문한 상품을 배송하려고 할 때. 시스템에서 진행 단계를 배송 상태로 수정하는 처리와 신규로 배송정보를 입력하는 두 가지의 데이터 처리가 있다고 가정해보자. 만약 처리 단계를 수정하는 단계에서 에러가 발생되었는데 배송정보가 입력이 되었다면, 배송 단계가 아닌 데도 배송처리가 된다. 올바른 결과를 가지려면 두 가지 처리가 모두 성공해야만 전체적으로 처리가 되는 것이고, 두 개 중에 하나라도 오류가 발생되면 이미 성공한 것도 실패한 것으로 처리 되어 아무것도 수행하지 않은 이전의 상태로 되돌려야 한다. 이렇게 데이터 정합성을 관리하는 것을 transaction 처리라고 한다. 하나의 transaction안에 포함된 DB처리가 모두 성공하게 되면 commit이 되고, 하나라도 실패하면 원래의 상태로 돌아가게 되며 이를 rollback이라고 한다.



[그림 9-12] Transaction 개념

Transaction을 활용하기 위해서는 먼저 서비스를 제공할 transaction manager를 결정해야한다. Transaction Manager는 두 가지가 있는데, Spring에서 제공하는 datasource 기반의 transaction manager를 사용하여 관리하는 방법과 WAS에서 제공하는 JTA(Java Transaction API)를 활용하는 방법이 있다. Datasource를 활용하는 방법은 다음과 같이 transaction manager bean을 생성하면 된다.

```
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

[소스 9-24] Datasource기반 transaction manager 활용

JTA를 활용하는 방법은 WAS에서 transaction manager를 설정하고 이를 JNDI로 연결하여 활용하는 방식으로 앞에서 설명했던 connection pool을 활용하는 방법과 비슷하다. 웹 로직에서 JTA를 활용하여 transaction manager를 설정하는 예시는 다음과 같으며, jndi-name이나 provider url은 설정한 환경에 맞추어 변경을 해야 한다. Datasource transaction 서비스와 달리 별도 transaction manager bean을 정의하지 않아도 된다. 상용 WAS를 활용하는 경우 WAS에서 제공하는 JTA를 활용하는 것이 좋다.

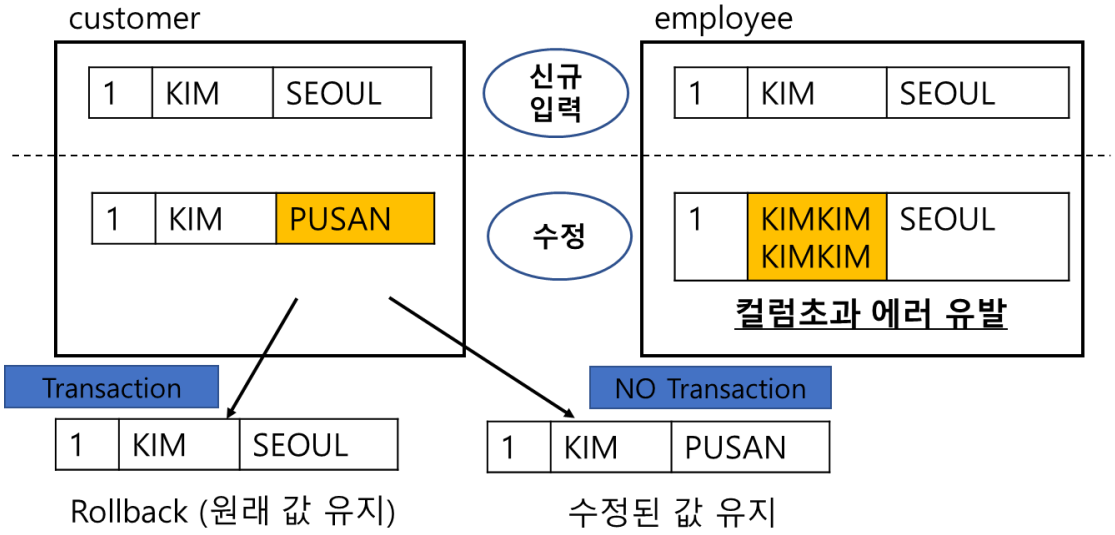
```
<tx:jta-transaction-manager />
<jee:jndi-lookup id="dataSource" jndi-name="dbmsXADS"
    resource-ref="true">
    <jee:environment>
        java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
        java.naming.provider.url=t3://was:7002
      </jee:environment>
</jee:jndi-lookup>
```

[소스 9-25] JTA 기반 transaction manager 활용

Transaction manager가 설정되면 소스코드에서 transaction을 처리하는 방식을 결정해야한다. 첫번째는 Annotation을 활용하는 방법, 두번째는 AOP를 활용하는 방법, 세번째는 직접 코드에서 transaction API를 호출하는 방법이 있다. 세가지 방법 중에 어느 것을 선택하

여도 결과는 같으며, 활용하기 편리한 것으로 결정하면 된다. AOP를 활용하면 XML로 advice와 point cut설정만 하면 소스코드에 아무런 설정이 없어도 transaction이 적용되어 편리하다.

Transaction 설정 테스트를 위해서 추가로 employee테이블을 생성한 후에 동일한 값("1","KIM","SEOUL")을 INSERT 한다. 다음으로 customer 테이블의 값에서 "SEOUL"을 "BUSAN"으로 수정하고, employee는 수정시에 컬럼 수를 초과로 값을 입력하여 고의로 에러를 유발한다. 트랜잭션 처리를 한 상태에서 결과를 출력하면 rollback 처리가 되어 수정 이전의 값인 "SEOUL"을 유지하게 된다. 트랜잭션 처리를 하지 않고 결과를 출력하면 수정된 값이 그대로 유지되어 "PUSAN"이 출력되게 된다.



[그림 9-13] Transaction 테스트 시나리오

먼저, 앞에서 생성한 customer테이블을 컬럼을 그대로 유지하고 이름만 employee로 변경하여 새로 생성하였다.

```
create table employee (
id varchar(10) primary key,
name varchar(10),
addr varchar(10));
```

[소스 9-26] employee 테이블 생성

소스코드는 예제 9-2에 신규로 생성된 employee 테이블 처리를 위한 mapper 및 DAO를 신규로 생성하고, transaction이 적용되도록 설정을 추가하였다. 먼저 annotation 기반으로 transaction 설정을 위하여 spring 설정파일인 context-tranactoin.xml을 다음과 같이 추가하였다.

```
<bean id="txManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
```

annotation활용을 위한 설정

```
</bean>

<tx:annotation-driven transaction-manager="txManager" />
```

[소스 9-27] transaction 설정 (context-transaction.xml)

그 다음에 신규로 생성된 employee 테이블에 데이터 입력, 수정, 삭제를 위한 mapper를 신규로 생성하였다. 형태는 컬럼이 동일하기 때문에 앞의 customer mapper와 비슷하다. Result Object, Parameter Object는 그대로 활용이 가능하며, SQL문과 SQL id를 수정해야 된다. 마찬가지 방법으로 Employee DAO도 생성한다. 서비스(CustomerServiceImpl)에 transaction처리를 적용하기 위해 설정을 추가한다. 업데이트 처리하는 updateCustomer 메소드에 "@Transactionl"를 적용하였고, 두 개 테이블에 수정하도록 각각 DAO의 update 메소드를 호출한다. employeeDAO는 에러를 유발하기 위해 parameter object를 새로 정의하여 전달 한다.

```
@Service("customerService")
public class CustomerServiceImpl implements CustomerService {

    @Resource(name = "customerDAO")
    public CustomerDAO customerDAO;

    @Resource(name = "employeeDAO")
    public EmployeeDAO employeeDAO;

        public void insertCustomer(CustomerVO customerVO) throws Exception {
                customerDAO.insertCustomer(customerVO);
                employeeDAO.insertEmployee(customerVO);
        }


        public void deleteCustomer(CustomerVO customerVO) throws Exception {
                customerDAO.deleteCustomer(customerVO);
                employeeDAO.deleteEmployee(customerVO);
        }

        @Transactional
        public void updateCustomer(CustomerVO customerVO) throws Exception {
                customerDAO.updateCustomer(customerVO);
                CustomerVO vo = new CustomerVO();
                vo.id = "1";
                vo.name = "KIMKIMKIMKIMKIM";
                vo.address = "SEOUL";
                employeeDAO.updateEmployee(vo);
        }

        public List<CustomerVO> selectCustomerList(CustomerVO customerVO)
                throws Exception {
```

DAO연결 위한 DI 설정

Transaction 적용 (메소드 전체)

에러유발을 위해 값을 재정의하고, DAO 호출

```
                return customerDAO.selectCustomerList(customerVO);
        }

        public List<CustomerVO> selectEmployeeList(CustomerVO customerVO)
                throws Exception {
                return employeeDAO.selectEmployeeList(customerVO);
        }
}
```

[소스 9-28] 서비스(CustomerServiceImpl.java)에 transaction 적용

```
public class CustomerApp {
        public static void main(String[] args) {
                String configLocation = "classpath*:META-INF/spring/context-*.xml";
                ApplicationContext context =
                                new ClassPathXmlApplicationContext(configLocation);

                CustomerService customer=
                                (CustomerService)context.getBean("customerService");
                List<CustomerVO> resultList;
                CustomerVO vo = new CustomerVO();
                try {
                customer.deleteCustomer(vo);

                vo.id = "1";
                vo.name = "KIM";
                vo.address = "SEOUL";
                customer.insertCustomer(vo);

                resultList= customer.selectCustomerList(vo);
                printResult(resultList);

                vo.id = "1";
                vo.name = "KIM";
                vo.address = "PUSAN";
                customer.updateCustomer(vo);

                } catch(Exception e) {
                        e.printStackTrace();
                }
                finally {
                        try {
                                resultList= customer.selectCustomerList(vo);
                                printResult(resultList);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }
```

INSERT 입력 값 세팅 및 INSERT

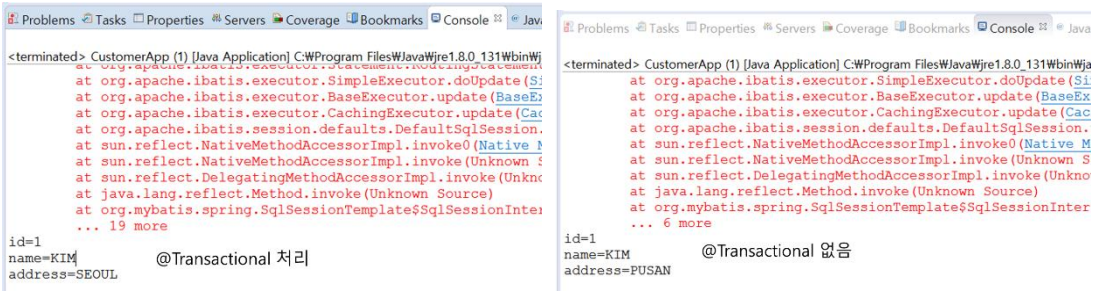
UPDATE 입력 값 세팅 및 UPDATE

에러후에 결과 출력을 위해 finally에서 처리

```
        public static void printResult(List<CustomerVO> resultList) {
                int num = resultList.size();
                for (int i=0; i<num; i++) {
                        CustomerVO resultvo = resultList.get(i);
                        System.out.println("id="+ resultvo.id);
                        System.out.println("name="+ resultvo.name);
                        System.out.println("address="+ resultvo.address);
                }
        }
}
```

[소스 9-29] 실행을 위한 어플리케이션 구현 (CustomerApp.java)

Java 어플리케이션을 실행하여 결과를 확인해 보면 다음과 같다. Service에서 Transaction을 적용하기 위해 @Transactional을 설정하고 실행하면 rollback 처리가 되어 이전의 "SEOUL"을 유지하게 되고, @Transactional을 주석처리 한 후에 실행하면 transaction처리가 되지 않기 때문에 수정된 "PUSAN" 값이 그대로 출력된다.



[그림 9-14] Java 어플리케이션 실행결과 확인

앞서 설명한 transaction 설정 방법은 Service 메소드별로 @Transactional로 annotation 설정이 필요하다. 이러한 설정이 필요 없이 자동으로 transaction처리를 하려면 AOP를 활용하여 처리가 가능하다. 아래와 같이 AOP에 설정을 추가해 주면 lab 패키지 밑에 Impl로 끝나는 모든 클래스의 모든 메소드에 transaction이 설정된다.

```
<aop:config>
    <aop:pointcut id="requiredTx"
        expression="execution(* lab.*Impl.*(..))" />
    <aop:advisor advice-ref="txAdvice" pointcut-ref="requiredTx" />
</aop:config>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />
        <tx:method name="createNoRBRole" no-rollback-for="NoRoleBackTx" />
        <tx:method name="createRBRole" rollback-for="RoleBackTx" />
        <tx:method name="create*" />
    </tx:attributes>
</tx:advice>
```

[소스 9-30] AOP를 활용한 transaction 설정

이외에도 transactionTemplate을 활용하여 개발자가 직접 transaction 영역을 지정하는 방법도 있다. Transaction manager 및 transactionTemplate 빈을 설정한다.

```
<bean id="transactionTemplate"
    class="org.springframework.transaction.support.TransactionTemplate">
    <property name="transactionManager" ref="transactionManager" />
</bean>
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

[소스 9-31] transactionTemplate 설정

앞에서 설정된 TransactionTemplate 빈을 활용하여, execute()로 정의된 영역에 transaction이 설정된다.

```
public void testInsertCommit() throws Exception {
   transactionTemplate.execute(new TransactionCallbackWithoutResult() {
      public void doInTransactionWithoutResult(TransactionStatus status) {
         try {
            …
         } catch (Exception e) {
            status.setRollbackOnly();
         }
      }
   });
```

[소스 9-32] transactionTemplate를 활용한 programmatic 방법

10. MVC(Model View Controller)

소스코드에서 화면과 비즈니스 로직이 분리되지 않게 되면 중복된 코드가 증가하게 되고, 코의 변경도 어렵게 된다. 화면이 증가하게 되면 모듈도 증가하는 형태여서 반복적인 코딩이 많아지고 복잡하게 된다. MVC는 모듈을 역할별로 나누어 구성하여 소스코드를 체계적으로 구성되도록 하고, 소스코드의 변경도 유연해지게 된다.

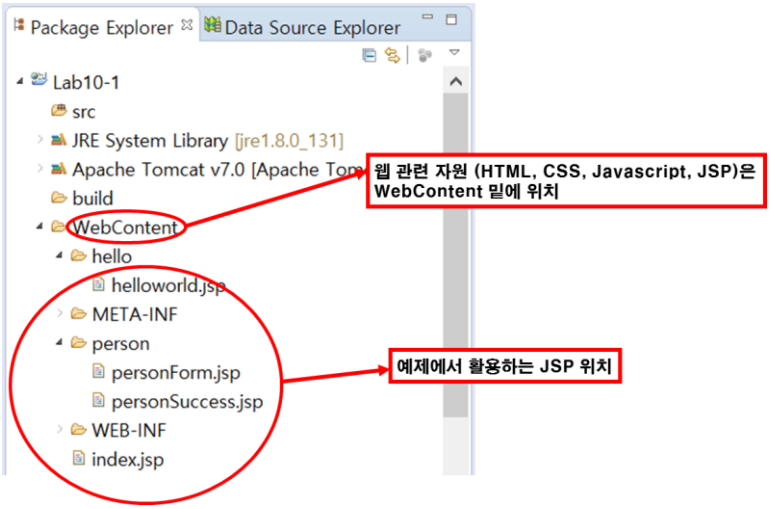
10.1 JSP 웹 프로그래밍

웹 프로그래밍의 초창기에는 기존 프로그램 언어를 활용하여 웹으로 연동을 지원하기 위해서 printf(C언어)와 같은 명령을 활용하여 HTML코드를 출력시키는 CGI방식을 활용하였다. Java의 경우도 JSP가 등장하기 전에는 Setvlet을 활용하여 println으로 HTML을 출력하는 형태로 웹 프로그래밍을 하였다. 디자이너가 코딩한 HTML을 일일이 개발자가 print문으로 바꾸어 주어야 했으며, 따옴표 같은 경우 그대로 쓰면 syntax에러가 나기 때문에 백슬래시(\)를 앞에 붙여줘야 했다. 아래 코드는 C를 활용한 CGI 프로그램 소스코드 일부이다.

```
void print_rows() {
    printf("<tr><td>%d</td><td>%s</td><td>%8.2f</td><td>%d</td></tr>₩n",
        emp_rec.emp_number, emp_rec.emp_name,
        emp_rec.salary, emp_rec.dept_no);
}
```

[소스 10-1] C로 구현된 CGI예제

JSP는 HTML을 그대로 활용하면서 프로그램에 해당되는 영역은 "<% … %>" 사이에 입력할 수 있도록 지원하고 있어 CGI나 Servlet에 비해 웹 프로그래밍을 쉽게 할 수 있다. Lab10-1은 입력 폼을 통해 정보를 입력 받아 다른 JSP에 정보를 전달하여 보여주는 어플리케이션이다. JSP로만 구현되어 있어 Dynamic Web Project로 구성되어 있다. 프로젝트의 디렉터리 구성은 아래 그림과 같다. WebContent 아래에 웹 자원이 위치하게 된다.



[그림 10-1] 프로젝트 디렉터리 구성

[표 10-1] Lab10-1 구성

구분	코드명	설명
WebContent	index.jsp	Default 실행을 위한 JSP
	hello/helloworld.jsp	Hello World 출력 예제
	personForm..jsp	신상 정보를 입력하는 폼
	personSuccess.jsp	폼에서 입력 받은 정보를 출력

정보입력 폼을 출력하는 personForm.jsp와 폼으로부터 정보를 전달 받아 내용을 출력하는 personSuccess.jsp 소스코드는 다음과 같다. 아래 코드 중 request.getContextPath()는 현재 실행되고 있는 어플리케이션의 루트 URL 주소를 반환한다. 예를 들어 실행되고 있는 주소가 http://localhost:8080/Lab10-1/index.jsp라면 "http://localhost:8080/Lab10-1"의 값을 반환한다.

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>사원정보</title>
</head>
<body>
<h1>사원정보</h1>
<form action="<%=request.getContextPath()%>/person/personSuccess.jsp"
 method="post">
     <div>NAME :
     <input name="name"/></div><br>
     <div>COMPANY :
     <input name="company"/></div><br>
     <div>PHONE :
     <input name="phone"/></div><br>
     <div>EMAIL :
     <input name="email"/></div><br>
     <div><input type="submit" value="register"></div>
</form>
</body>
</html>
```

submit 시에 호출 URL

[소스 10-2] 입력 폼 (personForm.jsp)

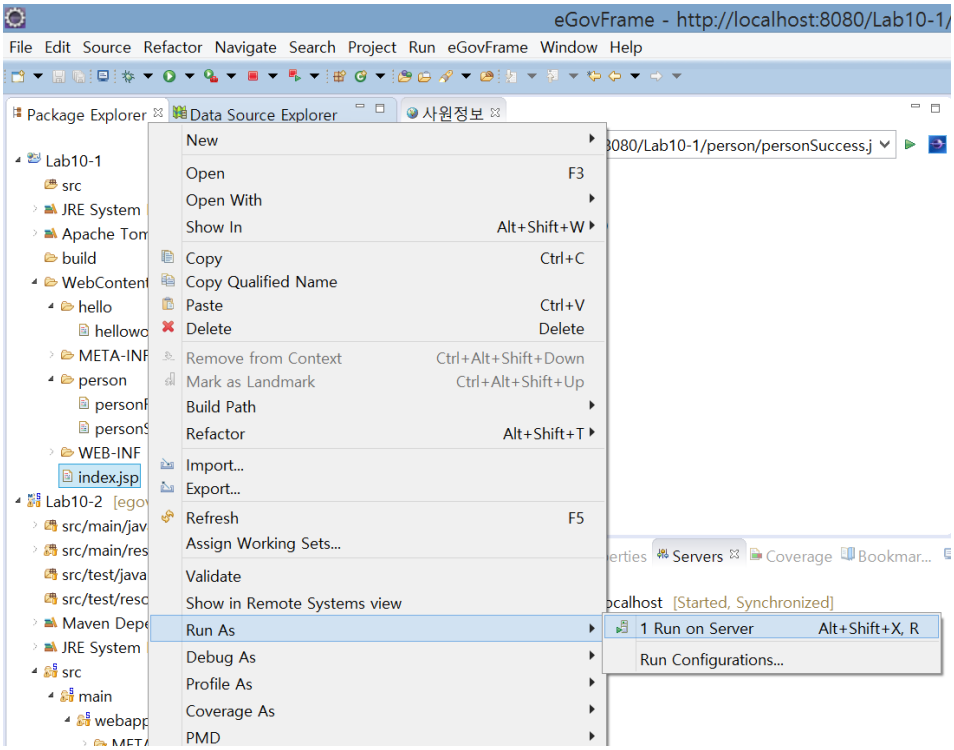
```
<%@ page contentType="text/html; charset=UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>사원정보</title>
</head>
<body>

<%
```

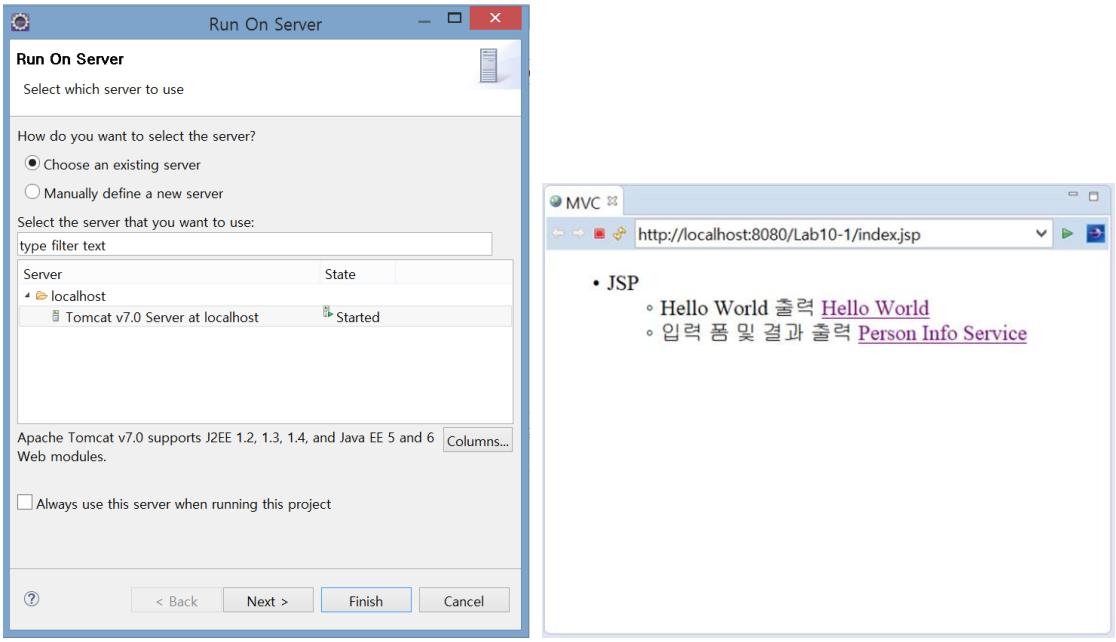
```
    request.setCharacterEncoding("UTF-8");
    String name = request.getParameter("name");
    String company = request.getParameter("company");
    String phone = request.getParameter("phone");
    String email = request.getParameter("email");
%>
<div>NAME: <%=name%></div>
<div>COMPANY: <%=company%></div>
<div>PHONE: <%=phone%></div>
<div>EMAIL: <%=email%></div>
</body>
</html>
```

전달받은 파라미터 값을 변수로 치환

[소스 10-3] 정보 출력 (personSuccess.jsp)

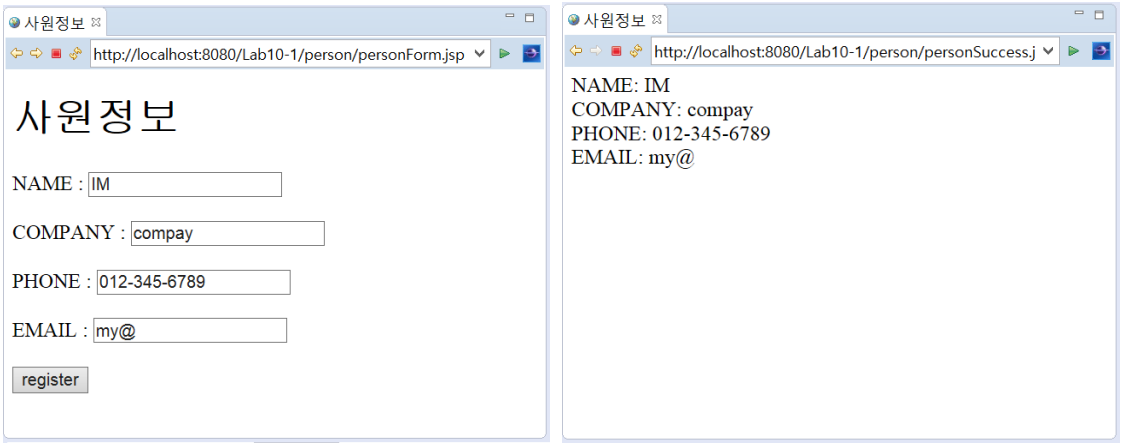
실행을 시키기 위해서는 index.jsp를 선택하고, 마우스 오른쪽 버튼을 누르고 Run As> Run on Server를 선택한다. 실행할 WAS를 선택하는 화면이 나오게 되고 선택 후에 Finish를 누르면 JSP 실행 링크를 포함하는 index 페이지가 출력된다.





[그림 10-3] 웹 프로젝트 실행

입력 폼 및 결과출력 링크를 눌러서 실행하면 다음과 같다.

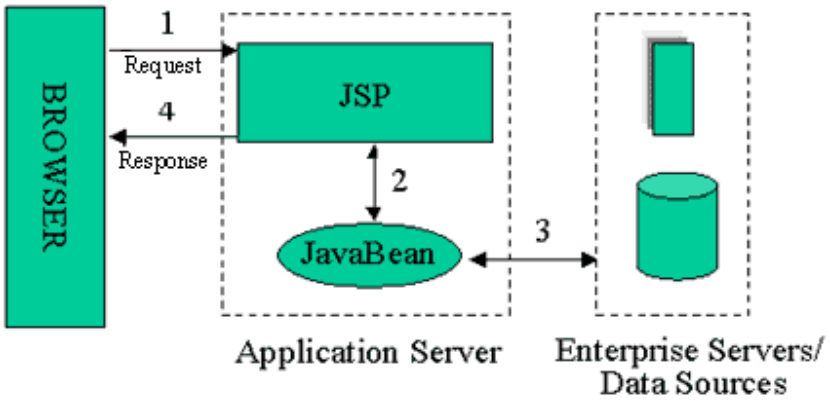


[그림 10-4] 실행 결과

10.2 MVC 개요

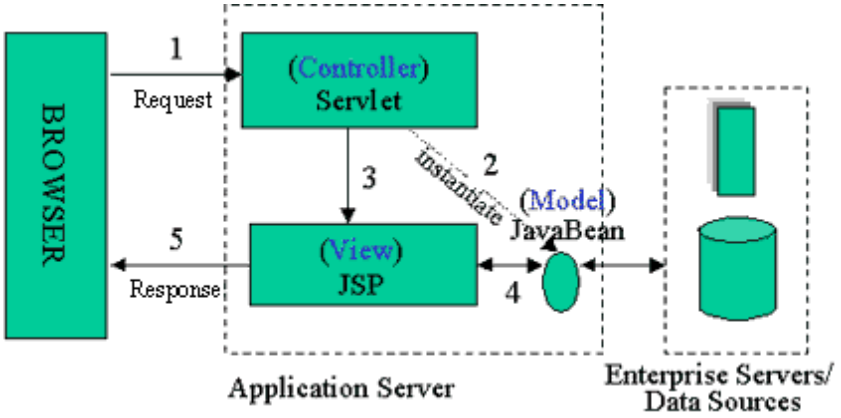
JSP는 간단하고 쉽게 웹 프로그래밍을 할 수 있지만, 로직이 복잡한 프로그램을 개발하기에는 불편한 점들이 많다. 소스코드 안에 웹 브라우저에서 전달 되는 인수 처리, 비즈니스 로직, DB연계 및 HTML 페이지 출력 등을 하나의 모듈에서 처리 하기 때문에 소스코드가 길어지고, 중복되는 코드 구현이 많아질 수 있다. 비즈니스 로직을 별도 Java class인 Java Bean에서 처리하고 결과를 전달 받아 처리하는 형태로 구현도 가능하지만 소스코드의 변경 유연성이나 재사용성을 위한 설계 및 패턴 적용은 개발자가 스스로 처리해야 하기 때문에

적용이 쉽지 않다. 대부분 프로젝트가 급하게 진행 되다 보니 반복되는 코드 등을 신경 쓰지 않고 Copy&Paste 방식으로 하나의 모듈로 구현하는 경우가 많다. 이러한 형태의 소스 코드를 모델1이라고 부른다. 기능별로 해당 JSP를 직접 호출하고, JSP 형태로 결과가 출력된다. 디렉터리가 변경되게 되면 연관된 모든 JSP에서 URL이 변경되기 때문에 URL이나 JSP 디렉터리 및 파일명 등 변경이 어려운 단점이 있으며, 개별 JSP로 직접 호출이 되기 때문에 정보의 흐름을 중앙에서 통제하기가 어렵다.



[그림 10-5] 모델1 구조

모델1은 화면과 비즈니스 로직이 분리되지 않을 경우 매우 복잡하고, 중복된 코드가 많이 발생되며, 화면 수정 등의 디자인 변경 작업에도 소스코드 수정이 필요하여 변경이 매우 어렵다. 이러한 문제점을 해결하기 위해서 모델2(MVC)가 제안되었다. Model2는 브라우저에서 처리 요청이 개별 JSP로 전달되고 결과가 JSP에서 출력되는 것이 아니라, 요청은 하나의 Servlet(프레임워크에서 주로 Dispatcher Servlet)에서 전달 받아 Controller로 처리를 요청하고, Controller는 비즈니스 로직을 처리하기 위하여 Model에 데이터를 전달하고 비즈니스 로직을 호출한다. 처리 결과는 Model에서 화면 출력을 담당하는 JSP에 전달되어 결과가 보여지게 된다. 여기서 JSP는 결과만 보여주는 View의 역할을 하게 된다. Model, View, Controller를 활용하여 화면과 비즈니스 로직을 분리하고, 요청을 단일한 접점에서 처리하는 아키텍처를 모델2(MVC)라고 한다.



[그림 10-6] 모델2(MVC) 구조

MVC는 화면 출력을 위한 UI코드와 비즈니스 로직을 분리함으로써 종속성을 줄이고, 재사용성을 높이고, 소스코드를 쉽게 변경할 수 있도록 한다. 또한 URL을 Controller에 매핑하는 방식이기 때문에 URL을 쉽게 변경할 수 있다. 모든 요청은 단일 접점인 Dispatcher Servlet을 통해 처리가 되기 때문에 보안정책 등의 중앙에서의 통제가 정책적용이 필요한 경우 유연하게 대응이 가능하다. MVC는 코드를 기능에 따라 Model, View, Controller로 분리한다.

[표 10-2] MVC 구성요소

구성요소	설명
Model	어플리케이션의 데이터와 비즈니스 로직을 담는 객체이다.
View	Model의 정보를 사용자에게 표시한다. 하나의 Model을 다양한 View에서 사용 할 수 있다.
Controller	사용자의 요청을 받아 처리를 수행할 Model에 데이터를 전달하고 로직을 실행시키고 결과 출력을 위한 View를 선택한다.

10.3 Spring MVC 구성 및 동작

MVC를 지원하는 프레임워크로는 Spring MVC, Struts, Webwork 등이 있으며, 전자정부 표준프레임워크에서는 Spring MVC를 활용한다. Spring MVC는 MVC 처리를 위한 View, Controller, Dispatcher Servlet, HandlerMapping 등 다양한 인터페이스와 구현 클래스를 제공한다. 웹 요청 파라미터와 커맨드 클래스간의 매핑 기능, 데이터 검증을 할 수 있는 validation 기능과 JSP를 쉽게 구성하도록 전용 Tag를 제공한다. Spring MVC는 다음과 같은 구성요소로 이루어져 있다.

[표 10-3] Spring MVC 구성요소

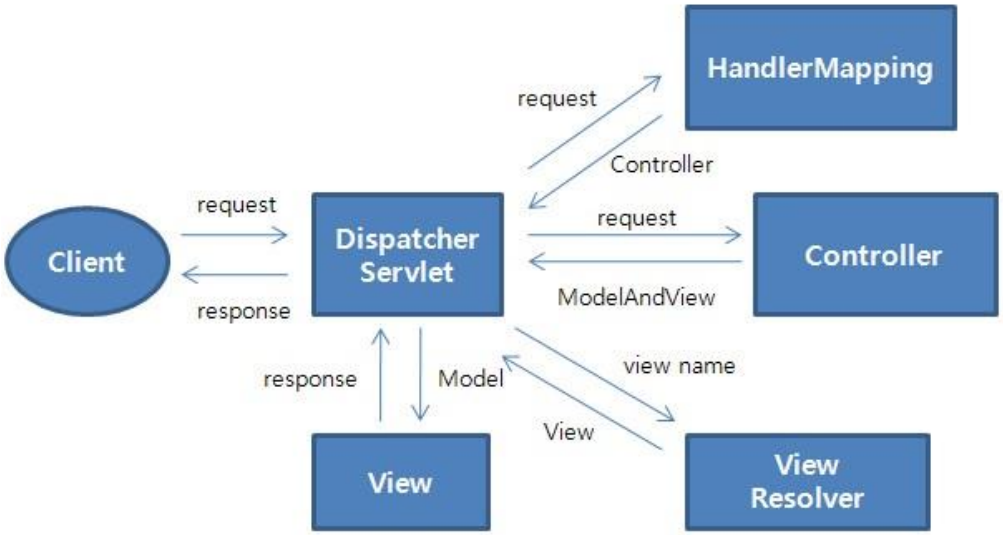
구성요소	설명
Dispatcher Servlet	웹 요청을 전달 받는 단일한 청구로 Front Controller 역할
Handler Mapping	URL로 전달된 웹 요청을 어떤 Controller가 처리할 지 결정
Controller	비즈니스 로직 수행 결과데이터를 ModelAndView에 반영
ModelAndView	Controller가 요청한 비즈니스 로직 수행결과를 반영하는 Model 객체와 이동할 View로 구성
ViewResolver	어떤 View를 선택할지 결정
View	Model의 결과 데이터를 View Resolver에서 선택된 형태로 출력

Spring MVC의 구성 흐름은 다음과 같다.

1. Client 요청이 들어오면 Dispatcher Servlet이 가장 먼저 요청을 받는다. 주로 *.do의 형태로 오는 요청을 처리한다.
2. HandlerMapping이 요청에 해당하는 Controller를 리턴한다. @RequestMapping의 형태로 요청되는 URL을 처리할 Controller 메소드가 Mapping된다.
3. Controller는 비즈니스 로직을 호출하고 전달 받은 결과를 ModelAndView에 반영하여 리턴한다.
4. ViewResolver는 어떤 형태로 출력할 것인지 형태를 결정하고, view name을 받아

해당되는 view객체를 리턴한다. 주로 JSP가 활용되며, XML, JSON등 활용가능하다.

5. View는 Model 객체를 받아 결과를 출력한다.



[그림 10-7] Spring MVC 동작

10.4 Spring MVC 설정 및 활용

Java 어플리케이션 환경에서 Spring 프레임워크 활용을 위하여 ApplicationContext를 활용하여 Spring Container를 생성하였다.

```
String configLocation = "classpath*:META-INF/spring/context-*.xml";
ApplicationContext context = new ClassPathXmlApplicationContext(configLocation);
```

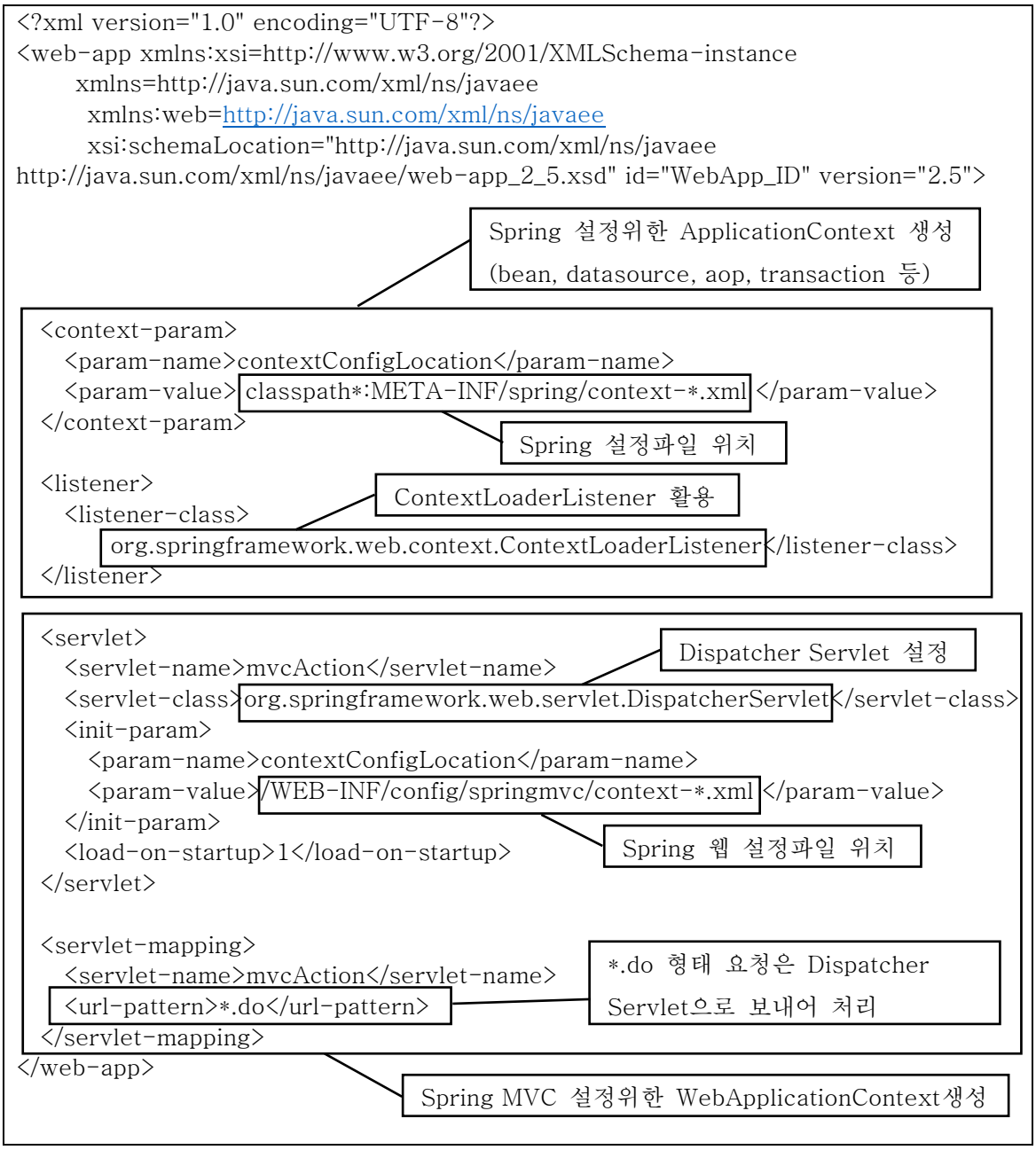
[소스 10-4] Java 어플리케이션에서 Spring Container 설정

JUnit 테스트 코드에서는 @ContextConfiguration을 활용하여 생성하였다.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath*:META-INF/spring/context-*.xml"})
public class CustomerServiceTest { ... }
```

[소스 10-5] JUnit에서 Spring Container 설정

마찬가지로, 웹 환경에서도 어플리케이션이 동작할 때 Spring 프레임워크 활용을 위한 설정이 필요하다. 이를 위하여 웹 어플리케이션 설정 파일인 web.xml을 활용한다. web.xml에 비즈니스 로직을 위한 서비스, DAO 등에서 Spring 활용을 위해서 ContextLoaderListener를 통해 ApplicationContext를 생성한다. 또한, 웹 환경에서 필요한 Spring MVC설정, 메시지 설정, 다국어, validator 등의 설정은 Dispatcher Servlet에서 WebApplicationContext를 생성한다. web.xml은 eGovFrame Web Project에서 src>main>webapp>WEB-INF 하위에 위치하고 있다.



[소스 10-6] 웹 어플리케이션에서 Spring 활용을 위한 web.xml 설정

Dispatcher Servlet 설정에서 URL 패턴이 *.do로 명시되어 있기 때문에, index.do 등 do로 끝나는 요청이 들어오게 되면 Dispatcher Servlet으로 전달 되어 처리가 되며, 다른 확장자를 활용할 수도 있다. *.egov라고 명시한다면 index.egov 등의 요청은 Dispatcher Servlet에서 처리하게 된다. WebApplicationContext 설정에 필요한 Spring MVC 설정파일은 주로 spring-servlet.xml로 지정하여 활용되며, 주요 내용은 아래와 같다. 설정파일에는 Spring MVC에서 활용되는 annotation(@controller, @RequestMapping, @ModelAttribute

등)을 component scan하기 위한 설정과 View를 지정하기 위한 View Resolver가 위치 한다. HandlerMapping의 경우 아무 설정이 되어 있지 않은 경우 defulat handler가 설정된다. (RequestMappingHandlerMapping)

```
<?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:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:component-scan base-package="lab" />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
        p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />

</beans>
```

@Controller 등
Spring MVC Annotation 활용

View를 설정 customerList.do의 View를 매핑한다면
/WEB-INF/jsp/customerList.jsp로 결정됨

[소스 10-7] Spring MVC 설정 (context-servlet.xml)

10.4 Spring MVC 활용

web.xml과 Spring MVC 설정이 되었다면, 이제 화면처리 기능 구현을 위한 Controller 작성이 필요하다. Controller 작성을 편리하게 할 수 있도록 annotation을 제공하고 있으며, 주요 내용은 다음과 같다.

[표 10-4] Controller 주요 annotation

Annotation	설명
@Controller	해당 클래스가 Controller 임을 선언
@RequestMapping	요청에 대해 어떤 Controller 및 메소드가 처리할지 맵핑
@RequestParam	Controller 메소드 파라미터와 웹 요청 파라미터와 맵핑
@ModelAttribute	Controller 메소드의 파라미터와 리턴값을 Model 객체와 바인딩
@SessionAttributes	Model 객체를 세션에 저장하고 사용

@RequestMapping은 GET, POST 방식의 웹 요청에 대해서 URL과 Controller 및 메소드를 맵핑하여 해당 요청이 들어오면 맵핑된 메소드가 실행 되도록 한다.

[표 10-5] @RequestMapping annotation

이름	타입	맵핑 조건	설명
value	String[]	URL값	@RequestMapping(value="/hello.do") @RequestMapping(value={"/hello.do", "/world.do" })

			@RequestMapping(”/hello.do”)
method	Request Method[]	HTTP Request Method	@RequestMapping(method = RequestMethod.POST) 사용 가능 메소드 : GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE

다음 Controller 소스코드에서 GET방식 /hello.do는 helloGet, POST방식이면 helloPost가 실행된다. @Controller가 활용되어 Controller임을 나타내고 있다.

```
@Controller
public class HelloController {

        @RequestMapping(value="/hello.do", method = RequestMethod.GET)
        public String helloGet(){
          ...
        }

        @RequestMapping(value="/hello.do", method = RequestMethod.POST)
        public String helloPost(){
          ...
        }
}
```

[소스 10-8] Controller에서 @RequestMapping 활용

@RequestParam은 웹 요청 파라미터를 변수로 맵핑 하기 위한 것으로 Java에서 제공하는 request. getParameter()와 역할이 같다. 해당 파라미터가 필수가 아닌 경우에는 required=false로 표시를 해야 한다. required=true가 default이기 때문에 유의해야 한다. 아래의 경우 name으로 들어오는 파라미터의 값을 name 변수에 저장한다. pageNo는 필수값이 아니기 때문에 생략이 가능하며, pageNo 변수에 저장된다.

```
@Controller
public class HelloController {

     @RequestMapping("/hello.do")
     public String hello(@RequestParam("name") String name,
                         @RequestParam(value="pageNo", required=false) String pageNo){
        ...
     }
}
```

[소스 10-9] Controller에서 @RequestPraram 활용

@RequestParam은 파라미터 개수만큼 정의가 되기 때문에, 어떤 입력 폼에서 40개의 항목이 입력된다면 40개의 @RequestParam을 정의해야 된다. 이러한 불편함을 없애기 위해서 View를 제공하는 JSP input 항목들과 model 클래스의 변수들이 매핑할 수 있도록 @ModelAttribute를 지원한다. personForm에서 submit이 호출되면 input 태그의 값들은 model에 담기어 controller로 전달된다. 코드에서 “personinfo”라고 지정된 값은 View의

commandName과 동일해야 맵핑이 된다. Controller로 전달 된 값은 @ModelAttribute에 정의 된 대로 person 클래스를 타입으로 하는 command 클래스로 생성된다.

[Controller] PersonServiceController.java

```
@RequestMapping(value = "/person.do", method=RequestMethod.POST)
public String regist(@ModelAttribute("personinfo") person command,
        BindingResult errors, ModelMap model) throws Exception {
    ...

    return successView;
}
```

View commnadName과 같아야 함
View에서의 input 값이 model에 맵핑되어 person type의 Command 클래스 생성

[View] personForm.jsp

```
<body>
<form:form commandName="personinfo">
 <div>NAME :
 <form:input path="name"/></div><br>
 <div>COMPANY :
 <form:input path="company"/></div><br>
 <div>PHONE :
 <form:input path="phone"/></div><br>
 <div>EMAIL :
 <form:input path="email"/></div><br>
 <div>
 <input type="submit" value="register">
 </div>
</form:form>
```

[Model] person.java

```
public class person {

    private String name;
    private String company;
    private String phone;
    private String email;

}
```

Name, company, phone, email
Model과 View input 항복 맵핑

[소스 10-10] Controller에서 @ModelAttribute 활용

이제 Controller에서 남은 부분은 리턴 메시지를 전달하여 view를 보여주는 것이다. View Resolver에 설정된 데로 JSP로 view가 보여지게 된다. Controller에서 GET일때와 POST일 때 다른 변수를 리턴하고 있다. GET일때는 formView (person/personForm)을 리턴하고, POST일때는 successView (person/personSuccess)를 리턴한다. 처음 person.do가 호출될 때는 GET방식으로 새로운 값을 입력할 수 있는 입력폼이 출력이 되고, 입력폼에서 값을 입력한 다음에 submit을 하면 POST로 전달되어 처리가 되고, 결과적으로 personSuccess가 출력된다. String으로 리턴을 할 때는 ViewResolver에서 정의 된 것처럼 /WEB-INF/jsp/ 디렉터리 밑에서 해당되는 JSP를 찾게 된다. SuccessView는 /WEB-INF/jsp/person/personSuccess.jsp를 찾게 되는 것이다. @ModelAttribute를 써서 바인딩 된 model 클래스 이외의 정보를 view로 전달하기 위해서는 인수 부분에 ModelMap model을 정의해주고, 전달할 값을 클래스에 넣은 다음에 model.addAttribute(“pinfo”, command)의 형태로 추가하여 전달하고, view에서는 ${pinfo.name}의 형태로 출력한다. 앞에서 살펴보았던 예제에서 리턴 타입과 화면출력에 관련하여 살펴보면 다음과 같다.

[Controller] PersonServiceController.java

```
private String formView = "person/personForm";
private String successView = "person/personSuccess";

@RequestMapping(value = "/person.do", method=RequestMethod.GET)
protected String idInput(ModelMap model) throws Exception{
        model.addAttribute("personinfo",new person());
        return formView;
}
@RequestMapping(value = "/person.do", method=RequestMethod.POST)
protected String regist(@ModelAttribute("personinfo") person command,
BindingResult errors, ModelMap model) throws Exception {

        model.addAttribute("pinfo",command);
        return successView;
}
```

입력 Form 초기화 위해 필요

person/personForm.jsp 출력

객체를 view에 전달

person/personSuccess.jsp 출력

[View] personSuccess.jsp

```
<body>
    <div>NAME: ${pinfo.name}</div>
    <div>COMPANY: ${pinfo.company}</div>
    <div>PHONE: ${pinfo.phone}</div>
    <div>EMAIL: ${pinfo.email}</div>
</body>
```

addAttribute에 의해
전달된 클래스를 출력

[소스 10-11] Controller에서 리턴 값에 의한 view 출력

수정폼을 구현할 경우 addAttribute에 view의 form commandName을 그대로 적어주고, 객체를 맵핑 해주면 input tag에 값이 초기화 되어 들어가게 된다.

```
@RequestMapping(value = "/person3.do", method=RequestMethod.GET)
protected String personInput3(ModelMap model) throws Exception{
        person pobject = new person();
        pobject.setName("KIM");
        pobject.setCompany("company");
        pobject.setPhone("02-1234-5948");
        pobject.setEmail("abc@email.com");

        model.addAttribute("personinfo",pobject);
        return formView;
}
```

pojbect의 값이 초기값으로
폼 화면에서 출력

[소스 10-12] Controller에서 초기값을 가진 view 출력

10.4 Spring MVC 예제

Spring MVC 에제 소스코드는 다음과 같이 구성되어 있다. 소스코드 내용은 대부분 앞의 설정에서 설명이 되었다. index.jsp와 Controller 전체 소스코드 내용을 설명하도록 하겠다.

[표 10-6] 소스코드 구성

구분	코드명	설명
[src/main/java]		
Controller	lab.web.HelloWorldController.java	HelloWorld 출력
	lab.web.PersonServiceController.java	@ModelAttribute를 활용한 model 바인딩과 폼입력과 출력을 설명
Model	lab.web.model.person.java	이름, 회사, 전화번호, email 관리
[src/main/resources]		
Spring	META-INF/spring/context-common.xml	Annotation처리를 위한 component scan (Service, DAO 용)
Log4j	log4j2.xml	로그출력
[src/main/webapp]		
Spring MVC	WEB-INF/config/springmvc/context-srvlet.xml	Spring MVC 설정
View (JSP)	WEB-INF/jsp/hello/helloworld.jsp	HelloWorldController에서 출력
	WEB-INF/jsp/person/personForm.jsp	psersonServiceController 폼 입력
	WEB-INF/jsp/person /personSuccess.jsp	폼 입력 결과를 출력
Web 설정	WEB-INF/web.xml	Spring ApplicationContext 및 Dispatcher Servlet 설정
Index	index.jsp	실행을 위한 링크

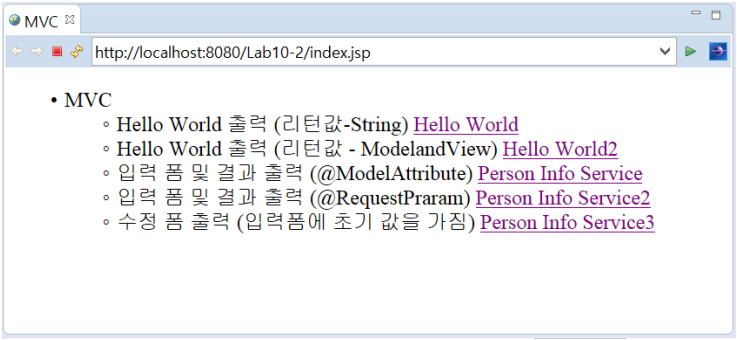
실행을 위한 index.jsp는 다음과 같다. HelloWorldController에 hello.do, hello2.do 두 개의 요청을 하게 되며, 리턴 값을 전달하는 view 호출 방식만 다르고 실행 결과는 같다. 마찬가지로 personSrviceController도 세 가지의 요청을 가지는데, person.do의 경우는 spring의 input 태그를 활용하여 model과 바인딩이 되어 있는 형태이고, person2.do의 경우는 @RequestParam을 이용 파라미터를 전달받는다. person3.do의 경우는 @ModelAttribute로 바인딩이 되어 있는 페이지에 초기 값을 출력시키게 되어 있어, 수정 폼 등으로 활용 할 수 있는 방식이다.

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>MVC</title>
</head>
<body>
<ul id="menu">
    <li> MVC
      <ul>
        <li>Hello World 출력 (리턴값-String) <a href="hello.do">Hello World</a></li>
         <li>Hello World 출력 (리턴값 - ModelandView)
                  <a href="hello2.do">Hello World2</a></li>
         <li>입력 폼 및 결과 출력 (@ModelAttribute)
                   <a href="person.do">Person Info Service</a></li>
```

```
            <li>입력 폼 및 결과 출력 (@RequestPraram)
                    <a href="person2.do">Person Info Service2</a></li>
            <li>수정 폼 출력 (입력폼에 초기 값을 가짐)
                    <a href="person3.do">Person Info Service3</a></li>
        </ul>
    </li>
</ul>
</body>
</html>
```

[소스 10-13] index.jsp 코드

index.jsp를 실행한 결과는 다음과 같다. 각 링크를 눌러서 결과를 확인할 수 있다. 입력 폼을 출력하고 입력된 값을 결과로 출력하는 형태로 구성되어 있고, 마지막 링크의 경우 임의의 값을 초기값으로 출력할 수 있도록 기능을 제공한다.



[그림 10-8] index.jsp 실행결과

personServiceController의 내용은 다음과 같다.

```
@Controller
public class PersonServiceController {
        private String formView = "person/personForm";
        private String successView = "person/personSuccess";

        @RequestMapping(value = "/person.do", method=RequestMethod.GET)
        protected String personInput(ModelMap model) throws Exception{
                model.addAttribute("personinfo",new person());
                return formView;
        }

        @RequestMapping(value = "/person.do", method=RequestMethod.POST)
        protected String regist(@ModelAttribute("personinfo") person command,
                        BindingResult errors, ModelMap model) throws Exception {
                if (errors.hasErrors()) {
                        return formView;
                }
                model.addAttribute("pinfo",command);
                return successView;
        }
```

@ModelAttribute 바인딩

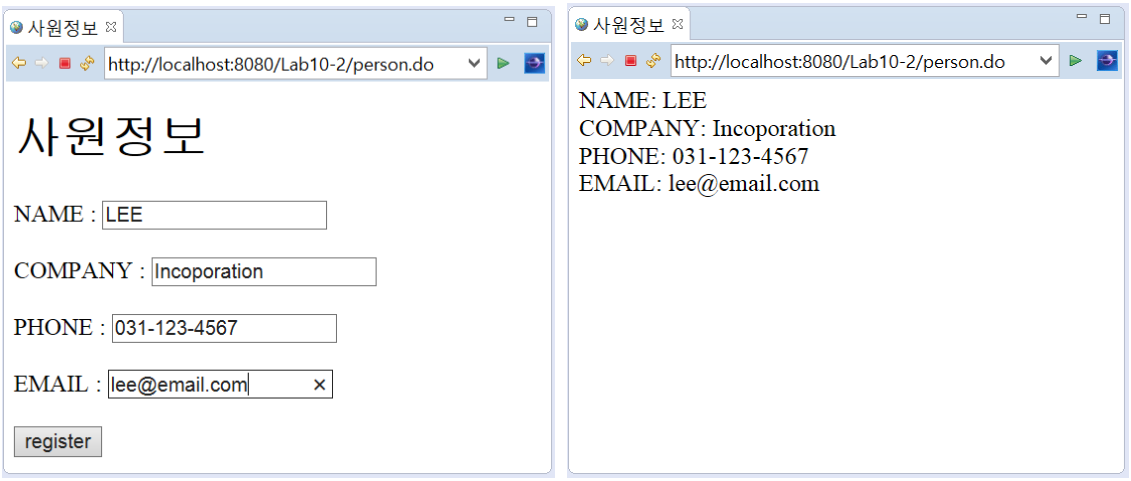
```
	@RequestMapping(value = "/person2.do", method=RequestMethod.GET)
	protected String personInput2(ModelMap model) throws Exception{
		model.addAttribute("personinfo",new person());
		return formView;
	}

	@RequestMapping(value = "/person2.do", method=RequestMethod.POST)
	protected String regist2(@RequestParam("name") String name,
		@RequestParam(value="company", required=false) String company,
		@RequestParam(value="phone", required=false) String phone,
		@RequestParam(value="email", required=false) String email,
		ModelMap model) throws Exception {
		person personinfo = new person();
		personinfo.setName(name);
		personinfo.setCompany(company);
		personinfo.setPhone(phone);
		personinfo.setEmail(email);

		model.addAttribute("pinfo",personinfo);
		return successView;
	}

	@RequestMapping(value = "/person3.do", method=RequestMethod.GET)
	protected String personInput3(ModelMap model) throws Exception{
		person pobject = new person();
		pobject.setName("KIM");
		pobject.setCompany("company");
		pobject.setPhone("02-1234-5948");
		pobject.setEmail("abc@email.com");
		model.addAttribute("personinfo",pobject);
		return formView;
	}

	@RequestMapping(value = "/person3.do", method=RequestMethod.POST)
	protected String regist3(@ModelAttribute("personinfo") person command,
			BindingResult errors, ModelMap model) throws Exception {
		if (errors.hasErrors()) {
			return formView;
		}
		model.addAttribute("pinfo",command);
		return successView;
	}
}
```

@RequestParam 활용

초기값 설정 및 출력

[소스 10-14] PersonServiceController 코드

실행결과는 다음과 같다. person3.do로 실행하는 경우에는 처음에 초기 값이 입력되어 출력되는 점이 다르다.



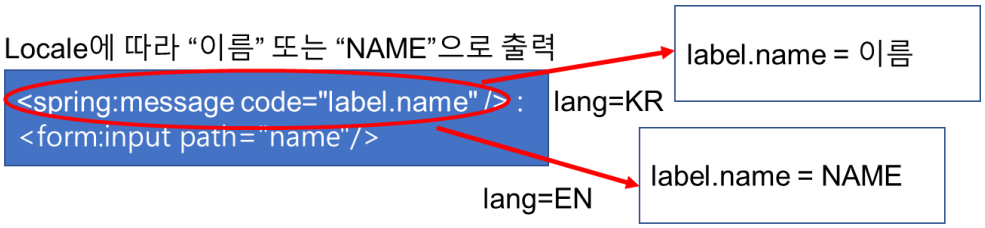
[그림 10-9] personServiceController 실행결과

11. 국제화(Internalization)

국제화는 여러 언어로 웹 서비스가 필요할 때 중복으로 페이지를 만들지 않고, 단일한 페이지에 언어 설정에 따라 다른 언어를 보여줄 수 있도록 한다.

11.1 국제화 개요

여러 언어로 웹 서비스를 제공해야 될 경우에 언어별로 페이지를 별도로 만들어 제공하는 방법이 있지만, 이경우 언어별로 페이지를 관리해야 되기 때문에 불편할 수 있다. Spring MVC 국제화 기능은 동일한 페이지를 사용자 설정에 따라 해당 언어로 보여줄 수 있도록 제공한다. 쿠키나 브라우저 locale 정보를 활용할 수 도 있는데, 일반적으로 세션에 저장된 언어 설정에 따라 해당 언어를 보여주는 방법이 가장 많이 활용 된다. 화면을 출력하는 JPS 코드에 직접 해당 언어를 활용하여 구현을 하는 것이 아니라, Spring message tab를 활용하여 나타내고 실제로 페이지가 보여질 때 해당 언어 설정에 따라 파일에서 값을 읽어와서 대체하여 출력하게 된다.



[그림 11-1] 국제화 방식

11.2 국제화 설정

다국어가 활용 될 수 있도록 캐릭터셋 인코딩 설정이 필요하다. 시스템에서 다국어 처리가 용이하도록 캐릭터셋을 UTF-8로 활용하는 것이 일반적이다. 이를 위해 web.xml에 다음과 같이 캐릭터셋 인코딩을 위한 필터 설정을 추가한다.

```
<filter>
        <filter-name>encoding-filter</filter-name>
        <filter-class>
            org.springframework.web.filter.CharacterEncodingFilter
        </filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
  <filter-mapping>
       <filter-name>encoding-filter</filter-name>
       <url-pattern>/*</url-pattern>
  </filter-mapping>
```

[소스 11-1] UTF-8 인코딩 설정

10장에서 설명하였듯이 웹과 관련된 설정은 Spring MVC Dispatcher Servlet에 설정을 해야 한다. 웹 페이지에서 어떤 언어로 출력을 해야 하는지 설정정보를 주로 세션을 통하여 관리하는데, 세션 기반 locale을 활용하기 위해서는 SessionLocaleResolver 설정이 필요하다. 또한, View를 출력하기 전에 어떤 언어로 출력하도록 세션에 설정 되었는지 확인이 필요한데, 이러한 처리를 위해 interceptor 설정이 필요하다. Interceptor는 Controller가 실행되기 전에 자동으로 동작하는데, 사용하기 위해서는 handlerMapping의 설정이 필요하다. 10장에서는 default로 활용을 하였기 때문에 별도의 handler mapping을 설정하지 않아도 기본 handler mapping인 RequestMappingHandlerMapping이 자동으로 설정되어 활용 할 수 있었지만, 국제화 기능을 활용하기 위해서는 RequestMappingHandlerMapping과 언어설정을 확인하기 위해 localeChangeInterceptor 설정이 필요하다. localeChageInterceptor는 Controller가 실행되기 전에 세션에 저장된 파라미터 값을 확인한다. 아래 예제에서 파라미터 값은 lang으로 설정되어 있어, 페이지 요청시에 “?lang=kr” 또는 “?lang=en”과 같은 형태로 활용한다. 파라미터 값 lang은 설정에서 변경 가능하다.

```
  <context:component-scan base-package="lab" />

  <bean   class="org.springframework.web.servlet.view.InternalResourceViewResolver"
        p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />

<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />

<bean id="localeChangeInterceptor"
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
       <property name="paramName" value="lang" />
</bean>

<bean id="annotationMapper"
class="org.springframework.web.servlet.mvc.method.annotation.
         RequestMappingHandlerMapping">
       <property name="interceptors">
            <list>
                 <ref bean="localeChangeInterceptor" />
            </list>
       </property>
</bean>
<bean class="org.springframework.web.servlet.mvc.method.annotation.
                RequestMappingHandlerAdapter" />
```

세션 기반 Locale관리

Locale확인 위한 Interceptor 설정

Locale확인 위한 Interceptor 설정

[소스 11-2] Dispatcher Servlet 설정

11.2 메시지 Source 설정

웹 페이지에 직접 내용이 표시되는 것이 아니라, locale 설정에 맞는 메시지 값을 읽어서 보여주는 형태이기 때문에 언어별로 메시지를 설정할 수 있도록 설정이 필요하다. 메시지 Soruce 설정에는 언어별 메시지(message properties) 파일의 경로와 파일명 정보를 가진다.

Message properties 파일은 locale정보에 따라 파일 이름이 결정된다. 메시지 파일명이 message-common이고, locale이 ko인 경우에는 message-common_ko.properties, en인 경우에는 message-common_en.properties으로 설정한다. Default 메시지의 경우 locale정보 없이 message-common.properties와 같은 형태로 설정하면 된다.

```
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>META-INF/messages.message-common</value>
        </list>
    </property>
</bean>
```

메시지 파일경로 및 파일명

[소스 11-3] message source 설정

다음과 같이 message properties파일을 설정하였다.

[표 11-1] Message properties 파일 설정

message-common.properties message-common_kr.properties	message-common_en.properties
label.name=이름 label.company=회사 label.phone=전화 label.email=전자메일	label.name=NAME label.company=COMPANY label.phone=PHONE label.email=EMAIL

11.3 View 설정 (JSP)

국제화를 적용하여 디스플레이 할 항목에 <spring:message code="label.name"/> 형태로 JSP에 Spring message 태그를 이용하여 코드를 생성하면 페이지가 로드 될 때 설정된 locale에 해당되는 message properties 값을 읽어서 HTML 출력 시에 값을 바꾸어 보여지도록 한다.

```
<form:form commandName="personinfo">
    <div><spring:message code="label.name" /> :
    <form:input path="name"/></div><br>
    <div><spring:message code="label.company" /> :
    <form:input path="company"/></div><br>
    <div><spring:message code="label.phone" /> :
    <form:input path="phone"/></div><br>
    <div><spring:message code="label.email" /> :
    <form:input path="email"/></div><br>
    <div><input type="submit" value="register"></div>
</form:form>
```

Message 태그를 이용 파일에서 label.name을 읽어와서 출력

[소스 11-4] JSP 페이지

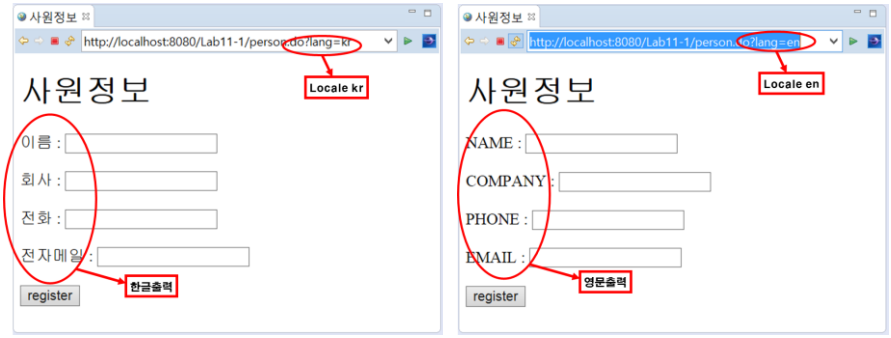
11.4 국제화 예제 코드

국제화 예제는 전체적으로 예제10-2와 동일하나, 국제화 적용을 위한 설정과 view코드가 변경되었다. Locale 별로 출력되는 메시지를 저장하는 message properties파일과 설정이 새로 추가 되었다.

[표 11-2] 소스코드 구성

구분	코드명	설명
[src/main/java]		
Controller	lab.web.HelloWorldController.java	HelloWorld 출력
	lab.web.PersonServiceController.java	@ModelAttribute를 활용한 model 바인딩과 폼입력과 출력을 설명
Model	lab.web.model.person.java	이름, 회사, 전화번호, email 관리
[src/main/resources]		
Spring	META-INF/spring/context-common.xml	Annotation처리를 위한 component scan (Service, DAO 용)
Log4j	log4j2.xml	로그출력
Message Properties	META-INF/messages/ message-common.properties	Default locale message 설정
	META-INF/messages/ message-common_kr.properties	한글 message 설정
	META-INF/messages/ message-common_en.properties	영문 message 설정
[src/main/webapp]		
Spring MVC	WEB-INF/config/springmvc/context-srvlet.xml	Spring MVC 설정, Message Source 설정
View (JSP)	WEB-INF/jsp/hello/helloworld.jsp	HelloWorldController에서 출력
	WEB-INF/jsp/person/personForm.jsp	psersonServiceController 폼 입력
	WEB-INF/jsp/person /personSuccess.jsp	폼 입력 결과를 출력
Web 설정	WEB-INF/web.xml	Spring ApplicationContext 및 Dispatcher Servlet 설정
Index	index.jsp	실행을 위한 링크

예제를 실행한 결과는 다음과 같다. lang=kr인 경우 한글로 출력이 되었고, lang=en인 경우에는 영어로 출력이 되었다. 헤더의 "사원정보"는 국제화 처리가 되어 있지 않아 그대로 한글로 출력이 되었다.



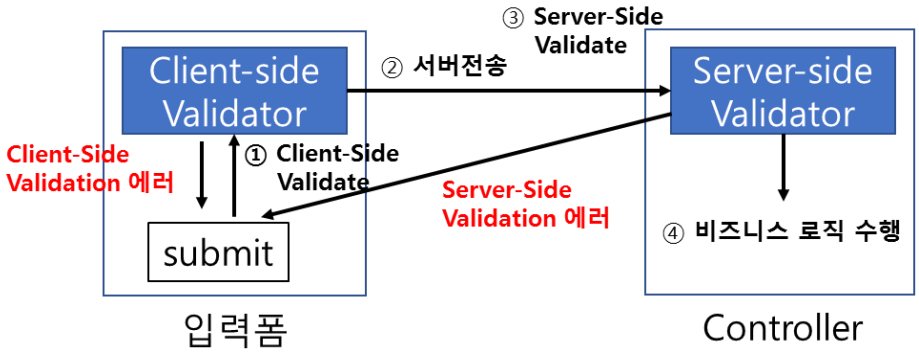
[그림 11-2] locale변경에 따른 결과 message 출력 확인

12. Validation

Validation은 사용자의 입력 값이 적정한 지를 검사하여 어플리케이션이 잘 동작되도록 한다. 개발자가 체크 로직을 일일이 구현하지 않고 규칙만 정의하면 필요한 처리를 미리 구현된 모듈을 호출하여 자동으로 수행할 수 있다.

12.1 Validation 개요

웹 서비스에서 사용자의 입력을 받아 DB에 저장하거나 계산 등의 처리를 수행하게 되는데, 정보가 잘못 입력되게 되면 시스템에 문제가 발생되고 정상적인 처리가 어렵게 된다. 어플리케이션에서 사용자가 입력한 값의 유효성을 검증(Validation)하는 부분은 중요한 부분이다. 하지만, 입력 필드가 많고 검증하는 내용이 많아지면 그만큼 개발자가 처리해야 되는 부분도 많아지게 된다. 숫자가 입력되는 필드에 문자가 입력되거나 반드시 입력되어야 하는 필드가 비어 있거나, 정해진 길이 보다 길게 입력되거나 하면 어플리케이션에 문제가 발생 될 수 있다. 이러한 점을 방지하기 위해 JSP나 JavaScript에서 입력된 값의 타입, 길이, 값 유무 등에 대해서 체크를 하게 되는데 일일이 IF 등을 활용하여 구현을 하게 되면 매우 복잡하고 많은 시간이 소요된다. 이러한 문제점을 해결하기 위해 Apache Commons Validator는 validation을 위한 규칙을 XML로 정의하면 JSP나 JavaScript에서 체크해야 되는 로직들이 자동으로 처리되도록 지원하여 매우 편리하다. Validation은 웹 브라우저에서 입력된 값이 서버로 전송되기 전에 JavaScript에서 체크되는 client-side validation과 서버에서 JSP나 Controller에서 체크 하는 server-side validation으로 구성된다. 입력된 값은 먼저 JavaScript에서 검증이 되고, 실패하면 에러 메시지를 출력하고 다시 입력을 받게 된다. Client-side validation이 통과되면 서버로 전송되고 server-side validation을 수행한다. 정상적으로 검증이 되면 비즈니스 로직이 수행되고, 에러가 발생되면 View(JSP)에 에러메시지를 출력하고 다시 입력폼으로 되돌아가게 된다. Server-side validation은 client-side에서 수행되고 다시 반복 되는 것 같아 불필요한 것 같지만, 웹 브라우저를 통하지 않고 프로그램을 작성해서 POST로 처리를 수행하게 하는 등의 방법들이 있어서 server-side validation도 반드시 필요하다.



[그림 12-1] Validation 동작방법

12.2 Validation 설정

표준프레임워크에서는 Validation처리를 위하여 Apache commons validator를 Spring 에 연계하여 활용한다. Apache commons validator는 필수 값, 변수 타입, 최대최소길이, 이메일 등 입력된 값의 유효성을 검증할 수 있도록 기능이 제공된다. 또한 Java와 JavaScript를 모두 지원하여 client-side, server-side 모두 하나의 설정으로 관리할 수 있어 편리하다. Validator는 화면처리 레이어인 egovframework.rte.ptl.mvc의 pom.xml에 아래와 같이 설정되어 있다. Spring에서 연계할 수 있는 template 모듈인 spring-modules-validator을 활용하며, Apache commons validator 1.4.0을 활용한다. 표준프레임워크 화면처리 레이어 pom.xml에 설정이 되어 있기 때문에 별도로 설정 할 필요는 없다.

```
<dependency>
        <groupId>org.springmodules</groupId>
        <artifactId>spring-modules-validation</artifactId>
        <version>0.9</version>
</dependency>
<dependency>
        <groupId>commons-validator</groupId>
        <artifactId>commons-validator</artifactId>
        <version>1.4.0</version>
        <exclusions>
        <!-- Exclude Commons Logging in favor of SLF4j -->
                <exclusion>
                        <artifactId>commons-logging</artifactId>
                        <groupId>commons-logging</groupId>
                </exclusion>
        </exclusions>
</dependency>
```

[소스 12-1] validator활용을 위한 pom.xml 설정

Spring 설정을 위해서 spring-modules-validation에 포함된 DefaultBeanValidator, DefaultValidatorFacotry을 context-validator.xml에 설정한다. DefaultValidatorFactory는 validation rule이 정의된 설정파일인 validator.xml에 따라 유효성을 검증한다. validator.xml 은 유효성 검증을 위한 규칙이 템플릿으로 정의되어 있는 validator-rules.xml을 기반으로 동작한다. DefaultBeanValidator는 DefaultValidatorFactory에 의해서 생성 되며, Controller에서는 DefaultBeanValidator를 참조하여 활용하면 된다.

```
  <bean id="beanValidator"
  class="org.springmodules.validation.commons.DefaultBeanValidator">
      <property name="validatorFactory" ref="validatorFactory"/>
  </bean>

  <bean id="validatorFactory"
   class="org.springmodules.validation.commons.DefaultValidatorFactory">
      <property name="validationConfigLocations">
         <list>
```

```
            <value>/WEB-INF/config/validator-rules.xml</value>
            <value>/WEB-INF/config/validator.xml</value>
        </list>
    </property>
</bean>
```

Validation 설정

[소스 12-2] Validator bean 설정 (context-validator.xml)

Validation 설정을 위하여 validator.xml은 다음과 같은 형태로 설정한다. Validator-rules.xml은 그대로 활용하면 된다. 필요에 따라 새로운 Rule을 생성할 때는 Validator-rules.xml에 정의하여 활용할 수 있다. form name은 입력 유효성을 검증할 대상과 맵핑하게 된다. “employee”의 경우에 server-side는 @ModelAttribute의 폼 이름이 “employee” 인 경우, client-side는 <validator:javascript formName=”employee” 인 경우 입력 값을 검증하게 된다. field property는 유효성 검증을 할 항목으로 server-side는 class 멤버변수이며, client-side는 JSP의 input path이다. 유효성 검증을 위한 조건으로 필수여부, 한글, 정수 값 등을 명시한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE form-validation PUBLIC
"-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1//EN"
    http://jakarta.apache.org/commons/dtds/validator_1_1.dtd>
<form-validation>
    <formset>
        <form name="person">
            <field property="name" depends="required,korean">
                <arg0 key="label.name" />
            </field>
            <field property="company" depends="required">
                <arg0 key="label.company" />
            </field>
            <field property="phone" depends="required">
                <arg0 key="label.phone" />
            </field>
        </form>
    </formset>
</form-validation>
```

Server-side: form 클래스명

Client-side: JSP validator form name

Message 표시 항목명

유효성 검증 필드명과 조건

(필수항목, 한글)

[소스 12-3] validator.xml 설정

유효성 검증을 위한 조건은 다음 표와 같다. Apache commons에서 제공하는 항목과 표준프레임워크에서 만들어진 항목이 있다.

Spring Modules: org.springmodules.validation.commons.FieldChecks

표준프레임워크: egovframework.rte.ptl.mvc.validation.RteFieldChecks

[표 12-1] 유효성 검증 조건

name(validation rule)	FieldCheck 메소드	기능
required	validateRequired	필수값 체크
minlength	validateMinLength	최소 길이 체크
maxlength	validateMaxLength	최대 길이 체크
mask	validateMask	정규식 체크
byte	validateByte	Byte형 체크
short	validateShort	Short형 체크
integer	validateInteger	Integer형 체크
long	validateLong	Long형 체크
float	validateFloat	Float형 체크
double	validateDouble	Double형 체크
date	validateDate	Date형 체크
range	validateIntRange	범위 체크
intRange	validateIntRange	int형 범위 체크
floatRange	validateFloatRange	Float형 범위체크
creditCard	validateCreditCard	신용카드번호체크
email	validateEmail	이메일체크
ihidnum	validateIhIdNum	주민등록번호체크
korean	validateKorean	한글체크
htmltag	validateHtmlTag	<,> 태그 정보 체크

12.3 Message 설정

유효성 검증 결과 에러가 있는 경우에는 에러메시지를 출력하게 된다. 에러메시지는 11장에서 설정했던 방식을 그대로 활용할 수 있다. 국제화 방식을 그대로 활용하여 locale에 따라 에러메시지 출력이 가능하다. validator.xml에서 <arg0 key="label.name" />의 형태로 설정되어 활용이 되고, 유효성 검증이 실행하여 에러가 발생되었을 때 에러메시지에 대하여 설정이 필요하다. 디폴트 설정인 message-common.xml과 한글 설정은 다음과 같다.

```
label.name=이름
label.company=회사
label.phone=전화
label.email=전자메일

errors.required={0}은 필수 입력값입니다.
errors.minlength={0}은 {1}자 이상 입력해야 합니다.
errors.maxlength={0}은 {1}자 이상 입력할수 없습니다.
errors.invalid={0}은 유효하지 않은 값입니다.
```

```
errors.byte={0}은  byte타입이어야 합니다.
errors.short={0}은  short타입이어야 합니다.
errors.integer={0}은 integer 타입이어야 합니다.
errors.long={0}은 long 타입이어야 합니다.
errors.float={0}은 float 타입이어야 합니다.
errors.double={0}은 double 타입이어야 합니다.
errors.date={0}은 날짜 유형이 아닙니다.
errors.range={0}은 {1}과 {2} 사이의 값이어야 합니다.
errors.creditcard={0}은 유효하지 않은 신용카드 번호입니다.
errors.email={0}은 유효하지 않은 이메일 주소입니다.
errors.korean={0}은 한글을 입력하셔야 합니다.
errors.htmltag=태그는 사용하실수 없습니다.
```

[소스 12-4] message설정 (message-common.properties, message-common_kr.properties)

영문 메시지 설정은 다음과 같다.

```
label.name=NAME
label.company=COMPANY
label.phone=PHONE
label.email=EMAIL

errors.required={0} is necessary.
errors.minlength={0} should be over {1} letters.
errors.maxlength={0} is not allowed over {1} letters.
errors.email={0} is not invalid email address.
errors.korean={0} should be korean.
```

[소스 12-5] message설정 (message-common_en.properties)

12.4 Server-side validation

Server-side validation을 위해서는 Controller에 DefaultBeanValidator를 활용하여 유효성 검증이 필요하다. 폼에서 submit이 되면 POST로 전달이 되게 되며, 유효성 검증을 수행하기 위해 validate를 실행하고, 유효성 검증 조건을 만족하지 않으면 에러를 발생시키게 된다. 에러가 있는 경우에는 입력 폼으로 다시 돌아가서 에러메시지를 출력한다.

```
@Autowired
private DefaultBeanValidator beanValidator;


@RequestMapping(value = "/person.do", method=RequestMethod.GET)
protected String personInput(ModelMap model) throws Exception{
        model.addAttribute("person",new Person());
        return formView;
}

@RequestMapping(value = "/person.do", method=RequestMethod.POST)
protected String regist(@ModelAttribute("person") Person command,
                BindingResult errors, ModelMap model) throws Exception {
```

Validator 의존성 설정

```
    beanValidator.validate(command, errors);
        if (errors.hasErrors()) {
                return formView;
        }
                model.addAttribute("pinfo",command);
        return successView;
}
```

유효성 검증 수행하고, 에러가 있는 경우 입력 폼 출력

[소스 12-6] Server-side validatoin을 위한 Controller 구현

JSP에는 에러가 발생되었을 때 메시지 출력을 위한 구현이 추가 된다. Spring form errors 태그를 활용하며, validator.xml에 정의된 field에 대해서 구현한다. Server-side validation 확인을 위해 JavaScript로 되어 있는 client-side validation을 피하고 바로 서버로 submit 할 수 있도록 버튼을 추가 하였다.

```
<form:form commandName="person">
    <div><spring:message code="label.name" /> :
    <form:input path="name"/><form:errors path="name" /></div><br>
    <div><spring:message code="label.company" /> :
    <form:input path="company"/><form:errors path="company" /></div><br>
    <div><spring:message code="label.phone" /> :
    <form:input path="phone"/><form:errors path="phone" /></div><br>
    <div><spring:message code="label.email" /> :
    <form:input path="email"/></div><br>
    <div><input type="button" value="CLIENT" onclick="save(this.form)"/>
        <input type="submit" value="SERVER"/></div>
</form:form>
```

에러 메시지 출력

Server로 바로 전송

[소스 12-7] Server-side validatoin을 위한 JSP 구현

12.5 Client-side validation

Client-side validation은 서버로 전송이 되지 않고 웹 브라우저 상에서 JavaScript기반으로 동작한다. View가 실행되어 화면에 출력될 때 client-side validation 수행을 위한 JavaScript 로직이 생성되어 출력된다. 생성되는 JavaScript 로직은 validator-rules.xml에 구현되어 있고, 호출되어 활용된다. 아래 코드는 validator-rules.xml에서 required에 해당되는 내용을 보여주고 있다. validator.xml에 유효성 검증 필드가 required로 설정되어 있으면 아래의 validateRequired function을 호출하여 실행되게 된다.

```
<validator name="required"
            classname="org.springmodules.validation.commons.FieldChecks"
                method="validateRequired"
          methodParams="java.lang.Object,
                        org.apache.commons.validator.ValidatorAction,
                        org.apache.commons.validator.Field,
                        org.springframework.validation.Errors"
                   msg="errors.required">
          <javascript><![CDATA[
            function validateRequired(form) {
```

```
                var isValid = true;
                var focusField = null;
                var i = 0;
                var fields = new Array();
                oRequired = new required();
                for (x in oRequired) {
                        var field = form[oRequired[x][0]];
                    if (field.type == 'text' ||
                        field.type == 'textarea' ||
                        field.type == 'file' ||
                        field.type == 'select-one' ||
                        field.type == 'radio' ||
                        field.type == 'password') {

                        var value = '';
                        if (field.type == "select-one") {
                                var si = field.selectedIndex;
                                if (si >= 0) {
                                        value = field.options[si].value;    }
                                        } else {            value = field.value;        }
                        if (trim(value).length == 0) {
                                if (i == 0) {
                                    focusField = field;
                                }
                                fields[i++] = oRequired[x][1];
                                isValid = false;
                        }
                    }
                }
                if (fields.length > 0) {
                    focusField.focus();
                    alert(fields.join('₩n'));
                }
                return isValid;
            }
            function trim(s) {
                return s.replace( /^₩s*/, "" ).replace( /₩s*$/, "" );
            }
            ]]>
        </javascript>
    </validator>
```

[소스 12-8] validator-rules.xml의 required function

Client-side validation은 개별 view에서 validator.do를 호출하여 validation-rules.xml에 정의한 javaScript를 호출하여 활용한다. 따라서, Controller에 보기와 같이 validatr.do가 호출될 수 있도록 requestMapping을 추가하고, validator.jsp를 JSP 메인 디렉터리에 위치하도록 설정 한다.

```
@RequestMapping(value = "/validator.do")
protected String getValidator() throws Exception{
        return "validator";
}
```

[소스 12-9] validator.do를 위한 RequestMapping 설정

```
<%@ page language="java" contentType="javascript/x-javascript" %>
<%@ taglib prefix="validator" uri="http://www.springmodules.org/tags/commons-
validator" %>
<validator:javascript dynamicJavascript="false" staticJavascript="true"/>
```

[소스 12-10] validator.jsp

Request Mapping을 mvc태그 라이브러리를 이용하여 Dispatcher Servlet 설정(context-servlet.xml)에 다음과 같이 설정할 수도 있다.

```
<mvc:view-controller path="/validator.do"/>
```

[소스 12-11] mvc태그를 활용한 설정

다음으로는 개별 JSP에 validator 태그 라이브러리를 설정하고 validator.do가 실행 될 수 있게 자바스크립트 코드를 추가한다. validator javascript에는 validation 설정된 내용을 호출하기 위해 validator.xml의 forname을 설정한다. JavaScript를 추가하여 바로 서버로 submit이 되지 않고, validateVO클래스명()의 형태로 먼저 validation 로직이 호출되도록 한다. WAS에서 실행하여 소스보기를 하여 JSP를 살펴보면 validator-rules.xml의 함수를 호출하고, 에러 메시지를 출력하도록 JavaScript가 생성되어 있는 것을 확인 할 수 있다.

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="validator"
      uri="http://www.springmodules.org/tags/commons-validator" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>사원정보</title>
<script type="text/javascript" src="<c:url value="/validator.do"/>"></script>
<validator:javascript formName="person" staticJavascript="false" xhtml="true"
 cdata="false"/>
<script type="text/javascript">
function save(form){
        if(!validatePerson(form)){
                return;
        }else{
                form.submit();
        }
```

Validator 태그 라이브러리 추가

Validator.do 호출설정

Validator.xml 설정 맵핑

Validation JavaScript 호출하고
에러발생하면 입력폼을 다시 출력

```
}
</script>
</head>
```

[소스 12-12] JSP 설정

12.6 Validation 예제코드

Servier-side validation을 위해서 spring에 default validator bean을 설정하고 설정파일의 위치를 지정한다. Controller에는 server-side validation을 위해 validate 수행하도록 코드를 수정, client-side validation을 위해 validator.do를 설정한다. Message properties에는 에러출력을 위한 메시지 정의를 한다. JSP에는 client-side validation을 위해 validator.do가 실행되도록 설정하고 sever-side validation에서 에러메시지 출력을 위한 message설정을 한다.

[표 12-2] 소스코드 구성

구분	코드명	설명
[src/main/java]		
Controller	lab.web.HelloWorldController.java	HelloWorld 출력
	lab.web.PersonServiceController.java	@ModelAttribute를 활용한 model 바인딩과 폼입력과 출력을 설명 Server-side validation 설정 validator.do controller 설정
Model	lab.web.model.person.java	이름, 회사, 전화번호, email 관리
[src/main/resources]		
Spring	META-INF/spring/ context-common.xml	Annotation처리를 위한 component scan (Service, DAO 용)
	META-INF/spring/ context-validator.xml	Default validator bean 설정 Validator 설정파일 위치 지정
Log4j	log4j2.xml	로그출력
Message Properties	META-INF/messages/ message-common.properties	Default locale message 설정 에러메시지 설정
	META-INF/messages/ message-common_kr.properties	한글 message 설정 에러메시지 설정
	META-INF/messages/ message-common_en.properties	영문 message 설정 에러메시지 설정
Validator	META-INF/validator/ validator-rules.xml	Client-Side validation 실행 템플릿 (JavaScript)
	META-INF/validator/validator.xml	Validaton 설정
[src/main/webapp]		
Spring MVC	WEB-INF/config/springmvc/context-srvlet.xml	Spring MVC 설정, Message Source 설정
View (JSP)	WEB-INF/jsp/hello/helloworld.jsp	HelloWorldController에서 출력
	WEB-INF/jsp/person/personForm.jsp	psersonServiceController 폼 입력 Client-side validation 설정 에러 메시지 설정
	WEB-INF/jsp/person /persionSuccess.jsp	폼 입력 결과를 출력

	WEB-INF/jsp/validator.jsp	Cliet-side validation 실행을 위한 JSP 설정
Web 설정	WEB-INF/web.xml	Spring ApplicationContext 및 Dispatcher Servlet 설정
Index	index.jsp	실행을 위한 링크

Server-side, Client-side validation 설정을 위한 Controller와 JSP 전체 소스코드는 다음과 같다.

```
@Controller
public class PersonServiceController {

	private String formView = "person/personForm";
	private String successView = "person/personSuccess";

	@Autowired
	private DefaultBeanValidator beanValidator;

	@RequestMapping(value = "/validator.do")
	protected String getValidator() throws Exception{
		return "validator";
	}

	@RequestMapping(value = "/person.do", method=RequestMethod.GET)
	protected String personInput(ModelMap model) throws Exception{
		model.addAttribute("person",new Person());
		return formView;
	}

	@RequestMapping(value = "/person.do", method=RequestMethod.POST)
	protected String regist(@ModelAttribute("person") Person person,
			BindingResult errors, ModelMap model) throws Exception {
		beanValidator.validate(person, errors);
		if (errors.hasErrors()) {
			return formView;
		}
		model.addAttribute("pinfo",person);
		return successView;
	}
}
```

Server-side validator
Default bean validator 설정

Client-side validator
validator.do 설정

Server-side validate 수행

[소스 12-13] PersonServiceController.java

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="validator"
	uri="http://www.springmodules.org/tags/commons-validator" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
```

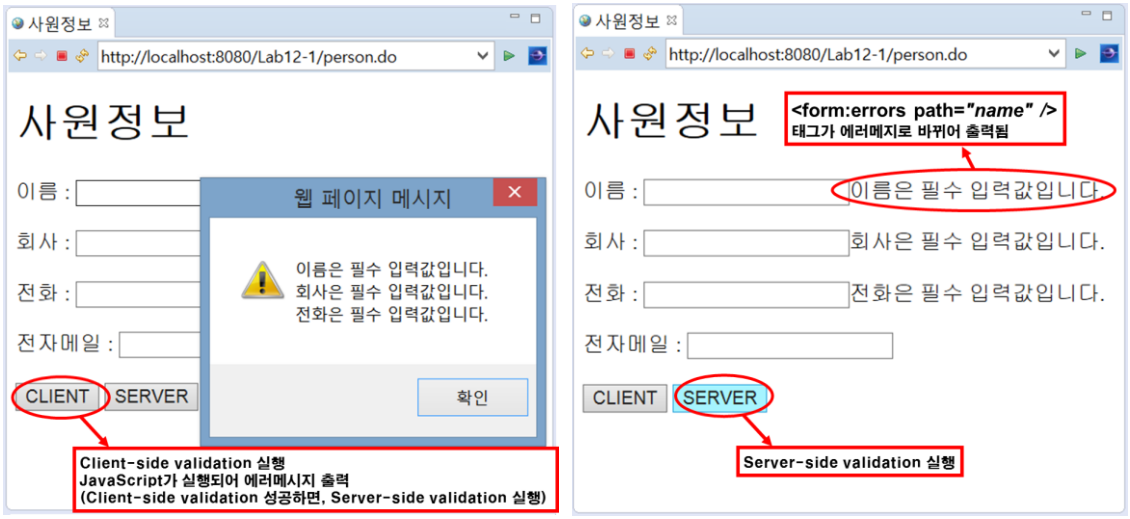
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>사원정보</title>
<script type="text/javascript" src="<c:url value="/validator.do"/>"></script>
<validator:javascript formName="person" staticJavascript="false" xhtml="true"
 cdata="false"/>
<script type="text/javascript">
function save(form){
        if(!validatePerson(form)){
                return;
        }else{
                form.submit();
        }
}
</script>
</head>
<body>
<h1>사원정보</h1>
<form:form commandName="person">
    <div><spring:message code="label.name" /> :
    <form:input path="name"/><form:errors path="name" /></div><br>
    <div><spring:message code="label.company" /> :
    <form:input path="company"/><form:errors path="company" /></div><br>
    <div><spring:message code="label.phone" /> :
    <form:input path="phone"/><form:errors path="phone" /></div><br>
    <div><spring:message code="label.email" /> :
    <form:input path="email"/><form:errors path="email" /></div><br>
    <div><input type="button" value="CLIENT" onclick="save(this.form)"/>
        <input type="submit" value="SERVER"/> </div>
</form:form>
</body>
</html>
```

Client-side validate 수행

Server-side validation 에러메시지 설정

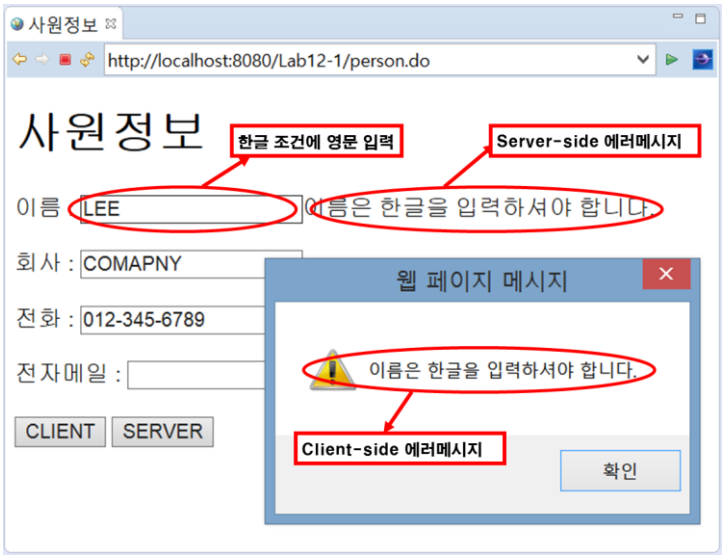
[소스 12-14] PersonForm.jsp

예제를 실행하면 입력 폼이 출력되고, 아무것도 입력하지 않고 CLIENT 또는 SERVER 버튼을 누르면 현재 이름, 회사, 전화번호가 필수조건으로 되어 있기 때문에 다음과 같이 에러메시지가 출력된다.



[그림 12-2] Validation 에러메시지 출력

이름은 필수이면서 한글 입력 조건으로 되어 있기 때문에, 영어로 입력을 하면 한글로 입력을 해야 된다는 메시지가 출력 된다. Server-side와 client-side 에러 메시지를 동시에 출력하였다.



[그림 12-3] 한글입력 조건 validation 에러 메시지 출력

13. Comprehensive Practice

입력, 수정, 삭제 및 조회 기능을 제공하는 어플리케이션을 JSP와 표준프레임워크 기반으로 각각 구현하여 아키텍처 및 구성을 비교한다. 프레임워크를 활용하면 초기에는 복잡하지만 프로그램 수와 기능이 늘어나더라도 정형화된 패턴을 유지하여 프로그램의 변경이나 유지보수에 유리하다.

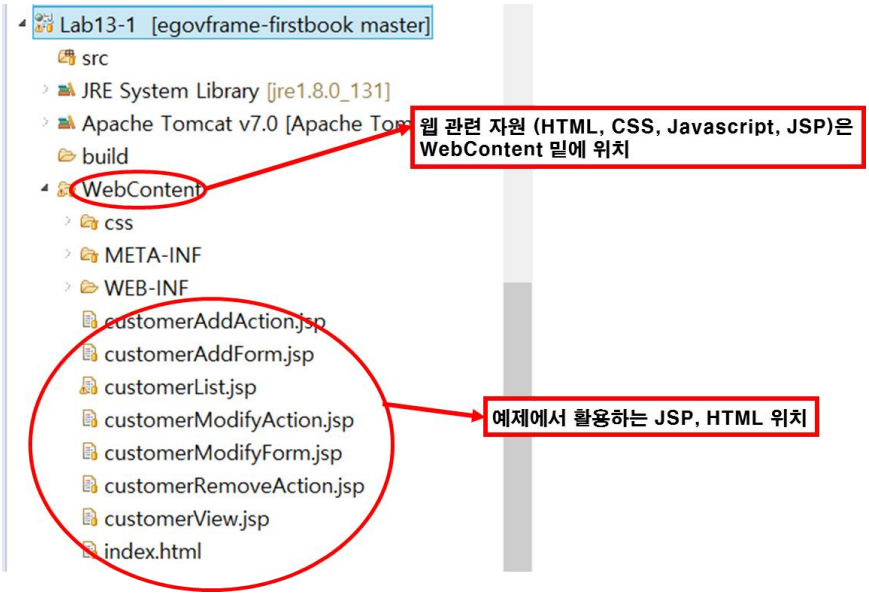
13.1 JSP 프로그래밍

Lab13-1은 사원정보를 입력, 수정, 삭제 및 조회가 가능하도록 Java SQL을 활용한 JSP 프로그램이다. MySQL DB를 실행하고 employee 테이블을 다음 스크립트를 활용하여 생성한다.

```
create table employee (
id varchar(10) primary key,
dept char(2),
name varchar(10),
age int,
phone varchar(20),
email varchar(20),
addr varchar(100));
```

[소스 13-1] employee테이블 생성스크립트

다음 그림은 예제코드의 디렉터리 구조를 설명하고 있다. 웹 자원은 WebContent 밑에 위치하게 된다.



[그림 13-1] eGovFrame Web Project 디렉터리 구조

src>main>webapp 디렉터리에 다음과 같은 코드로 구성되어 있다.

[표 13-1] Lab13-1 구성

구분	코드명	설명
HTML	index.html	프로젝트 실행 후에 default로 실행을 위한 페이지 사원정보목록으로 이동 링크
CSS	css/style.css	테이블 디자인 css파일
JSP	employeeList.jsp	사원정보의 목록을 출력
	employeeAddForm.jsp	사원정보 입력을 위한 폼을 출력
	employeeAddAction.jsp	폼으로부터 입력 받은 정보를 DB에 INSERT
	employeeView.jsp	리스트에서 선택된 사원정보를 조회
	employeeModifyForm.jsp	선택된 사원정보의 수정을 위한 폼을 출력
	employeeModifyAction.jsp	폼으로부터 입력 받은 정보를 DB로 UPDATE
	employeeRemoveAction.jsp	선택된 사원정보를 삭제

employeeList.jsp의 소스코드는 아래와 같다. DB에 연결하고 SQL을 실행한 후에 결과값을 HTML로 출력하는 형태로 구성되어 있다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER LIST</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
<body>
<%

    String dbUrl = "jdbc:mysql://127.0.0.1:3306/com";
    String dbUser = "com";
    String dbPw = "com01";
    Connection connection = null;
    PreparedStatement totalStatement = null;
    PreparedStatement listStatement = null;
    ResultSet totalResultSet = null;
    ResultSet listResultSet = null;
    try {
        Class.forName("com.mysql.jdbc.Driver");
        connection = DriverManager.getConnection(dbUrl, dbUser, dbPw);

    String listSql = "SELECT id,name FROM employee";
    listStatement = connection.prepareStatement(listSql);
    listResultSet = listStatement.executeQuery();
%>
<div id="wrap">
```

JDBC 드라이버 로드
DB접속

SQL실행

```
        <table class="tbl_List">
                <caption>Board</caption>
                <colgroup>
                        <col style="width:9%" />
                        <col style="width:16%" />
                        <col style="width:16%" />
                        <col style="width:13%" />
                        <col style="width:11%" />
                        <col style="width: ;" />
                </colgroup>
                <thead>
                        <tr>
                                <th>번호</th>
                                <th>이름</th>
                        </tr>
                </thead>
                <tbody>
<%
        while(listResultSet.next()) {
%>
                <tr>
                    <td><a
          href="<%=request.getContextPath()%>/employeeView.jsp?id=
<%=listResultSet.getString("id")%>"><%=listResultSet.getInt("id")%></a></td>
                    <td><%=listResultSet.getString("name")%></td>
                </tr>
<%
            }
%>
        </tbody>
    </table>
    <p>
        <div class="txt-rt mt20">
                <input type="button" value="사원추가"
onclick="location.href='<%=request.getContextPath()%>/employeeAddForm.jsp'" />
        </div>
</div>
<%
    } catch(Exception e) {
        e.printStackTrace();
        out.print("목록 가져오기 실패!");
    } finally {
        try {listResultSet.close();} catch(Exception e){}
        try {listStatement.close();} catch(Exception e){}
        try {connection.close();} catch(Exception e){}
    }
%>
</body>
</html>
```

결과출력

[소스 13-2] 사원정보 리스트출력 (employeeList.jsp)

리스트화면에서 게시글 입력을 누르면 입력 폼을 출력하게 된다. 입력 폼에서 입력된 값은 input 태그에 있는 변수 값인 “id”, “name”, “address”에 해당되는 value값으로 action에 지정된 customerAddAction.jsp로 POST 메소드로 전달 된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER ADD</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
<body>
<form action="<%=request.getContextPath()%>/employeeAddAction.jsp"
 method="post">
<div id="wrap">
  <table class="tbl_View">
        <caption>Employee</caption>
        <colgroup>
                <col style="width:30%" />
                <col style="width: ;" />
        </colgroup>
        <tbody>
          <tr>
            <th>사원번호</th>
            <td><input name="id" type="text" size="63" maxlength="20" /></td>
           </tr>
           <tr>
             <th>부서</th>
             <td>
                <select name="dept">
                <option selected="selected" value="">상위부서를 선택하세요.</option>
                <option value="01">기획팀</option>
                <option value="02">영업팀</option>
                <option value="03">개발팀</option>
                </select>
             </td>
           </tr>
           <tr>
            <th>이름</th>
              <td><input name="name" type="text" size="63" maxlength="20" /></td>
           </tr>
           <tr>
            <th>나이</th>
            <td><input name="age" type="text" size="63" maxlength="20" /></td>
           </tr>
           <tr>
```

Submit이 되면 employeeAddAction 으로 입력 값 전달

```
            <th>전화번호</th>
            <td><input name="phone" type="text" size="63" maxlength="20" /></td>
           </tr>
           <tr>
            <th>이메일</th>
            <td><input name="email" type="email" size="63" maxlength="30" /></td>
           </tr>
           <tr>
            <th>주소</th>
            <td><input name="address" type="text" size="63" maxlength="20" /></td>
           </tr>
           </tbody>
         </table>
         <div class="txt-rt mt20">
         <input type="button" value="글저장" onclick="form.submit()"/>
         <input type="button" value="목록"
          onclick="location.href='<%=request.getContextPath()%>/employeeList.jsp'"/>
    </div>
</div>
</form>
</body>
</html>
```

[소스 13-3] 신규 사원정보 입력 폼 (employeeAddForm.jsp)

입력 폼으로 전달 받은 메시지를 getParameter()를 이용하여 id, name, address 값을 얻어낸다. 입력 된 값은 INSERT SQL의 변수 값으로 치환되어 입력 값으로 활용된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title></title>
</head>
<body>
<%
    request.setCharacterEncoding("UTF-8");
    String id = request.getParameter("id");
    String name = request.getParameter("name");
    String dept = request.getParameter("dept");
    String age = request.getParameter("age");
    String phone = request.getParameter("phone");
    String email = request.getParameter("email");
    String address = request.getParameter("address");
    String dbUrl = "jdbc:mysql://127.0.0.1:3306/com";
    String dbUser = "com";
    String dbPw = "com01";
```

전달받은 파라미터 값을 변수로 저장

```
    Connection connection = null;
    PreparedStatement statement = null;
    try {
        Class.forName("com.mysql.jdbc.Driver");
        connection = DriverManager.getConnection(dbUrl, dbUser, dbPw);
        String sql = "INSERT INTO employee(id, name, dept, age, phone, email, addr)
        values(?,?,?,?,?,?,?)";
        statement = connection.prepareStatement(sql);
        statement.setString(1,id);
        statement.setString(2,name);
        statement.setString(3,dept);
        statement.setString(4,age);
        statement.setString(5,phone);
        statement.setString(6,email);
        statement.setString(7,address);
        statement.executeUpdate();

      response.sendRedirect(request.getContextPath()+ "/employeeList.jsp");
    } catch(Exception e) {
        e.printStackTrace();
        out.print("입력 예외 발생");
    } finally {
        try {statement.close();} catch(Exception e){}
        try {connection.close();} catch(Exception e){}
    }
%>
</body>
</html>
```

DB연결, SQL문 준비,
입력 변수 치환 및 SQL실행

[소스 13-4] 신규 사원정보 입력 처리 (employeeAddAction.jsp)

리스트 페이지에서 id를 선택하면 id에 해당되는 사원정보를 보여주는 employeeView.jsp가 작성되었으며, 화면에서 수정, 삭제, 목록으로 돌아가기를 선택할 수 있다. 소스코드에는 id를 파라미터로 전달받아 DB에서 select하여 화면에 출력한다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER VIEW</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
 <script>
        function init(code){
                var form   = document.hmsForm;
                selectBox(form.dept, code);
        }
```

SELECT에 코드 값에 해당되는 부서명을 selected로 설정

```
        function selectBox(sel, val){
                var selOptions = sel.options;
                var size = selOptions.length;
                for(var i = 0; i<size; i++){
                        if(selOptions[i].value == val){
                                selOptions[i].selected = true;
                                break;
                        }
                }
        }
 </script>
<%
    if(request.getParameter("id") == null) {
        response.sendRedirect(request.getContextPath()+"/employeeList.jsp");
    } else {
        String id = request.getParameter("id");
        String dbUrl = "jdbc:mysql://127.0.0.1:3306/com";
        String dbUser = "com";
        String dbPw = "com01";
        Connection connection = null;
        PreparedStatement statement = null;
        ResultSet resultSet = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(dbUrl, dbUser, dbPw);
            String sql = "SELECT id, name, dept, age, phone, email, addr
                          FROM employee WHERE id=?";
            statement = connection.prepareStatement(sql);
            statement.setString(1, id);
            resultSet = statement.executeQuery();
            if(resultSet.next()) {
%>
<body onload="javascript:init(<%=resultSet.getString("dept")%>)">
<form name="hmsForm">
<div id="wrap">
        <table class="tbl_View">
                <caption>Employee</caption>
                <colgroup>
                        <col style="width:30%" />
                        <col style="width: ;" />
                </colgroup>
                <tbody>
                        <tr>
                                <th>사원번호</th>
                                <td><%=id%></td>
                        </tr>
                        <tr>
                                <th>이름</th>
                                <td><%=resultSet.getString("name")%></td>
                        </tr>
```

파라미터로 받은 id로 SELECT

```
                        <tr>
                                <th>부서</th>
                                <td>
                                        <select name="dept" disabled>
                                            <option value="01">기획팀</option>
                                            <option value="02">영업팀</option>
                                            <option value="03">개발팀</option>
                                        </select>
                                </td>
                        </tr>
                        <tr>
                                <th>나이</th>
                                <td><%=resultSet.getString("age")%></td>
                        </tr>
                        <tr>
                                <th>전화번호</th>
                                <td><%=resultSet.getString("phone")%></td>
                        </tr>
                        <tr>
                                <th>이메일</th>
                                <td><%=resultSet.getString("email")%></td>
                        </tr>
                        <tr>
                                <th>주소</th>
                                <td><%=resultSet.getString("addr")%></td>
                        </tr>
                </tbody>
        </table>
        <div class="txt-rt mt20">
        <input type="button" value="목록"
    onclick="location.href='<%=request.getContextPath()%>/employeeList.jsp'"/>
        <input type="button" value="수정" onclick="location.href=
     '<%=request.getContextPath()%>/employeeModifyForm.jsp?id=<%=id%>'"/>
        <input type="button" value="삭제" onclick="location.href=
   '<%=request.getContextPath()%>/employeeRemoveAction.jsp?id=<%=id%>'"/>
    </div>
</div>
<%      }
        } catch(Exception e) {
            e.printStackTrace();
            out.println("VIEW ERROR!");
        } finally {
            try {resultSet.close();} catch(Exception e){}
            try {statement.close();} catch(Exception e){}
            try {connection.close();} catch(Exception e){}
        }
    }
%>
</form></body></html>
```

[소스 13-5] 사원정보 조회 (employeeView.jsp)

사원정보를 수정하는 수정폼은 처음 페이지가 로딩 될 때 DB에서 값을 조회하여 보여주는 것을 빼고는 입력폼과 코드가 비슷하다. 수정처리의 경우도 SQL이 UPDATE가 실행되는 점을 빼고는 입력처리와 비슷하다. JSP로 처리가 될 때에는 반복되는 JSP별로 반복되는 코드가 많다. 입력과 수정을 하나의 모듈로 처리를 해서 반복되는 요소를 줄일 수도 있으나 내부적으로 신규입력 상태인지 수정인지를 조건으로 분기 처리를 해야 되기 때문에 코드가 복잡해진다. 삭제 처리의 경우에는 id를 전달 받아 해당 id를 DELETE하도록 SQL을 실행하게 된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER REMOVE ACTION</title>
</head>
<body>
<%
    if(request.getParameter("id") == null) {
        response.sendRedirect(request.getContextPath()+ "/customerList.jsp");
    } else {
        String id = request.getParameter("id");
        String dbUrl = "jdbc:mysql://127.0.0.1:3306/com";
        String dbUser = "com";
        String dbPw = "com01";
        Connection connection = null;
        PreparedStatement statement = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(dbUrl, dbUser, dbPw);
            String sql = "DELETE FROM employee WHERE id=?";
            statement = connection.prepareStatement(sql);
            statement.setString(1, id);
            if(statement.executeUpdate()==1){
                response.sendRedirect(request.getContextPath()+ "/employeeList.jsp");
            }
        } catch(Exception e) {
            e.printStackTrace();
            out.print("REMOVE ERROR!");
        } finally {
            try {statement.close();} catch(Exception e){}
            try {connection.close();} catch(Exception e){}
        }
    } %>
</body>
</html>
```

삭제(DELETE)하고 리스트로 이동

[소스 13-6] 사원정보 삭제 (employeeRemoveAction.jsp)

실행을 위해서는 실행시킬 JSP나 html을 선택하고 마우스 오른쪽 버튼을 눌러 Run As > Run on Server를 클릭하고 어느 WAS에서 실행시킬 것인지 선택한 다음 Finish를 누르면 실행된다. 리스트 출력하는 employeeList.jsp를 실행하면, 다음과 같은 결과를 얻을 수 있으며, 웹 브라우저에 직접 URL을 입력하여도 같은 결과화면이 출력되는 것을 볼 수 있다.

[그림 13-2] 리스트 실행결과 (employeeList.jsp)

사원추가 버튼을 누르면 다음과 같이 입력 폼 employeeAddForm.jsp가 실행된다. 글저장을 누르면 employeeAddAction.jsp를 호출하여 DB에 저장된다.

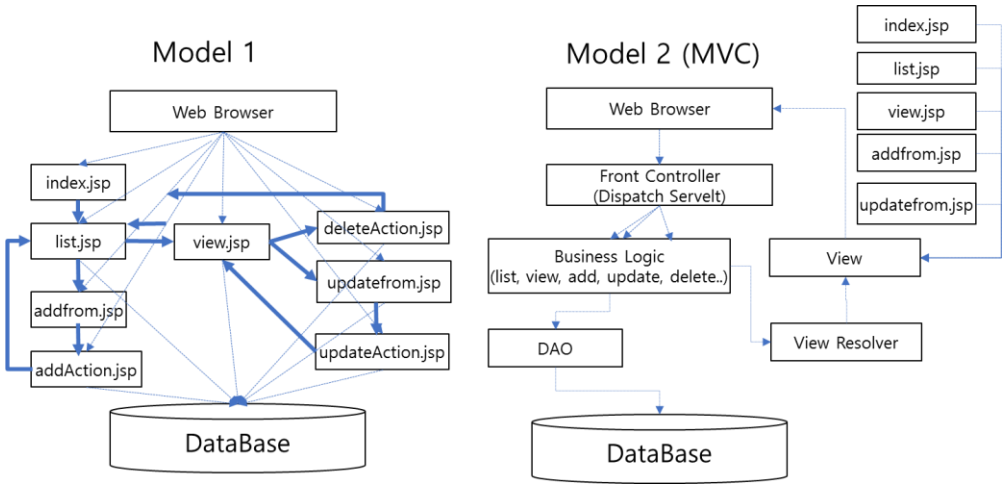
[그림 13-3] 입력폼 실행결과 (employeeAddForm.jsp)

리스트에서 id를 선택하면 사원정보 조회가 가능하다. 조회화면에서 수정을 하거나 삭제할 수 있다.

[그림 13-4] 조회 실행결과 (employeeView.jsp)

13.2 표준프레임워크 프로그래밍

앞에서 살펴본 JSP예제 코드는 모델1으로 구성되어 비즈니스 로직마다 모듈이 구성되어 필요한 로직이 생길 때마다 모듈이 증가하게 되고, 모듈 간의 관계도 매우 복잡해지게 된다. 또한 웹 브라우저로부터의 요청이 직접 개별 JSP로 직접 전달이 되고, DB에도 직접 접근하게 되어 중앙에서의 통제나 제어가 어렵게 된다. 다음 그림은 모델1의 코드 구조와 모델2인 MVC 및 표준프레임워크 기반으로 구현되었을 때의 구조를 설명하고 있다. 프레임워크를 활용할 때는 Front Controller인 Dispatcher Servlet을 통해서 요청을 전달 받게 되며, 요청된 처리별로 비즈니스 로직으로 분기가 된다. DB는 DAO를 통해서 접근하게 되며, 처리 결과는 Model을 통해서 View인 JSP를 통해 출력하게 된다. 여기의 JSP들은 화면출력을 위한 템플릿 역할만을 하게 되며, 직접 실행이 되지 않는다. 실행이 되지 않는 WEB-INF밑에 위치를 시키고, 실행이 된다고 하더라도 템플릿 내용만 출력된다. 필요한 기능이 증가되는 경우에 모듈이 증가되는 것이 아니고, 현재의 구조도 그대로 유지하게 되어 대용량의 시스템 구축에 적합하고 변경, 유지보수가 편리하다.



[그림 13-5] 예제 코드 모델1과 모델2 구조 비교

예제 13-1을 표준프레임워크 기반으로 구현한 결과는 예제 13-2와 같다. 소스코드 구성은 다음과 같다.

[표 13-2] Lab13-2 소스코드 구성

구분	코드명	설명
[src/main/java]		
Controller	lab.EmployeeServiceController.java	@ModelAttribute를 활용한 model 바인딩과 폼입력과 출력을 설명
Service	EmployeeService.java	사원정보관리를 위한 인터페이스
	EmployeeServiceImpl.java	사원정보관리 구현
DAO	EmployeeDAO.java	사원정보관리 데이터베이스 처리
Model	lab.Employee.java	사원정보관리 클래스(VO)
[src/main/resources]		
Spring	META-INF/spring/ context-common.xml	Annotation처리를 위한 component scan (Service, DAO 용)
	META-INF/spring/ context-datasource.xml	DBCP를 활용한 데이터 풀 제공
	META-INF/spring/ context-mybatis.xml	Spring과 MyBatis 연결설정
	META-INF/spring/jdbc.properties	DB연결정보 설정
MyBatis	META-INF/sqlmap/mappers/ sql-mybatis-config.xml	MyBatis 환경설정
	META-INF/sqlmap/mappers/ lab-dao-class.xml	SQL설정파일인 Mapper정의
Log4j	log4j2.xml	로그출력
[src/main/webapp]		
Spring MVC	WEB-INF/config/springmvc/ context-srvlet.xml	Spring MVC 설정
View (JSP)	WEB-INF/jsp/employee/ employeeAddForm.jsp	사원정보 입력 폼 출력
	WEB-INF/jsp/employee/ employeeList.jsp	사원정보 목록 출력
	WEB-INF/jsp/employee/	사원정보 수정 폼 출력

	employeeModifyForm.jsp	
	WEB-INF/jsp/employee /employeeView.jsp	사원정보 상세보기 출력
스타일 쉬트	css/style.css	JSP 페이지 스타일 정의
Web 설정	WEB-INF/web.xml	Spring ApplicationContext 및 Dispatcher Servlet 설정
Index	index.jsp	실행을 위한 링크

구현 방법은 다음과 같다.

1) eGovFrame Web Project를 생성

2) MySQL JDBC 드라이버 설정

MySQL DB 활용을 위해 JDBC 드라이버 dependency를 pom.xml에 설정

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.42</version>
</dependency>
```

[소스 13-6] pom.xml에 MySQL JDBC드라이버 의존성 설정

3) Spring 설정

Service와 DAO에서 활용하기 위한 annotation 설정과 DB연결과 활용을 위한 MyBatis 관련 환경을 설정한다.

- context-common.xml: Annotation활용 위한 component-scan 설정 (소스 9-16)
- context-datasource.xml: DB연결(풀링 활용을 위한 DBCP설정) (소스 9-17)
- jdbc.properties: DB연결 환경 변수 (소스 9-18)
- context-mybatis.xml: Spring에서 MyBatis활용을 위한 sqlsession Bean 설정 (소스 9-15)

4) MyBatis 설정 (sql-mybatis-config.xml: alias와 mapper파일 위치를 설정)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
<typeAliases>
        <typeAlias alias="employeeVO" type="lab.Employee" />
</typeAliases>
<mappers>
        <mapper resource ="META-INF/sqlmap/mappers/lab-dao-class.xml" />
</mappers>
</configuration>
```

[소스 13-7] MyBatis의 VO alias와 mapper 파일 위치 설정

5) MyBatis SQL Mapper설정

DAO에서 DB 입력, 수정, 삭제, 조회를 수행하는 SQL을 설정하고, SQL실행에 필요한 입력 파라미터와 결과 값을 반환하는 오브젝트인 resultmap을 설정한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="employee">

        <resultMap id="employeeResult" type="employeeVO">
                <id property="id" column="id" />
                <result property="name" column="name" />
                <result property="dept" column="dept" />
                <result property="age" column="age" />
                <result property="phone" column="phone" />
                <result property="email" column="email" />
                <result property="address" column="addr" />
        </resultMap>

        <insert id="insertEmployee" parameterType="employeeVO">
                <![CDATA[
                insert into employee (id, name, dept, age, phone, email, addr)
                values   (#{id}, #{name}, #{dept}, #{age}, #{phone}, #{email},
                 #{address})
                ]]>
        </insert>

        <update id="updateEmployee" parameterType="employeeVO">
                <![CDATA[
                update employee set name=#{name}, dept=#{dept}, age=#{age},
                phone=#{phone}, email=#{email}, addr=#{address}
                where id = #{id}
                ]]>
        </update>

        <delete id="deleteEmployee" parameterType="employeeVO">
                <![CDATA[
                delete from employee where id = #{id}
                ]]>
        </delete>

        <select id="selectEmployeeList" parameterType="employeeVO"
        resultMap="employeeResult">
                <![CDATA[
                select    id, name from     employee
                ]]>
        </select>

        <select id="selectEmployee" parameterType="employeeVO"
```

```
          resultMap="employeeResult">
                <![CDATA[
                select    id, name, dept, age, phone, email, addr
                from      employee
                where id = #{id}
                ]]>
        </select>
</mapper>
```

[소스 13-8] SQL Mapper 설정

6) Spring 활용을 위한 웹 어플리케이션 설정 (web.xml) (소스10-6과 동일)

7) Controller annotation위한 component-scan 및 View Resolver 설정을 위한 context-servlet.xml 작성 (소스10-7)

8) Model 작성 (Employee.java)

MyBatis에서 파라미터와 resultmap 정의에 필요한 VO 클래스를 정의

```
public class Employee {
        String id;
        String name;
        String dept;
        int age;
        String phone;
        String email;
        String address;
        public String getId() {
                return id;
        }
        public void setId(String id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getDept() {
                return dept;
        }
        public void setDept(String dept) {
                this.dept = dept;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int age) {
                this.age = age;
        }
```

```
        public String getPhone() {
                return phone;
        }
        public void setPhone(String phone) {
                this.phone = phone;
        }
        public String getEmail() {
                return email;
        }
        public void setEmail(String email) {
                this.email = email;
        }
        public String getAddress() {
                return address;
        }
        public void setAddress(String address) {
                this.address = address;
        }
}
```

[소스 13-9] Model 작성

9) DAO작성 (EmployeeDAO.java)

DAO에서 MyBatis에서 정의한 SQL ID와 실행에 필요한 파라미터를 정의하여 입력, 수정, 삭제 작업을 위한 메소드를 정의하며, 서비스에서 호출되어 실행된다.

```
package lab;

import java.util.List;

import org.springframework.stereotype.Repository;
import egovframework.rte.psl.dataaccess.EgovAbstractMapper;

@Repository("employeeDAO")
public class EmployeeDAO extends EgovAbstractMapper {

   public void insertEmployee(Employee vo) {
         insert("employee.insertEmployee", vo);
     }

     public int updateEmployee(Employee vo) {
         return update("employee.updateEmployee", vo);
     }

     public int deleteEmployee(Employee vo) {
         return delete("employee.deleteEmployee", vo);
     }

     public List<Employee> selectEmployeeList(Employee vo) {
         return selectList("employee.selectEmployeeList", vo);
```

```
        }

        public Employee selectEmployee(Employee vo) {
            return (Employee) selectOne("employee.selectEmployee", vo);
        }

    }
```

[소스 13-10] DAO 작성

10) Service 작성

서비스에서 작성할 메소드에 대해 인터페이스를 정의

```
public interface EmployeeService  {
    public void insertEmployee(Employee employeeVO) throws Exception;
    public void updateEmployee(Employee employeeVO) throws Exception;
    public void deleteEmployee(Employee employeeVO) throws Exception;
    public List<Employee> selectEmployeeList(Employee employeeVO)
        throws Exception;
    public Employee selectEmployee(Employee employeeVO) throws Exception;
}
```

[소스 13-11] Service Interface 작성

인터페이스에서 정의된 메소드를 구현하며, DAO의 메소드를 호출한다. 별도 비즈니스 로직이 없기 때문에 단순히 호출만 하고 있으나, 실제 프로그램에서는 여러 개의 DAO를 호출할 수도 있고, 다른 서비스의 메소드를 호출 한 후에 DAO를 호출하는 등의 작업이 필요하다.

```
@Service("employeeService")
public class EmployeeServiceImpl implements EmployeeService {

    @Resource(name = "employeeDAO")
    public EmployeeDAO employeeDAO;

        public void insertEmployee(Employee employeeVO) throws Exception {
                employeeDAO.insertEmployee(employeeVO);
        }
        public void updateEmployee(Employee employeeVO) throws Exception {
                employeeDAO.updateEmployee(employeeVO);
        }
        public void deleteEmployee(Employee employeeVO) throws Exception {
                employeeDAO.deleteEmployee(employeeVO);
        }
        public List<Employee> selectEmployeeList(Employee employeeVO) throws
Exception {
                return employeeDAO.selectEmployeeList(employeeVO);
        }
        public Employee selectEmployee(Employee employeeVO) throws Exception {
                return employeeDAO.selectEmployee(employeeVO);
```

```
        }
}
```

[소스 13-12] Service Implementation 작성

11) Controller 작성

Controller에서는 사용자가 요청한 URL을 매핑하여 해당되는 서비스 메소드를 실행하도록 한다. 같은 URL이지만 GET방식으로 호출이 되는 경우에는 입력 또는 수정 폼이 생성된다. 출력된 폼에서 submit이 되면 POST방식으로 호출이 되어 실제 입력 및 수정을 위한 서비스 메소드가 호출 된다.

```
@Controller
public class EmployeeServiceController {
        @Resource(name="employeeService")
        EmployeeService employeeservice;

        @RequestMapping(value = "/employeeAdd.do", method=RequestMethod.GET)
        protected String employeeAddForm(ModelMap model) throws Exception{
                model.addAttribute("employee",new Employee());
                return "employee/employeeAddForm";
        }
        @RequestMapping(value = "/employeeAdd.do", method=RequestMethod.POST)
        protected String employeeAdd(@ModelAttribute("employee") Employee
            command, BindingResult errors, ModelMap model) throws Exception {
                if (errors.hasErrors()) {
                        return "employee/employeeAddForm";
                }
                employeeservice.insertEmployee(command);
                return "redirect:/employeeList.do";
        }
        @RequestMapping(value = "/employeeList.do")
        protected String employList(ModelMap model) throws Exception{
                Employee vo = new Employee();
                List<Employee> employeelist =
                                employeeservice.selectEmployeeList(vo);
                model.addAttribute("employeelist",employeelist);
                return "employee/employeeList";
        }
        @RequestMapping(value = "/employeeView.do")
        protected String employeeAdd(@RequestParam String id, ModelMap model)
        throws Exception {
                Employee vo = new Employee();
                vo.setId(id);
                Employee command = employeeservice.selectEmployee(vo);
                model.addAttribute("employeeinfo",command);
                return "employee/employeeView";
```

사원정보 입력을 위한 폼 출력

신규로 사원정보를 입력하고, 리스트를 출력

사원정보 리스트를 조회하고 리스트를 출력

```
        }
                                                    선택된 사원정보 조회하고 정보를 출력
        @RequestMapping(value = "/employeeModify.do",
        method=RequestMethod.GET)
        protected String employeeModifyForm(@RequestParam String id,ModelMap
model) throws Exception{
                Employee vo = new Employee();
                vo.setId(id);
                Employee command = employeeservice.selectEmployee(vo);
                model.addAttribute("employee",command);
                return "employee/employeeModifyForm";
        }
                                                    수정을 위한 수정 폼을 출력
        @RequestMapping(value = "/employeeModify.do",
        method=RequestMethod.POST)
        protected String employeeModify(@ModelAttribute("employee") Employee
            command, BindingResult errors, ModelMap model) throws Exception {
                if (errors.hasErrors()) {
                        return "employee/employeeModifyForm";
                }
                employeeservice.updateEmployee(command);
                return "redirect:/employeeView.do?id="+ command.getId();
        }
                                                    수정된 사원정보를 반영하고 조회화면 출력
        @RequestMapping(value = "/employeeRemove.do")
        protected String employeeRemove(@RequestParam String id) throws Exception
{
                Employee vo = new Employee();
                vo.setId(id);
                employeeservice.deleteEmployee(vo);
                return "redirect:/employeeList.do";
        }
                                                    선택된 사원정보를 삭제하고 목록 출력
}
```

[소스 13-13] Controller 작성

12) View(JSP) 작성

View는 controller에서 서비스 메소드를 실행한 후에 결과를 화면에 보여주게 된다. 여기에서 활용되는 JSP는 자바스크립트 등 클라이언트 로직을 제외한 서버 단의 비즈니스 로직은 가지고 있지 않으며, model로부터 데이터를 전달 받아 화면에 출력하여 HTML로 화면을 보여주게 된다. 입력 폼은 신규 추가 버튼을 눌렀을 때 보여지게 되며, submit이 될 때 지정된 action으로 input에 입력된 값이 클래스로 바인딩이 되어 controller의 @ModelAttribte로 매핑되어 전달 된다

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER ADD</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
<body>
<form:form commandName="employee" action="employeeAdd.do">
<div id="wrap">
        <table class="tbl_View">
         <caption>Employee</caption>
                <colgroup>
                <col style="width:30%" />
                <col style="width: ;" />
        </colgroup>
        <tbody>
        <tr>
          <th>사원번호</th>
        <td><form:input path="id" type="text" size="63" maxlength="20" /></td>
        </tr>
        <tr>
          <th>부서</th>
        <td>
        <form:select path="dept">
    <form:option selected="selected" value="">상위부서를 선택하세요.</form:option>
          <form:option value="01">기획팀</form:option>
          <form:option value="02">영업팀</form:option>
          <form:option value="03">개발팀</form:option>
    </form:select>
        </td>
        </tr>
        <tr>
          <th>이름</th>
        <td><form:input path="name" type="text" size="63" maxlength="20" /></td>
        </tr>
        <tr>
          <th>나이</th>
        <td><form:input path="age" type="text" size="63" maxlength="20" /></td>
        </tr>
        <tr>
          <th>전화번호</th>
        <td><form:input path="phone" type="text" size="63" maxlength="20" /></td>
        </tr>
        <tr>
          <th>이메일</th>
        <td><form:input path="email" type="email" size="63" maxlength="30" /></td>
        </tr>
        <tr>
          <th>주소</th>
        <td><form:input path="address" type="text" size="63" maxlength="20" /></td>
```

Model 클래스와 @Attribute로 바인딩을 위한 매핑을 정의하고 submit 시의 action 정의

입력은 spring의 form:input 태그를 활용

```
        </tr>
        </tbody>
        </table>
        <div class="txt-rt mt20">
        <input type="button" value="글저장" onclick="form.submit()"/>
        <input type="button" value="목록" onclick="location.href='employeeList.do'"/>
    </div>
</div>
</form:form>
</body>
</html>
```

[소스 13-14] 입력 폼 작성

리스트 출력의 경우 SQL 실행결과로 출력된 레코드가 클래스에 담겨서 리스트 형태로 출력된다. 따라서 리스트에서 클래스를 하나씩 꺼내고 메소드를 view 태그에 매핑하여 출력한다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER LIST</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
<body>
<form:form commandName="employee" action="employeeList.do">
<div id="wrap">

        <table class="tbl_List">
                <caption>Board</caption>
                <colgroup>
                        <col style="width:9%" />
                        <col style="width:16%" />
                        <col style="width:16%" />
                        <col style="width:13%" />
                        <col style="width:11%" />
                        <col style="width: ;" />
                </colgroup>
                <thead>
                        <tr>
                                <th>번호</th>
                                <th>이름</th>
                        </tr>
                </thead>
                <tbody>
```

리스트로 입력된 클래스를 루프로 출력

```
                <c:forEach items="${employeelist}" var="employeeinfo">
                <tr>
                    <td><a href="employeeView.do?id=${employeeinfo.id}">
                        ${employeeinfo.id}</a></td>
                    <td>${employeeinfo.name}</td>
                </tr>
        </c:forEach>
        </tbody>
    </table>
    <p>
        <div class="txt-rt mt20">
                <input type="button" value="사원추가"
                        onclick="location.href='employeeAdd.do'" />
        </div>
</div>
</form:form>
</body>
</html>
```

클래스에 저장된 변수 값을 매핑하여 출력

[소스 13-15] 리스트 출력 작성

수정 폼의 출력은 입력 폼과 비슷하며, 화면이 보여질 때 저장되어 있던 값이 select 되어 보여지는 것이 다르다. input 태그에 value의 형태로 초기값을 가지고 출력 된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER MODIFY FORM</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
<body>
<form:form commandName="employee" action="employeeModify.do">
<div id="wrap">
       <table class="tbl_View">
           <caption>Employee</caption>
               <colgroup>
                 <col style="width:30%" />
                 <col style="width: ;" />
               </colgroup>
         <tbody>
         <tr>
                 <th>사원번호</th>

                 <td><form:input path="id" value="${employeeinfo.id}" type="text"
size="63" maxlength="20" /></td>
         </tr>
```

input 태그에 value 값을 주어 초기 값을 정의 하여 출력하고 사용자가 수정 가능

```
        <tr>
                <th>부서</th>
                <td>
                   <form:select path="dept">
                   <form:option  selected="selected"  value="">상위부서를  선택하세
요.</form:option>
                   <form:option value="01">기획팀</form:option>
                   <form:option value="02">영업팀</form:option>
                   <form:option value="03">개발팀</form:option>
                </form:select>
                </td>
        </tr>
        <tr>
                <th>이름</th>
                <td><form:input  path="name"  type="text"  size="63"  maxlength="20"
/></td>
        </tr>
        <tr>
                <th>나이</th>
                <td><form:input  path="age"  type="text"  size="63"  maxlength="20"
/></td>
        </tr>
        <tr>
                <th>전화번호</th>
                <td><form:input  path="phone"  type="text"  size="63"  maxlength="20"
/></td>
        </tr>
        <tr>
                <th>이메일</th>
                <td><form:input  path="email"  type="email"  size="63"  maxlength="30"
/></td>
        </tr>
        <tr>
                <th>주소</th>
                <td><form:input path="address" type="text" size="63" maxlength="20"
/></td>
        </tr>
        </tbody>
        </table>
        <div class="txt-rt mt20">
        <input type="button" value="글저장" onclick="form.submit()"/>
        <input type="button" value="목록" onclick="location.href='employeeList.do'"/>
    </div>
</div>
</form:form>
</body></html>
```

[소스 13-16] 수정 폼 작성

사원정보 조회화면은 선택된 단일 정보가 select되어 화면에 출력된다. 조회 화면에서 선택 정보를 수정하거나 삭제할 수 있도록 링크가 제공 된다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER VIEW</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
</head>
 <script>
        function init(code){
                var form   = document.hmsForm;
                selectBox(form.dept, code);
        }

        function selectBox(sel, val){
                var selOptions = sel.options;
                var size = selOptions.length;
                for(var i = 0; i<size; i+ + ){
                        if(selOptions[i].value == val){
                                selOptions[i].selected = true;
                                break;
                        }
                }
        }
 </script>
<body onload="javascript:init(${employeeinfo.dept})">
<form name="hmsForm">
<div id="wrap">
        <table class="tbl_View">
                <caption>Employee</caption>
                <colgroup>
                        <col style="width:30%" />
                        <col style="width: ;" />
                </colgroup>
                <tbody>
                        <tr>
                                <th>사원번호</th>
                                <td>${employeeinfo.id}</td>
                        </tr>
                        <tr>
                                <th>이름</th>
                                <td>${employeeinfo.name}</td>
                        </tr>
                        <tr>
                                <th>부서</th>
                                <td>
```

```
                                        <select name="dept" disabled>
                                                <option value="01">기획팀</option>
                                                <option value="02">영업팀</option>
                                                <option value="03">개발팀</option>
                                        </select>
                                </td>
                        </tr>
                        <tr>
                                <th>나이</th>
                                <td>${employeeinfo.age}</td>
                        </tr>
                        <tr>
                                <th>전화번호</th>
                                <td>${employeeinfo.phone}</td>
                        </tr>
                        <tr>
                                <th>이메일</th>
                                <td>${employeeinfo.email}</td>
                        </tr>
                        <tr>
                                <th>주소</th>
                                <td>${employeeinfo.address}</td>
                        </tr>
                </tbody>
        </table>
        <div class="txt-rt mt20">
        <input type="button" value="목록" onclick="location.href='employeeList.do'"/>
        <input type="button" value="수정"
                onclick="location.href='employeeModify.do?id=${employeeinfo.id}'"/>
        <input type="button" value="삭제"
                onclick="location.href='employeeRemove.do?id=${employeeinfo.id}'"/>
    </div>
</div>
</form>
</body>
</html>
```

[소스 13-17] 상세보기 폼 작성

13.3 국제화 및 validation 기능 추가 구현하기

앞에서 살펴본 표준프레임워크 코드는 JSP로 구현된 기존 코드를 MVC 형태의 기능단위로 재개발 하였다. 구현된 코드를 프로젝트에서 활용하기 위해 필수적인 기능 중에 부족한 기능을 보완하여 완성도를 높이고자 한다. 11장, 12장에서 배운 내용을 기반으로 validation 및 국제화 기능은 다음과 같이 추가 구현한다. 이전 코드에서 변경이 되는 부분은 볼드로 표시하였다.

[표 13-3] Lab13-3 소스코드 구성

구분	코드명	설명
[src/main/java]		
Controller	lab.EmployeeServiceController.java	@ModelAttribute를 활용한 model 바인딩과 폼입력과 출력을 설명
Service	EmployeeService.java	사원정보관리를 위한 인터페이스
	EmployeeServiceImpl.java	사원정보관리 구현
DAO	EmployeeDAO.java	사원정보관리 데이터베이스 처리
Model	lab.Employee.java	사원정보관리 클래스(VO)
[src/main/resources]		
Spring	META-INF/spring/ context-validator.xml	Default validator bean 설정 Validator 설정파일 위치 지정
Message Properties	META-INF/messages/ message-common.properties	Default locale message 설정 에러메시지 설정
	META-INF/messages/ message-common_kr.properties	한글 message 설정 에러메시지 설정
	META-INF/messages/ message-common_en.properties	영문 message 설정 에러메시지 설정
Validator	META-INF/validator/ validator-rules.xml	Client-Side validation 실행 템플릿 (JavaScript)
	META-INF/validator/validator.xml	Validaton 설정
[src/main/webapp]		
Spring MVC	WEB-INF/config/springmvc/ context-srvlet.xml	Spring MVC 설정
View (JSP)	WEB-INF/jsp/employee/ employeeAddForm.jsp	사원정보 입력 폼 출력 Client validation 설정
	WEB-INF/jsp/employee/ employeeList.jsp	사원정보 목록 출력 Client validation 설정
	WEB-INF/jsp/employee/ employeeModifyForm.jsp	사원정보 수정 폼 출력 Client validation 설정
	WEB-INF/jsp/validator.jsp	Cliet-side validation 실행을 위한 JSP 설정
스타일 쉬트	css/style.css	JSP 페이지 스타일 정의
Web 설정	WEB-INF/web.xml	Spring ApplicationContext 및 Dispatcher Servlet 설정
Index	index.jsp	실행을 위한 링크

1) 메시지 및 국제화 Spring 설정

Validation 결과 출력과 국제화 설정을 위한 메시지 설정과 출력언어 관련 정보 수집을 위한 locale interceptor를 Spring MVC 설정파일인 context-servlet.xml에 기술한다.

```
<?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:p="http://www.springframework.org/schema/p"
        xmlns:context="http://www.springframework.org/schema/context"
```

```
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
                                    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:component-scan base-package="lab" />

   <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"
        p:prefix="/WEB-INF/jsp/" p:suffix=".jsp" />

  <bean id="localeChangeInterceptor"
        class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
        p:paramName="lang" />

  <bean id="localeResolver"
        class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />

  <bean class="org.springframework.web.servlet.mvc.method.annotation
                .RequestMappingHandlerMapping">
      <property name="interceptors">
          <list>
              <ref bean="localeChangeInterceptor"/>
          </list>
      </property>
  </bean>
  <bean class="org.springframework.web.servlet.mvc.method.annotation.
          RequestMappingHandlerAdapter" />

   <bean id="messageSource"
      class="org.springframework.context.support.ResourceBundleMessageSource">
       <property name="basenames">
          <list>
              <value>META-INF/messages/message-common</value>
          </list>
       </property>
   </bean>
</beans>
```

국제화 language 설정

RequestMapping Hadler 설정
localeChaneInterceptor 때문에 설정

메시지 설정

[소스 13-18] context-servlet 설정

2) Validation 설정

Validation활용 beandValidator 생성을 위한 Spring 설정을 context-validator.xml에 추가한다. 설정에는 validator-rules.xml과 validator.xml을 위치정보가 포함된다.

```
<?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-4.0.xsd">
                                                    validatorFactory활용 beanValidator 생성
        <bean id="beanValidator"
                class="org.springmodules.validation.commons.DefaultBeanValidator">
                 <property name="validatorFactory" ref="validatorFactory"/>
        </bean>

        <bean id="validatorFactory"
                class="org.springmodules.validation.commons.DefaultValidatorFactory">
            <property name="validationConfigLocations">
                <list>
                    <value>classpath:META-INF/validator/validator-rules.xml</value>
                    <value>classpath:META-INF/validator/*.xml</value>
                </list>                                  Validation 설정파일 위치
            </property>
        </bean>

</beans>
```

[소스 13-19] context-validator 설정

validator-rules.xml 과 context-validator.xml 설정을 추가한다. Validator-rules.xml은 그대로 활용하면 되고, validator.xml에는 id, name, dept, age, phone, email이 가지는 validation(유효성 검증) 조건을 required(필수), korean(한글), integer(정수) 등의 형태로 설정한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons Validator Rules Configuration
1.1//EN" "http://jakarta.apache.org/commons/dtds/validator_1_1.dtd">

<form-validation>
    <formset>
        <form name="employee">
                <field property="id" depends="required,integer">      필수값, 정수
                        <arg0 key="label.id" />
                        </field>
                <field property="name" depends="required,korean">     필수값, 한글
                        <arg0 key="label.name" />
                        </field>
                  <field property="dept" depends="required">
                        <arg0 key="label.dept" />
                        </field>
                  <field property="age" depends="required,integer">
                        <arg0 key="label.age" />
                        </field>
                  <field property="phone" depends="required">
                        <arg0 key="label.phone" />
```

```
                    </field>
                <field property="email" depends="email">        이메일
                    <arg0 key="label.email" />
                    </field>
        </form>
    </formset>
</form-validation>
```

[소스 13-20] validator.xml 설정

3) 메시지 설정

국제화에 설정을 통해 출력되는 페이지와 validator실행 결과를 언어별로 출력할 수 있도록 메시지를 설정한다. 에러 메시지는 소스12-4, 12-5와 유사하다. 속성이 추가되었으므로 해당 라벨을 더 추가한다.

```
label.id=아이디
label.name=이름
label.dept=부서
label.age=나이
label.phone=전화
label.email=전자메일
label.address=주소

errors.required={0}은 필수 입력값입니다.
errors.minlength={0}은 {1}자 이상 입력해야 합니다.
errors.maxlength={0}은 {1}자 이상 입력할수 없습니다.
errors.invalid={0}은 유효하지 않은 값입니다.
errors.byte={0}은  byte타입이어야 합니다.
errors.short={0}은  short타입이어야 합니다.
errors.integer={0}은 integer 타입이어야 합니다.
errors.long={0}은 long 타입이어야 합니다.
errors.float={0}은 float 타입이어야 합니다.
errors.double={0}은 double 타입이어야 합니다.
errors.date={0}은 날짜 유형이 아닙니다.
errors.range={0}은 {1}과 {2} 사이의 값이어야 합니다.
errors.creditcard={0}은 유효하지 않은 신용카드 번호입니다.
errors.email={0}은 유효하지 않은 이메일 주소입니다.
errors.korean={0}은 한글을 입력하셔야 합니다.
errors.htmltag=태그는 사용하실수 없습니다.
```

[소스 13-21] message-common.xml, message-common_kr.xml 설정

영문설정을 위한 message-common_en.xml은 다음과 같다.

```
label.id=ID
label.name=NAME
label.dept=DEPARTMENT
label.age=AGE
label.phone=PHONE
label.email=EMAIL
```

```
label.address=ADDRESS

errors.required={0} is necessary.
errors.minlength={0} should be over {1} letters.
errors.maxlength={0} is not allowed over {1} letters.
errors.email={0} is not invalid email address.
errors.korean={0} should be korean.
errors.integer={0} should be integer.
```

[소스 13-22] message-common_en.xml 설정

4) Controller 설정

Server 및 client validation 활용을 위한 로직을 controller에 추가한다. 폼에서 submit된 정보를 Service 실행하기 전에 유효성 검증을 위해 beanValidator.validate를 실행 한다. 폼에서 client-side validation을 위해 validator.jsp가 실행 될 수 있도록, validator.do를 설정 한다.

```
@Controller
public class EmployeeServiceController {
        @Resource(name="employeeService")
        EmployeeService employeeservice;

        @Autowired
        private DefaultBeanValidator beanValidator;

        @RequestMapping(value = "/validator.do")
        protected String getValidator() throws Exception{
                return "validator";
        }

        @RequestMapping(value = "/employeeAdd.do", method=RequestMethod.GET)
        protected String employeeAddForm(ModelMap model) throws Exception{
                model.addAttribute("employee",new Employee());
                return "employee/employeeAddForm";
        }

  @RequestMapping(value = "/employeeAdd.do", method=RequestMethod.POST)
  protected String employeeAdd(@ModelAttribute("employee") Employee command,
                BindingResult errors, ModelMap model) throws Exception {
          beanValidator.validate(command, errors);

                if (errors.hasErrors()) {
                        return "employee/employeeAddForm";
                }
                employeeservice.insertEmployee(command);
                return "redirect:/employeeList.do";
        }
```

설정된 beanValidtor활용 위해 DI설정

Client-side validator 실행 위한 설정

Server-side validator 실행

[소스 13-23] Controller 설정

5) View(Form) 설정

View 출력하는 JSP에서 Client-side validation이 실행될 때 validator.do가 호출 될 수 있도록 validator.jsp를 추가한다.

```
<%@ page language="java" contentType="javascript/x-javascript" %>
<%@ taglib prefix="validator" uri="http://www.springmodules.org/tags/commons-validator" %>
<validator:javascript dynamicJavascript="false" staticJavascript="true"/>
```

[소스 13-24] validator.jsp 추가

Client validation을 위한 설정을 추가하고, 언어별로 페이지 출력을 위한 메시지와 validation 에러메시지가 출력될 수 있도록 JSP에 설정을 추가한다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="validator" uri="http://www.springmodules.org/tags/commons-validator" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER ADD</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="<c:url value="/validator.do"/>"></script>
<validator:javascript formName="employee" staticJavascript="false" xhtml="true"
 cdata="false"/>
<script type="text/javascript">
function save(form){
        if(!validateEmployee(form)){
                return;
        }else{
                form.submit();
        }
}
</script>
</head>
<body>
<form:form commandName="employee" action="employeeAdd.do">
<div id="wrap">
       <table class="tbl_View">
         <caption>Employee</caption>
         <colgroup>
                <col style="width:30%" />
                <col style="width: ;" />
         </colgroup>
         <tbody>
```

Validator.do 호출설정

Validator.xml 설정 맵핑

Validation JavaScript 호출하고
에러발생하면 입력폼을 다시 출력

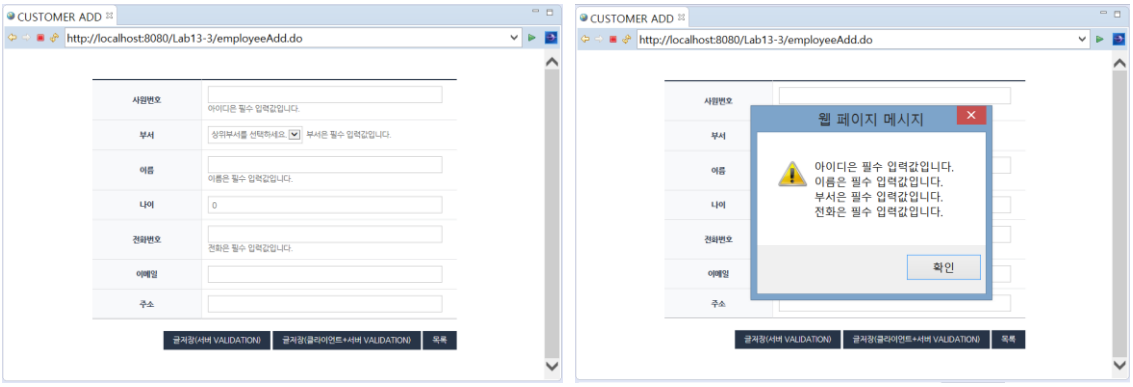
국제화를 위한 메시지 지정. 설정된
언어에 맞도록 메시지가 출력

```
        <tr>
                <th><spring:message code="label.id" /></th>
                <td><form:input path="id" type="text" size="63" maxlength="20" />
                <form:errors path="id" /></td>
        </tr>
        <tr>                                    Server-side validation 에러 메시지 출력
                <th><spring:message code="label.dept" /></th>
                <td>
                        <form:select path="dept">
                                <form:option selected="selected" value="">
                                상위부서를 선택하세요.</form:option>
                                <form:option value="01">기획팀</form:option>
                                <form:option value="02">영업팀</form:option>
                                <form:option value="03">개발팀</form:option>
                        </form:select>
                        <form:errors path="dept" />
                </td>
        </tr>
        <tr>
                <th><spring:message code="label.name" /></th>
                <td><form:input path="name" type="text" size="63" maxlength="20" />
                <form:errors path="name" /></td>
        </tr>
        <tr>
                <th><spring:message code="label.age" /></th>
                <td><form:input path="age" type="text" size="63" maxlength="20" />
                <form:errors path="age" /></td>
        </tr>
        <tr>
                <th><spring:message code="label.phone" /></th>
                <td><form:input path="phone" type="text" size="63" maxlength="20" />
                <form:errors path="phone" /></td>
        </tr>
        <tr>
                <th><spring:message code="label.email" /></th>
               <td><form:input path="email" type="email" size="63" maxlength="30" />
                <form:errors path="email" /></td>
        </tr>
        <tr>
                <th><spring:message code="label.address" /></th>
        <td><form:input path="address" type="text" size="63" maxlength="20" /></td>
        </tr>
        </tbody>
        </table>
        <div class="txt-rt mt20">
        <input type="button" value="글저장(서버 validation)" onclick="form.submit()"/>
        <input type="button" value="글저장(클라이언트+서버 validation)"
                  onclick="save(this.form)"/>
        <input type="button" value="목록" onclick="location.href='employeeList.do'"/>
    </div>
```

```
</div>
</form:form>
</body>
</html>
```

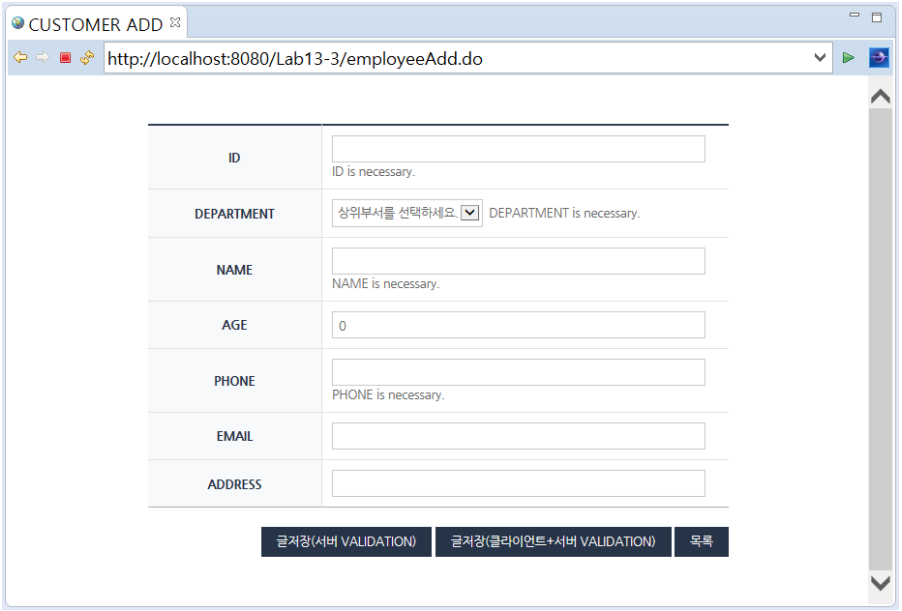
[소스 13-25] View 설정 (employeeAddFrom.jsp)

Validation이 설정되어 설정조건에 맞지 않은 경우 에러메시지를 출력하게 된다. Server-side, client-side validation 수행 결과 출력되는 에러메시지는 다음 그림과 같다.



[그림 13-6] Validation 결과 메시지

국제화가 적용되어 정보 출력 되는 라벨과 validation 결과 메시지가 다음 그림과 같이 영어로 출력된다.



[그림 13-7] 국제화 적용 결과

13.4 페이징과 검색기능 추가

앞에서 살펴본 표준프레임워크 예제 코드는 국제화와 validation 기능을 추가하였지만, 여

전히 부족한 점이 남아 있다. 첫번째는 리스트에서 내역이 많을 경우 적정한 수로 나누어 보여주고 이동도 가능한 페이징 기능과 제목이나 작성자 등 DB에 저장된 값을 검색하여 결과를 보여주는 부분이다. 두 가지 기능은 리스트를 처리하는 것과 관련성이 있다. 기능 추가를 위해 변경 되는 부분은 다음과 같다.

[표 13-4] Lab14-1 소스코드 구성

구분	코드명	설명
[src/main/java]		
Controller	lab.EmployeeServiceController.java	검색조건, 페이징 처리 위한 파라미터 추가
	EmployeeServiceImpl.java	리스트출력 파라미터 MAP으로 변경
DAO	EmployeeDAO.java	리스트출력 파라미터 MAP으로 변경
Model	lab.SearchCritetia.java	검색조건 VO추가
[src/main/resources]		
MyBatis	META-INF/sqlmap/mappers/ lab-dao-class.xml	검색과 페이징 반영 List SQL 수정, 페이지위한 Count SQL 추가
[src/main/webapp]		
View (JSP)	WEB-INF/jsp/employee/ employeeList.jsp	사원정보 목록 출력

페이징 처리와 검색을 위해서 List출력 SQL의 변경이 필요하고, 전체 출력 Count를 구하는 SQL이 추가로 필요하다. 검색 처리를 위해서는 검색 값에 따라서 SQL에 where절이 추가 되도록 Dynamic SQL을 활용하였다. 예제에서는 이름을 입력 받아 앞과 뒤로 같은 문자열이 있는 결과를 출력하도록 like절을 활용하였다. 데이터양이 아주 많을 경우 앞뒤로 like 검색을 하는 것은 인덱스를 활용하지 못하기 때문에 시스템에 무리가 될 수 있어 주의해야 한다. 만약 검색어가 없는 경우에는 where절 자체가 없이 실행이 된다. 파라미터 타입도 검색어와 페이징 조건이 함께 들어오기 때문에 Map을 활용하였다. 페이징 처리를 위해서 첫번째 ROW와 페이징처리 개수(예제에서는 3으로 입력)를 입력하면 DB에서 해당되는 만큼만 SELECT하여 결과를 보여준다. limit에서 첫번째 ROW는 0부터 시작하는 점에 주의해야 한다. limit 0,10 이라면 첫 번째 행부터 10개를 보여주게 된다.

```
<select id="selectEmployeeList" parameterType="java.util.Map"
        resultMap="employeeResult">
<![CDATA[
        select   id, name
        from     employee
]]>
        <where>
                <if test="searchName != null and searchName != ''">
                        name like '%${searchName}%'
                </if>
        </where>
<![CDATA[
```

파라미터를 MAP으로 변경

검색어가 입력되면 where절 추가

```
            order by CONVERT(id,SIGNED)
              limit #{firstIndex}, #{recordCountPerPage}
    ]]>
    </select>

    <select id="getEmployeeCount" parameterType="java.util.Map"
                resultType="int">
    <![CDATA[
            select count(*) id
            from employee
    ]]>
            <where>
                    <if test="searchName != null and searchName != ''">
                            name like '%${searchName}%'
                    </if>
            </where>
    </select>
```

페이징 처리를 위해 추가

페이징 처리를 위해 출력 되는 전체 ROW 수를 구하는 SQL 추가됨.

[소스 13-26] pom.xml에 MySQL JDBC드라이버 의존성 설정

DAO와 Service는 MAP 형태로 파라미터 형식을 변경하고, 전체 ROW 수를 구하는 메소드를 추가로 구현 한다.

```
    public List<Employee> selectEmployeeList(Map<?, ?> param) {
            return selectList("employee.selectEmployeeList", param);
    }

    public int getEmployeeCount(Map<?, ?> param) {
            return (Integer)selectOne("employee.getEmployeeCount", param);
    }
```

파라미터를 MAP으로 변경

메소드 추가

[소스 13-27] DAO 코드 변경

Service의 소스코드도 같은 형태로 파리미터를 MAP으로 변경하고, 메소드를 추가한다.

```
    public List<Employee> selectEmployeeList(Map<?, ?> param) throws Exception
{
            return employeeDAO.selectEmployeeList(param);
    }

    public int getEmployeeCount(Map<?, ?> param) {
            return employeeDAO.getEmployeeCount(param);
    }
```

[소스 13-28] Service 코드 변경

검색조건과 검색어 정보를 관리하는 VO를 추가한다. 본 예제에서는 이름으로만 검색하게 되어 있어 VO를 만드는 것이 귀찮을 수 있는데, 보통 검색 조건이 여러 개가 존재하는 경

우 VO로 만들어서 관리하는 것이 더 유용하다.

```
public class searchCriteria {
	public String searchName;

	public String getSearchName() {
		return searchName;
	}
	public void setSearchName(String searchName) {
		this.searchName = searchName;
	}
}
```

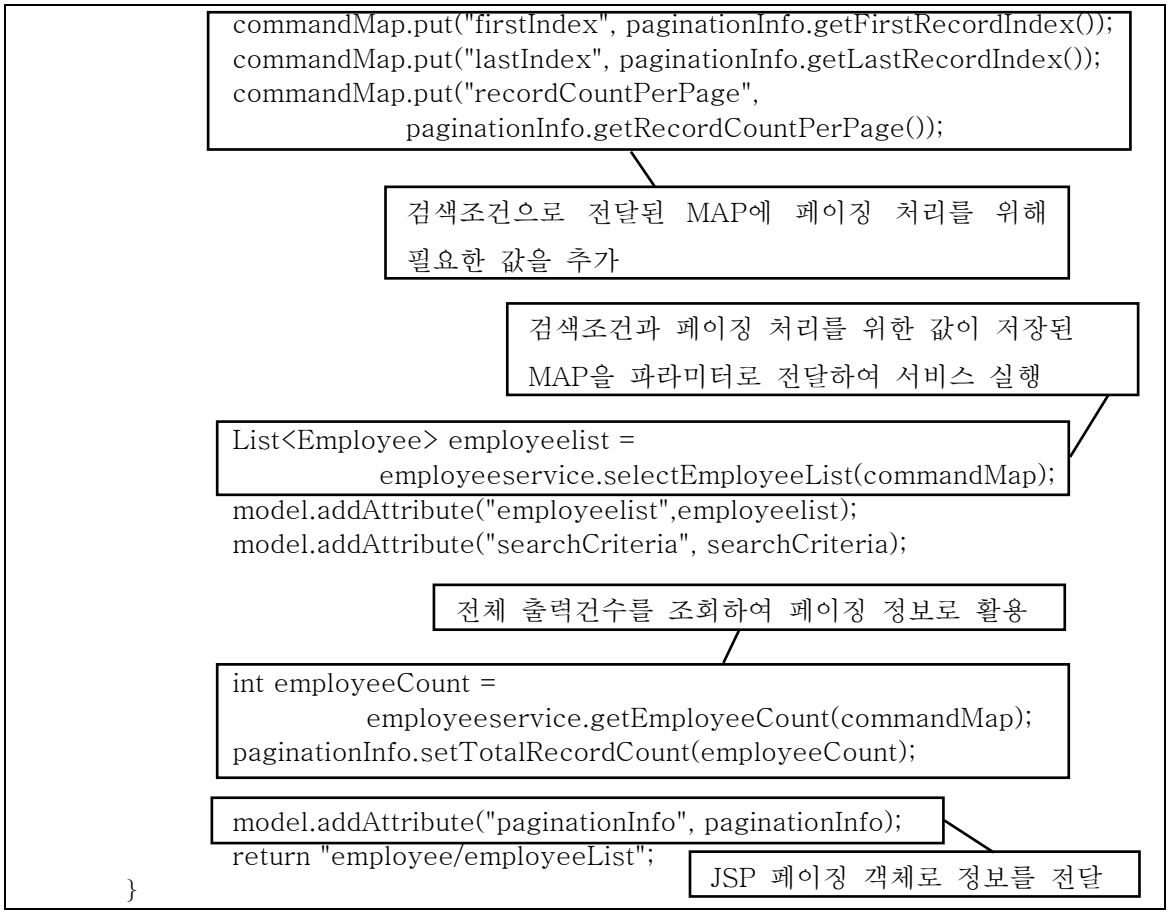
[소스 13-29] 검색조건 VO추가

페이징 처리와 검색기능 추가를 위해 가장 많이 변경이 되는 부분은 화면과 연관된 Controller와 View(JSP)이다. RequestMapping에서 현재 페이지의 번호를 파라미터로 전달 받는다. 검색조건에 관련된 부분은 @ModelAttribute를 활용한 부분과 @RequestParam으로 Map으로 동시에 전달 받는다. 결과적으로 searchCriteria VO와 commandMap Map에 같은 값을 받게 된다. @RequestParam Map<String, Object> commandMap 으로 파라미터를 받게 되면 파라미터가 모두 key와 value값으로 매핑이 되어 Map으로 저장된다. @ModelAttribute는 JSP에서 form 객체로 연결하기 위해 필요한 부분이다. 전달 받은 검색조건에 페이징 정보를 추가하여 서비스에 전달해야 되기 때문에 편의상 중복되지만 추가적으로 Map으로 파라미터를 전달 받았다. 전달 받은 검색조건에 페이징 처리를 위한 설정정보와 전체 출력건수를 View로 전달하여 페이징을 구성하게 된다.

```
@RequestMapping(value = "/employeeList.do")
protected String employList(@RequestParam(value = "pageNo", required = false)
 String pageNo, @ModelAttribute("searchCriteria") searchCriteria searchCriteria,
 ModelMap model, @RequestParam Map<String, Object> commandMap)
 throws Exception{

	int currentPageNo;
	try {
		currentPageNo = Integer.parseInt(pageNo);
	} catch (Exception e) {
		currentPageNo = 1;
	}

PaginationInfo paginationInfo = new PaginationInfo();
paginationInfo.setCurrentPageNo(currentPageNo);
paginationInfo.setRecordCountPerPage(3);
paginationInfo.setPageSize(10);
```

페이지번호, 검색조건 관련 파라미터 (MAP타입과 VO 타입 두가지 모두 활용됨)

표준프레임워크의 페이징 기능 활용을 위해 객체생성과 변수설정 (페이지당 3개, 페이지건수 10개씩 디스플레이)



[소스 13-30] Controller 코드 변경

리스트를 출력하는 JSP에는 검색을 위한 입력창과 버튼이 추가가 되고, 리스트 하단에 페이지 리스트가 생성되게 된다. 전체 출력 데이터가 50개이고, 한 페이지에 10개씩 데이터가 출력되면 페이지는 1-5까지 5개가 보여지게 된다. 이러한 처리를 직접 하게 되면 매우 복잡하다. 표준프레임워크에서 제공하는 pagination기능을 활용하면 태그라이브러리 형태로 제공이 되어 다음과 같이<ui:pagination paginationInfo = "${paginationInfo}" type="image" jsFunction="linkPage" /> 호출하여 간단하게 다음과 같이 페이징 구현이 가능하다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="ui" uri="http://egovframework.gov/ctl/ui"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CUSTOMER LIST</title>
<link href="css/style.css" rel="stylesheet" type="text/css">
```

```
</head>
<script type="text/javascript">
 function linkPage(pageNo){
          location.href = "employeeList.do?pageNo="+ pageNo;
  }
</script>

<body>
<form:form commandName="searchCriteria" action="employeeList.do">
<div id="wrap">
          <div class="search">
                    이름 <input id="searchName" name="searchName"
                style="width:120px;" type="text" value=""/>
                    <input type="submit" value="검색"
                     onclick="this.disabled=true,this.form.submit();" />
          </div>
   <table class="tbl_List">
          <caption>Board</caption>
          <colgroup>
                    <col style="width:9%" />
                    <col style="width:16%" />
                    <col style="width:16%" />
                    <col style="width:13%" />
                    <col style="width:11%" />
                    <col style="width: ;" />
          </colgroup>
          <thead>
                    <tr>
                              <th>번호</th>
                              <th>이름</th>
                    </tr>
          </thead>
          <tbody>
          <c:forEach items="${employeelist}" var="employeeinfo">
                <tr>
                    <td><a
href="employeeView.do?id=${employeeinfo.id}">${employeeinfo.id}</a></td>
                    <td>${employeeinfo.name}</td>
                </tr>
        </c:forEach>
        </tbody>
    </table>
          <div id="pagination">
          <ui:pagination paginationInfo = "${paginationInfo}"
                    type="image" jsFunction="linkPage" />
          </div>
    <p>
          <div class="txt-rt mt20">
          <input type="button" value="사원추가"
 onclick="location.href='employeeAdd.do'" />
```

페이지 리스트에 링크가 걸려서 페이지가 눌렸을 때 실행

검색값 입력 창과 실행버튼 생성

Controller로부터 전달받은 페이징 정보를 활용하여 페이지 리스트를 생성하는 태그라이브러리

```
        </div>
    </div>
    </form:form>
    </body>
    </html>
```

[소스 13-31] JSP변경

페이징 조건은 한 페이지에 3건, 페이지리스트 10개로 되어 있어 아래와 같이 3개만 표시 되고 4번째 데이터는 다음 페이지에 디스플레이 된다.

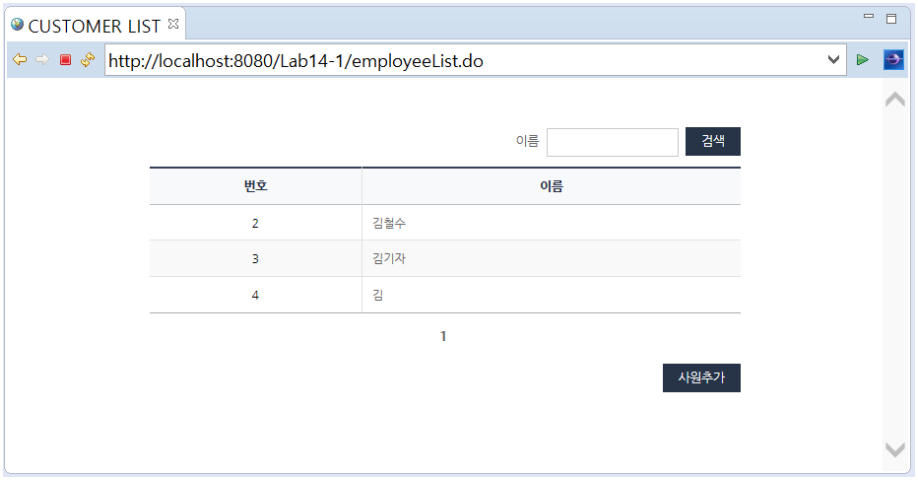
[그림 13-8] 페이징 처리결과

전체 페이징은 다음과 같이 페이지 리스트에서 페이지별 이동과 맨 처음, 마지막으로 이동할 수 있도록 링크가 제공되며, 이미지와 테스트 방식에 따라서 다르게 보여진다.

[표 13-5] 페이지리스트 구성

<ui:pagination paginationInfo = "${paginationInfo}" type="image" jsFunction="linkPage"/>	<ui:pagination paginationInfo = "${paginationInfo}" type="text" jsFunction="linkPage"/>
1 2 3 4 5 6 7 8	[처음] [이전] 1 2 3 4 5 6 7 8 [다음] [마지막]

검색을 수행하여 데이터 중에 “김”자가 들어가는 이름으로 출력을 하면 다음과 같이 검색결과가 적용되어 출력된다.


CUSTOMER LIST
http://localhost:8080/Lab14-1/employeeList.do
이름
검색
번호
이름
2
김철수
3
김기자
4
김
1
사원추가

[그림 13-9] 검색 결과

전자정부 표준프레임워크 퍼스트북

발행 | 2018년 3월1일
저자 | 임철홍

펴낸이 | 임철홍
펴낸곳 | (주)플랫폼기술연구소
출판등록 | 2018.01.15.(제2018-11호)
주소 | 서울시 종로구 종로 19, 르메이에르종로타운 729호
전화 |(02)723-0850
이메일 | imich@platformtech.co.kr

ISBN | 979-11-962942-1-2
www.platformtech.co.kr

국립중앙도서관 출판예정도서목록(CIP)

전자정부 표준프레임워크 퍼스트북 [전자자료] = eGovFrame first book / 저자: 임철홍. -- 서울 : 플랫폼기술연구소, 2018
전자책 책

ISBN 979-11-962942-1-2 95000 : 비매품

전자 정부[電子政府]

350.5-KDC6
352.380285-DDC23 CIP2018005088