Developing web application

with APPLICATION ARCHITECTURE for JAVA

목차

[I. 소개 6](#_Toc333925150)

[1.1 소개 7](#_Toc333925151)

[1.1.1 FUSE™ ARCHITECTURE 2.0 플랫폼 이란? 7](#_Toc333925152)

[1.1.2 레이어드 아키텍처 스타일 7](#_Toc333925153)

[1.1.3 이중화된 컨테이너 구조 9](#_Toc333925154)

[1.1.4 플러그인 아키텍처 11](#_Toc333925155)

[1.1.5 동적 아키텍처 12](#_Toc333925156)

[1.1.6 설정의 재사용 13](#_Toc333925157)

[1.1.7 공개소스 활용 15](#_Toc333925158)

[1.2 적용 시나리오 16](#_Toc333925159)

[1.2.1 AJAX 와 RESTful 기반의 웹 어플리케이션 16](#_Toc333925161)

[1.2.2 모바일 웹 어플리케이션 16](#_Toc333925162)

[1.2.3 웹 서비스 기반의 미들웨어 17](#_Toc333925163)

[II. 환경 19](#_Toc333925164)

[2.1 개발 환경 설정 20](#_Toc333925166)

[2.1.1 개발 도구 20](#_Toc333925169)

[2.1.2 형상관리 23](#_Toc333925170)

[2.1.3 빌드도구 23](#_Toc333925171)

[2.1.4 디폴트 예제 프로젝트 생성 24](#_Toc333925172)

[2.2 웹 애플리케이션 환경 설정 24](#_Toc333925173)

[2.2.1 홈 경로 설정 24](#_Toc333925175)

[2.2.2 인코딩 설정 25](#_Toc333925176)

[2.2.3 지원 하드웨어 / 소프트웨어 환경 26](#_Toc333925177)

[2.3 클러스터 설정 27](#_Toc333925178)

[2.4 서버 튜닝 27](#_Toc333925179)

[2.5 싱글사인온(Single Sign On) 설정 27](#_Toc333925180)

[III. 코어 컴포턴트 28](#_Toc333925181)

[3.1 로깅 29](#_Toc333925182)

[3.2 국제화 및 지역화 29](#_Toc333925185)

[IV. 프레젠테이션 계층 30](#_Toc333925186)

[4.1 Struts 2 기반 MVC 프레임워크 사용하기 31](#_Toc333925188)

[4.1.1 Struts 2 소개 31](#_Toc333925191)

[4.1.2 Struts 2 아키텍처 33](#_Toc333925192)

[4.1.3 Struts 2 환경설정 34](#_Toc333925193)

[4.2 Struts 기반 MVC 프레임워크 사용하기 34](#_Toc333925194)

[4.2.1 MVC 디자인 모델 34](#_Toc333925196)

[4.2.2 스트럿츠 소개 35](#_Toc333925197)

[4.2.3 Action 지원을 위한 클래스 36](#_Toc333925198)

[4.2.4 JSP를 이용한 뷰 구현 38](#_Toc333925199)

[4.3 Sitemesh 기반 UI 구현 38](#_Toc333925200)

[4.3.1 Sitemesh 소개 38](#_Toc333925202)

[4.3.2 Sitemesh 설치 39](#_Toc333925203)

[4.4 DWR을 이용한 동적 웹 구현 40](#_Toc333925204)

[4.4.1 DWR 소개 40](#_Toc333925206)

[4.4.2 DWR 설치 41](#_Toc333925207)

[V. 서비스 & 비즈니스 계층 43](#_Toc333925208)

[5.1 비즈니스 서비스 44](#_Toc333925210)

[5.1.1 서비스 구현 44](#_Toc333925213)

[5.1.2 서비스에서 로깅 44](#_Toc333925214)

[5.1.3 서비스에서 에러 처리 45](#_Toc333925215)

[5.1.4 서비스에서 데이터베이스 접근 48](#_Toc333925216)

[VI. 데이터 계층 50](#_Toc333925217)

[6.1 Spring 프레임워크 기반의 트랜잭션 관리 51](#_Toc333925219)

[6.1.1 선언형 트랜잭션 관리 (Declarative transaction Management) 51](#_Toc333925222)

[6.1.2 @Tranasctional 어노테이션을 사용하기 51](#_Toc333925223)

[6.1.3 프로그래밍을 통한 트랜잭션 관리 54](#_Toc333925224)

[6.2 SqlQuery 54](#_Toc333925225)

[6.2.1 SqlQuery 소개 54](#_Toc333925227)

[6.2.2 쿼리 XML 55](#_Toc333925228)

[6.2.3 SELECT, UPDATE 56](#_Toc333925229)

[6.2.4 배치 업데이트 57](#_Toc333925230)

[6.2.5 LOB 데이터 57](#_Toc333925231)

[6.2.6 페이징 처리 59](#_Toc333925232)

[6.2.7 동적 쿼리 59](#_Toc333925233)

[6.3 SqlQueryClient 60](#_Toc333925234)

[6.3.1 테이블 데이터 엑셀 Export 60](#_Toc333925236)

[6.3.2 테이블 데이터 엑셀 Import 61](#_Toc333925237)

[6.4 데이터베이스 연결 61](#_Toc333925238)

[6.4.1 DataSource 61](#_Toc333925240)

[6.4.2 커넥션 풀링 기반의 DataSource 64](#_Toc333925241)

[6.4.3 JDBC 드라이버 기반의 DataSoruce 65](#_Toc333925242)

[VII. 통합 계층 66](#_Toc333925244)

[7.1 웹 서비스 67](#_Toc333925245)

[7.2 메일 서비스 67](#_Toc333925246)

[7.3 메시지 서비스 67](#_Toc333925247)

[VIII. 보안 68](#_Toc333925248)

[참고 문헌 70](#_Toc333925249)

# 소개

1990년대 후반부터 일반 소프트웨어 산업의 경우와 마찬가지로 웹을 기반으로 하는 애플리케이션 개발에서 소프트웨어 재사용을 통한 생산성과 유지보수성이 아주 중요한 요인으로 인식되면서, 대안으로 등장했던 웹 프레임워크 기술은 성숙단계에 진입하여 웹 애플리케이션의 개발 생산성 향상에 큰 역할을 하고 있다. 그러나 현재 자바 영역에서 웹 애플리케이션 개발은 필요 이상으로 복잡하다. 웹 애플리케이션 개발에 하나 이상의 프레임워크를 사용하는 것이 일반적이며, 이들 프레임워크들 역시 다양한 요구사항을 만족 시키기 위하여 복잡성이 증대되고 있다.

최소의 설정만으로 웹 애플리케이션 구현이 가능하도록 지원하는 Rails 같은 동적 프레임워크들의 아이디어는 웹 애플리케이션 개발을 좀더 현재적으로 생각해 볼 수 있게 한다. FUSE™ ARCHITECTURE 2.0은 웹을 기반으로 하는 애플리케이션 개발 및 유지보수를 위한 기반 기술을 제공하는 플랫폼으로 DRY(don’t repeat yourself) 원칙과 동적 프레임워크들의 생산성증대 아이디어를 도입하여 웹 애플리케이션 개발의 복잡성을 극적으로 줄이도록 설계되었다.

## 소개

### FUSE™ ARCHITECTURE 2.0 플랫폼 이란?

### 레이어드 아키텍처 스타일

복잡한 소트웨어를 디자인할 때 공통적인 아키텍처 원칙을 적용하는 것이 유용하다는 것은 이미 널리 알려진 사실이다. FUSE™ ARCHITECTURE 2.0 플랫폼은 많은 웹 응용프로그램들을 통하여 유용함이 증명된 레이어드 아키텍처 스타일을 따른다. 다음 그림은 이해를 돕기 위하여 업무측면과 개발 측면에서 바라본 레이어드 아키텍처 스타일을 보여주고 있다.



그림 I‑1 업무 및 개발 관점 레이어드 아키텍처 스타일

FUSE™ ARCHITECTURE 2.0 플랫폼은 업무관점에서 비즈니스 프레임워크 레이어에 위치하며 업무 레이어에서 공통 및 단위 업무 개발을 돕는 역할을 한다. 아래 그림은 개발 관점에서 레이어드 아키텍처 스타일의 이해를 돕기 위하여 도식화한 것이다.



그림 I‑2 레이어드 아키텍처 스타일

### 이중화된 컨테이너 구조

FUSE™ ARCHITECTURE 2.0 플랫폼은 Spring[[1]](#footnote-1)의 경량 컨테이너 기술(IoC[[2]](#footnote-2))을 사용하여 이중화된 컨텍스트 영역으로 구성된다. 컨텍스트는 등록된 모듈(또는 객체)들 설정과 생명주기를 관리하는 실재적 컨테이너로 동작한다.

**

부모에 해당하는Bootstrap Application Context 에는 코어 모듈(또는 객체)들이 존재하며 자식에 해당하는 Web Application Context 에는 업무 모듈(또는 객체)들이 존재한다. 자식 컨텍스트는 부모 컨텍스트에 정의된 모듈들에 접근이 허용되지만 부모는 자식 컨텍스트에 접근이 허용되지 않는다. 이러한 목적에 따른 컨텍스트의 분리는 비즈니스 모듈 개발자들의 설정을 간소화하고 주요한 코어 모듈들에 대한 설정을 임으로 변경하는 것을 보호하는 효과를 제공한다.



### 플러그인 아키텍처

플러그인 기술은 애플리케이션이 동작 중에 모듈(또는 객체)들 또는 애플리케이션의 컴파일 동안 알지 못했던 클래스들을 동적으로 로드하여 확장할 수 있도록 하는 기술이다. FUSE™ ARCHITECTURE 2.0 플랫폼은 플러그인 기술 사용하여 응용프로그램의 확장성을 최대화한다.



### 동적 아키텍처

FUSE™ ARCHITECTURE 2.0 은 Groovy 기술을 사용하여 비즈니스 코드를 수정하더라고 소스 컴파일과 서버의 재 시작 없이 적용될 수 있도록 지원한다.



### 설정의 재사용

FUSE™ ARCHITECTURE 2.0 은 Rails 의 아이디어를 도입하여 다양한 환경에서 개발 및 운용을 위한 설정파일들을 미리 가지고 있으며 최소한의 설정 만으로 웹 어플리케이션을 개발 할 수 있도록 돕는다. 가장 대표적 설정 파일이 startup-config.xml 파일이며 이 파일을 통하여 데이터베이스 및 보안 등 다양한 기능들을 사용할 수 있다. 이러한 간결함을 통하여 생산성 향상 효과를 기대할 수 있다.

startup-config.xml 설정에 따라 코어 모듈 컨테이너 을 포함하는 spring xml 파일을 기반으로 bootstrap application context 생성한다. 또한 startup-config.xml 에 정의된 값에 따라 코어 모듈을 생성한다.



다음은 startup-config.xml 을 사용하여 데이터베이스를 설정하는 예시이다.

|  |
| --- |
| startup-config.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<startup-config>

<setup>

<complete>false</complete>

</setup>

<services>

<sql>

<location>/WEB-INF/sql</location>

</sql>

</services>

<!-- JDBC DATASOURCE SETTING -->

<database>

<default>

<!-- 1. Use jndi datasource. -->

<jndiDataSourceProvider order=*"1"*>

<jndiName></jndiName>

</jndiDataSourceProvider>

<!-- 2. Use the built-in pooled datasource. -->

<pooledDataSourceProvider order=*"2"*>

<driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>

<url>jdbc:oracle:thin:@//222.122.63.138:1521/INKIUMDB</url>

<username>ICAP\_HYUNDAICAPITAL</username>

<password>inkium</password>

<maxIdle>10</maxIdle>

<maxActive>100</maxActive>

<maxWait>10000</maxWait>

<validationQuery>SELECT 1 FROM DUAL</validationQuery>

<testOnBorrow>false</testOnBorrow>

<testWhileIdle>true</testWhileIdle>

<timeBetweenEvictionRunsMillis>1200000

</timeBetweenEvictionRunsMillis>

<minEvictableIdleTimeMillis>1800000</minEvictableIdleTimeMillis>

<numTestsPerEvictionRun>5</numTestsPerEvictionRun>

</pooledDataSourceProvider>

</default>

</database>

</startup-config>

### 공개소스 활용

FUSE™ ARCHITECTURE 2.0 는 다양한 기능들을 검증된 공개소스를 사용하여 구현하고 있다. 예를 들어 웹 MVC 프레임워크로는 Struts, JDBC 프로그래밍은 Spring, 스케줄링은 Quartz 등을 사용하고 있다.

## 적용 시나리오



### AJAX 와 RESTful 기반의 웹 어플리케이션



### 모바일 웹 어플리케이션



### 웹 서비스 기반의 미들웨어



# 환경



## 개발 환경 설정

FUSE™ ARCHITECTURE 2.0 플랫폼 기반의 응용 프로그램을 개발을 위하여 기본 개발 도구들을 설치한다. 주의할 것은 자바는 반듯이 1.5 이상을 사용하여야 한다.

|  |  |  |
| --- | --- | --- |
| **구분** | **항목** | **URL** |
| 언어 | JAVA | <http://www.oracle.com/technetwork/java/javase/downloads/index.html> |
| 개발도구 | eclipse | <http://www.eclipse.org/> |
| 빌드관리 | maven | <http://maven.apache.org> |
| 형상관리 | subversion | <http://subversion.tigris.org/> |



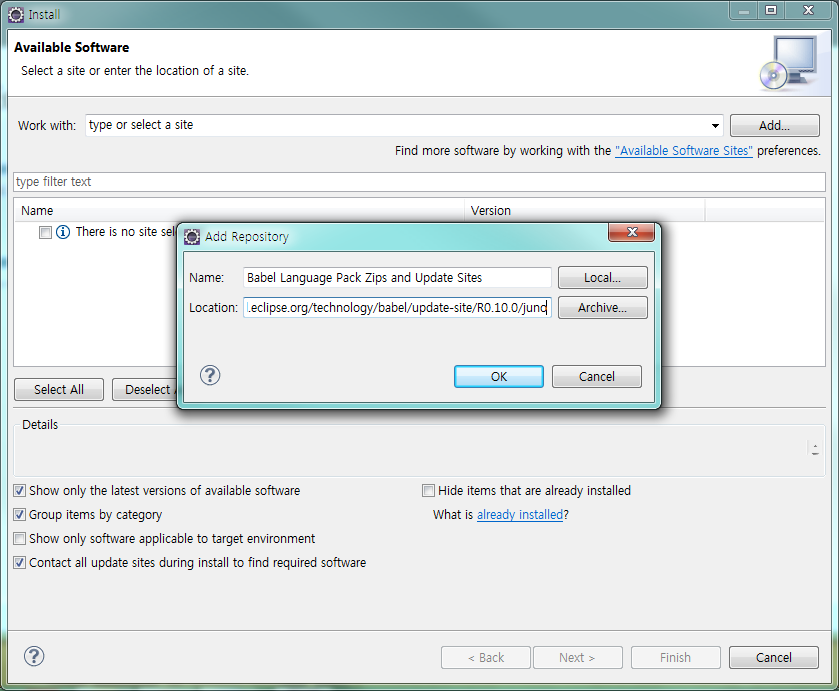
### 개발 도구

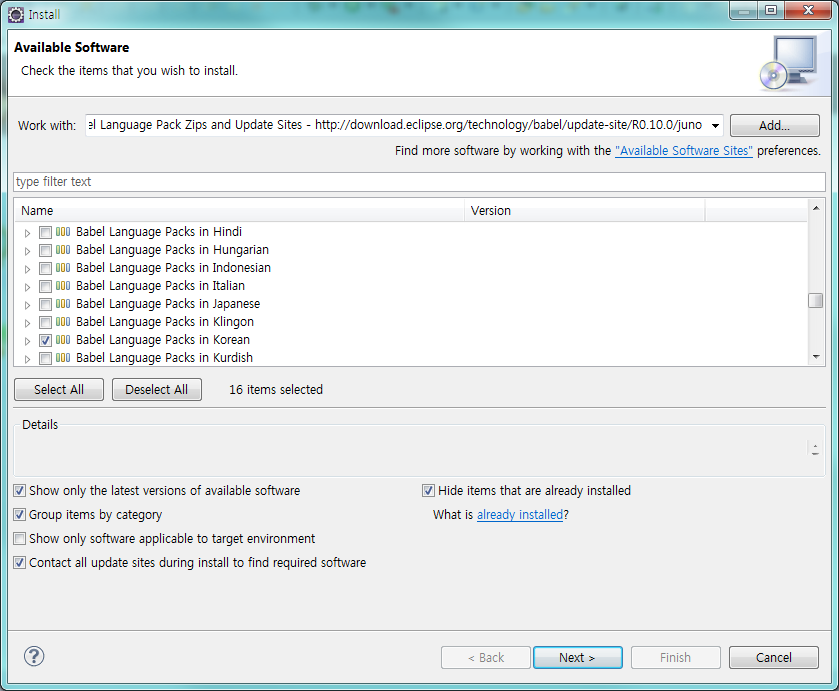
기본적인 개발 도구로 공개 소프트웨어인 Eclipse 를 지원하고 있다. [www.eclipse.org](http://www.eclipse.org) 사이트를 방문하여 개발자 환경에 맞는 Eclipse버전을 선택하여 다운로드하고 설치한다. 여기에서는 Eclipse IDE for Java EE Developers (Juno) 를 설치하였다.



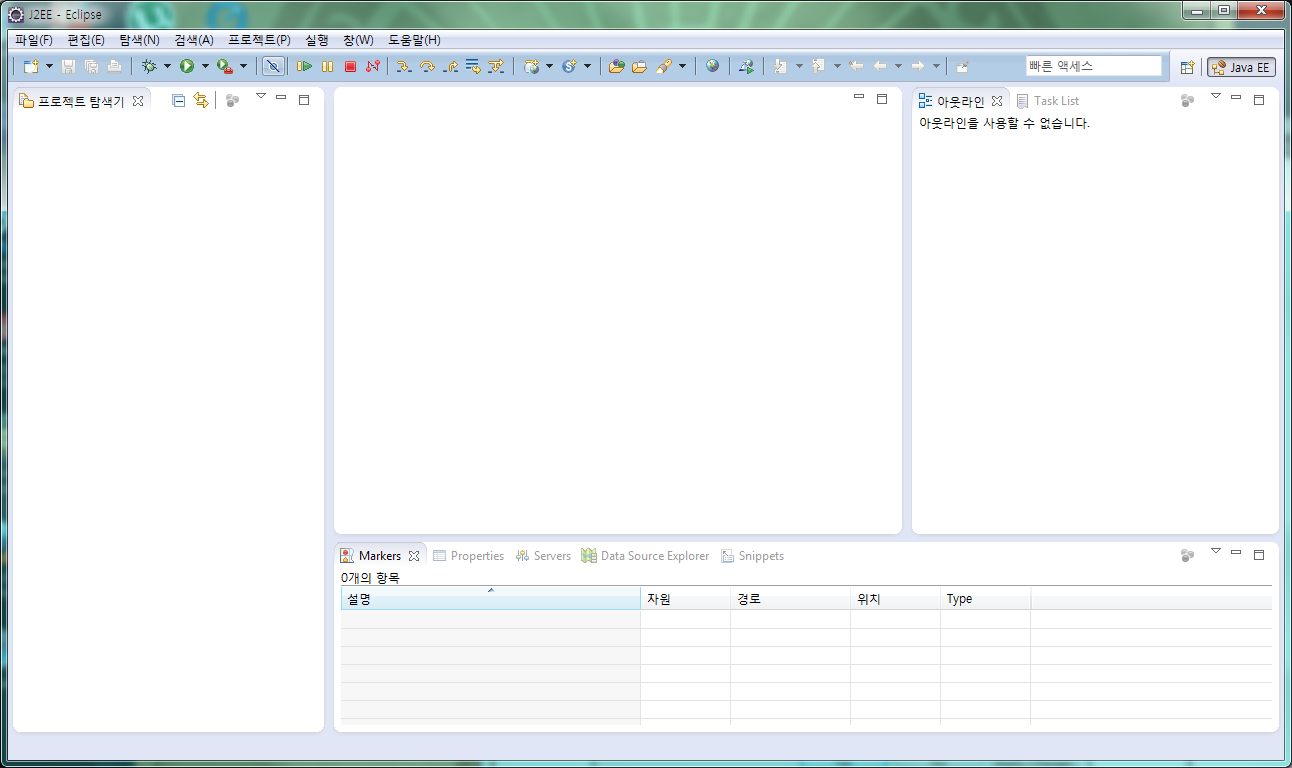
기본적으로 Eclipse는 영어를 지원하며 다국어 지원을 위해서는 [Eclipse Babel Project](http://www.eclipse.org/babel/) 에서 제공하는 언어 팩을 설치하면 된다. 한글 지원을 위한 언어 팩 설치를 위해서 Eclipse 메뉴의 Help > Install New Software …… 를 선택한다. Add 버튼을 클릭하고 언어 팩 설치를 위한 새로운 업데이트 사이트를 추가한다. 업데이트 사이트는 Juno 경우 아래와 같다. 만일 다른 버전의 Eclipse 를 사용하고 있다면 <http://www.eclipse.org/babel/> 사이트를 방문하여 해당하는 Update 사이트를 사용한다.

**Babel Language Pack Update Site for Juno**  
http://download.eclipse.org/technology/babel/update-site/R0.10.0/juno





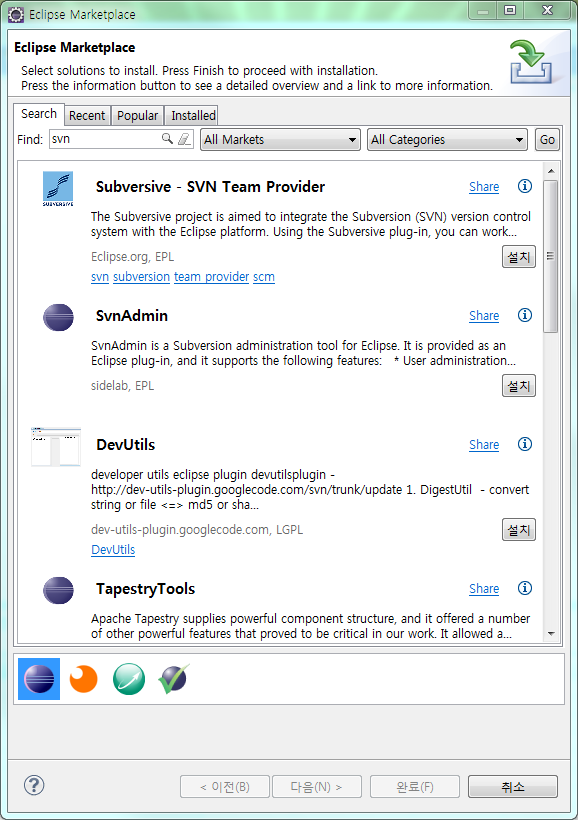
이제 Bable Language Packs in Korean 을 선택하고 설치를 진행한다. 설치가 완료되고 다시 Eclipse 를 실행하면 다음과 같이 한글화를 지원하게 된다.



이제 기본적인 Eclipse 설치가 완료되었다. FUSE™ ARCHITECTURE 2.0 플랫폼 기반의 웹 애플리케이션 개발을 위해서는 형상관리 및 빌드 도구와 같은 추가적인 기능이 필요하다.

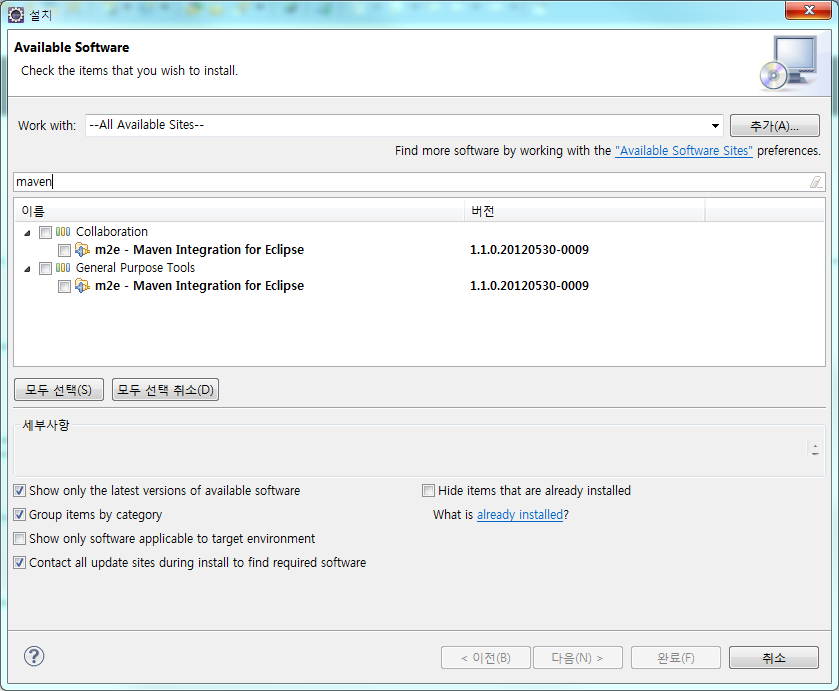
### 형상관리

기본적인 형상관리 도구로 공개 소프트웨어인 subversion 를 지원하고 있다. Eclipse 에서 subversion 사용을 위하여 subversive 를 설치하여 사용한다. 먼저 메뉴에서 도움말(H) > Eclipse Market Place … 를 클릭한다. 검색어로 svn 을 검색하고 결과화면에서 Subversive – SVN Team Provider 를 설치한다.



### 빌드도구

기본적인 빌드도구로 공개 소프트웨어인 maven 를 지원하고 있다. Eclipse 에서 maven 사용을 위하여 m2e – Maven Integration for Eclipse 를 설치하여 사용한다. 먼저 메뉴에서 도움말(H) > Install New Software … 를 클릭한다. 검색어로 maven 을 검색하고 결과화면에서 m2e – Maven Integration for Eclipse 를 설치한다.



### 디폴트 예제 프로젝트 생성

제공되는 디폴트 예제 프로젝트를 다운로드 하여 Eclipse 프로젝트로 Import 한다.

## 웹 애플리케이션 환경 설정



### 홈 경로 설정

FUSE™ ARCHITECTURE 2.0 플랫폼 기반의 웹 어플리케이션은 동작을 위하여 필요한 환경 설정 정보 및 데이터들을 저장하게 될 응용프로그램 홈 경로를 설정해야 한다. 명시적으로 홈 경로를 설정하지 않는 경우 배포된 웹 응용프로그램의 /WEB-INF 경로를 홈으로 사용하게 된다.

**방법1**

WEB-INF/classes 경로에 아래와 같이 XML 파일(applicatioin-init.xml)을 생성하여 어플리케이션 동작을 위하여 필요한 환경 설정 정보 및 데이터들을 저장하게 될 경로를 설정 한다. 디폴트 경로를 사용하여 오류가 나는 경우에 사용한다.

|  |
| --- |
| /WEB-INF/classes/application-init.xml |

/WEB-INF/classes/application-init.xml

<?xml version="1.0" encoding="UTF-8"?>

<application>

<home>file://C:\fuse\profile\default</home>

</application>

**방법2**

WEB-INF/web.xml 파일을 수정하여 “ARCHITECTURE\_RUNTIME\_ROOT” 이름으로 context-param 값에 경로를 지정한다.

|  |
| --- |
| /WEB-INF/web.xml |

/WEB

…

<context-param>

<param-name>ARCHITECTURE\_PROFILE\_ROOT</param-name>

<param-value>file://C:\fuse\profile\default</param-value>

</context-param>

</web-app>

### 인코딩 설정

웹 애플리케이션에서 다국어 처리를 위하여 요청된request 객체에 대한 인코딩 처리가 필요하다. (브라우저에서 입력된 파라메터가 자바 프로그램에서 깨져서 처리되는 경우에 적용)

|  |
| --- |
| /WEB-INF/web.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app id=*"WebApp\_ID"*>

<filter>

<filter-name>encoding</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-name>

<url-pattern>\*.do</url-pattern>

</filter-mapping>

</web-app>

**가. JEUS**

TmaxSoft 사의 JEUS를 사용하는 경우, CharacterEncodingFilter 필터를 사용하지않고 컨테이너가 제공하는 기능을 사용하여 처리할 수 있다. (CharacterEncodingFilter 필터를 사용하지 않아도 된다.) ${JEUS\_HOME}/config/[호스트이름]/[컨테이너]/WEBMain.xml 파일에 encoding 설정을 추가한다.

|  |
| --- |
| WEBMain.xml |

<?xml version="1.0" encoding="UTF-8"?>

<web-container xmlns="http://www.tmaxsoft.com/xml/ns/jeus">

<context-group>

<group-name>ICAP\_HYUNDAICAPITAL</group-name>

<webserver-connection>

</webserver-connection>

<encoding>

<request-encoding>

<default>utf-8</default>

</request-encoding>

<response-encoding>

<default>utf-8</default>

</response-encoding>

<postdata-encoding>

<default>utf-8</default>

</postdata-encoding>

</encoding>

</context-group>

</web-container>

### 지원 하드웨어 / 소프트웨어 환경

테스트된 프레임워크 기반의 응용프로그램 운영환경은 아래와 같다. 기술되지 않은 운영시스템 또는 검증 여부에 체크되지 않는 경우지 설치가 불가능할 수 도 있다. 일반적으로 JDK 1.5 이상을 지원하는 운영시스템에서는 설치가 가능하다.

|  |  |  |
| --- | --- | --- |
| H/W | IBM p780 | IBM p740 |
| OS | IBM AIX 5.3 | AIX 6.2 |
| WEB |  | Apache 2.2.22 |
| WAS | Oracle Weblogic 10.3 | Tomcat 7.0.29 |
| JAVA | IBM Java 1.6 | IBM Java 1.6.0 (64-bit) |
| JDBC |  | OJDBC 11.2.0.3.0 |
| DBMS | Oracle 10g | Oracle 11g (11.2.0.3.0) |
| 인증 | △ | ○ |
| 비고 | 클래스로더가 create와 같은 특정 이름의 함수를 검색하지 못함.  리턴값이 숫자가 아닌 프로시저 호출에 오류가 발생. |  |

## 클러스터 설정

## 서버 튜닝

## 싱글사인온(Single Sign On) 설정

# 코어 컴포턴트

## 

## 로깅



FUSE™ ARCHITECTURE 2.0 플랫폼을 구현하는 클래스들은 모두 [Apache Commons Logging API](http://commons.apache.org/logging) (이한 JCL)를 기반으로 하여 로깅을 처리하고 있다. JCL 의 좋은 점은 로깅을 구현하는 라이브러리를 변경하더라도 코드에 변경을 요구하지 않는다는 것이다.

## 국제화 및 지역화

Spring 의 다국어 메시지 지원을 위하여 제공되는 MessageSource 인터페이스를 확장하는 I18nTextResourceBundleMessageSource 클래스가 제공되며 이를 통하여 Spring 에서 제시하는 방법인 MessageSourceAware 인터페이스를 비즈니스 클래스에서 구현하여 사용하거나 WebApplicationHelper.getMessage() 함수를 사용하여 국제화를 처리할 수 있다. Spring 설정을 내부적으로 처리하기 때문에 사용을 위한 특별한 작업이 요구되지 않는다.

WebApplicationHelper.*getMessage*(“main.page.title”, **new** Object[]{}, **null**);

Localizer

I18nText

ObjectType, ObjectId, ObjectAttribute, Text, LocaleId

# 프레젠테이션 계층





## Struts 기반 MVC 프레임워크 사용하기



### MVC 디자인 모델

MVC 디자인 모델은 UI 가 포함된 프레임워크에는 거의 필수적으로 사용되는 패턴으로 원래는 일반 GUI 애플리케이션을 개발할 때 UI 와 비즈니스 로직을 효과적으로 분리하기 위하여 고안되었으나 웹 애플리케이션에서 역시 그 효과가 입증되어 널리 쓰이고 있다.

MVC의 구조는 MFC 프로그램에서 등장했던 Document-View 구조를 한층 더 발전시킨 것으로 프로그램의 구성요소를 모델, 뷰, 컨트롤러로 나누어서 각각 다른 역할을 처리하도록 하는 것이다.





### 스트럿츠 아키텍처

스트럿트는 아파치 그룹에 지원 받은 오픈소스 프로젝트로 Craig McClanahan에 의해 2000년 5월 개발이 시작되었고 얼마 후 오픈 소스 커뮤니티로 이전되어 많은 사람들에 의해 공동 개발된 MVC Model2[[3]](#footnote-3) 디자인에 기반한 웹 어플리케이션 프레임워크이다.

스트럿트는 Controller 계층을 ActionServlet 형태로 구현하고 JSP 태그 라이브러리를 이용하여 View 계층을 구현할 것을 권장하고 있다. 또한 Action 클래스를 통하여 Model 계층에 대한 래퍼를 제공하고 있다. 다음의 그림은 스트럿트의 구조와 사용자의 요청을 처리하는 단계를 보여주고 있다.



1. View1(JSP, HTML, WML)으로부터 사용자 요청이 발생한다.
2. ActionServlet이 사용자 요청을 받는다. ActionServlet은 Struts에서 Controller의 기능을 수행한다. 사용자에 의해 요청된 URI와 Configuration XML을 기초로 사용자 요구를 만족시키기 위해 필요한 비즈니스 로직(Model)을 호출하는 Action을 호출한다.
3. Action 클래스는 사용자 요구에 맞는 Model 컴포넌트에 대한 비즈니스 로직을 수행시킨다.
4. Action Class가 비즈니스 로직 수행을 끝내면 ActionServlet이 다시 제어권을 넘겨받는다. Action Class의 수행을 통해 발생한 결과 값을 가리키는 key 값을 ActionServlet이 넘겨받게 되고 ActionServlet은 이 값을 기초로 사용자에게 결과값을 전송한다.
5. ActionServlet이 Action 클래스로부터 넘겨받은 key값을 기초로 View를 생성하여 사용자에게 보여줌으로써 Struts 프로세스 사이클이 끝이 난다.

### Action 지원을 위한 클래스

FUSE™ ARCHITECTURE 2.0 의 Spring 기반 컨테이너에 대한 접근을 위하여 프레임워크가 제공하는 Action 클래스를 확장하는 방법이 제공된다.

* FrameworkActionSupport
* FrameworkDispatchActionSupport



이 클래스들은 컨테이너에 존재하는 클래스 타입을 인자로 해당하는 객체를 getComponent 함수와 객체를 컴포넌트에 존재하는 객체들을 사용하여 자동으로 인자로 전달된 객체의 프로퍼티를 설정하는 autowireComponent 함수를 제공한다.

|  |
| --- |
| QueryActioin.java |

/WEB

**package** tests.action;

**import** java.util.List;

**import** java.util.Map;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.struts.action.ActionForm;

**import** org.apache.struts.action.ActionForward;

**import** org.apache.struts.action.ActionMapping;

**import** tests.service.TestService;

**import** architecture.ee.web.struts.action.FrameworkDispatchActionSupport;

**import** architecture.ee.web.util.ApplicatioinConstants;

**import** architecture.ee.web.util.ModelMap;

**import** architecture.ee.web.util.ParamUtils;

**public** **class** TestAction **extends** FrameworkDispatchActionSupport {

**public** ActionForward getList(

ActionMapping mapping,

ActionForm form,

HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

String statement = ParamUtils.*getParameter*(

request, "statement", "TEST.SELECT\_ALL\_TABLE\_NAMES");

TestService service = getComponent(TestService.**class**);

List<Map<String, Object>> list = service.queryForList(statement);

ModelMap model = **new** ModelMap();

model.put("list", list);

request.setAttribute(ApplicatioinConstants.*MODEL\_ATTRIBUTE*, model);

**return** (mapping.findForward("success"));

}

}

### 스트럿츠 예외처리

스트럿트는 예외 처리를 위하여 ActionMessage클래스를 사용할 것을 권장하고 있다. ActionMessage클래스는 어플리케이션에서 발생된 예외를 잡아 뷰에 에러 메시지들을 캡슐화하여 전송한다. 뷰에 전달된 ActionMessage 는 스트럿츠에서 제공되는 유틸리티 (태그 라이브러리)를 사용하여 에러 메시지로 랜더링된다.



|  |
| --- |
| QueryActioin.java |

/WEB

**package** tests.action;

**import** java.util.List;

**import** java.util.Map;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.struts.action.ActionForm;

**import** org.apache.struts.action.ActionForward;

**import** org.apache.struts.action.ActionMapping;

**import** tests.service.TestService;

**import** architecture.ee.web.struts.action.FrameworkDispatchActionSupport;

**import** architecture.ee.web.util.ApplicatioinConstants;

**import** architecture.ee.web.util.ModelMap;

**import** architecture.ee.web.util.ParamUtils;

**public** **class** TestAction **extends** FrameworkDispatchActionSupport {

**public** ActionForward getList(

ActionMapping mapping,

ActionForm form,

HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

try{

String statement = ParamUtils.*getParameter*(

request, "statement", "TEST.SELECT\_ALL\_TABLE\_NAMES");

TestService service = getComponent(TestService.**class**);

List<Map<String, Object>> list = service.queryForList(statement);

ModelMap model = **new** ModelMap();

model.put("list", list);

request.setAttribute(ApplicatioinConstants.*MODEL\_ATTRIBUTE*, model);

**return** (mapping.findForward("success"));

}catch(Exception e){

logger.error(ex);

ActionMessages errors = new ActionMessages();

errors.add(ActionMessages.GLOBAL\_MESSAGE, **new** ActionMessage(ex.getMessage()));

saveErrors(request, errors);

return (mapping.findForward("error"));

}

}

}

### 개선된 예외처리

액션 구현 클래스에서 예외가 발생되었을 때 ActionMessage 클래스를 사용하여 오류 메시지를 뷰에 전달하는 것은 개발자가 직접 예외 처리 코드를 구현하여야 한다. FUSE™ ARCHITECTURE 2.0는 코드 기반의 오류 처리를 지원하기 위하여 ApplicationException 예외 클래스와 Coedable 인터페이스를 제공하고 있다. 이들을 구현하거나 확장하여 새로운 예외 클래스를 구현하거나 오류 코드와 메시지만을 정의하고 ApplicationException 클래스를 재 사용하여 사용할 수 있다.



Action 클래스는 예외가 발생되는 경우 다음과 같이 처리한다.

1. 발생이 예상되는 예외를 캐치하는 코드를 작성한다.
2. Codeable 인터페이스를 구현하는 예외가 아닌 경우 예상된 예외의 경우는 해당하는 오류 코드를 인자로 ApplicationException 객체를 생성한다.
3. 지정된 예외 페이지로 포워딩하거나 예외를 던져 올린다.

|  |
| --- |
| QueryActioin.java |

/WEB

**package** tests.action;

**import** java.util.List;

**import** java.util.Map;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.struts.action.ActionForm;

**import** org.apache.struts.action.ActionForward;

**import** org.apache.struts.action.ActionMapping;

**import** tests.service.TestService;

**import** architecture.ee.web.struts.action.FrameworkDispatchActionSupport;

**import** architecture.ee.web.util.ApplicatioinConstants;

**import** architecture.ee.web.util.ModelMap;

**import** architecture.ee.web.util.ParamUtils;

**public** **class** TestAction **extends** FrameworkDispatchActionSupport {

**public** ActionForward getList(

ActionMapping mapping,

ActionForm form,

HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

try{

String statement = ParamUtils.*getParameter*(

request, "statement", "TEST.SELECT\_ALL\_TABLE\_NAMES");

TestService service = getComponent(TestService.**class**);

List<Map<String, Object>> list = service.queryForList(statement);

ModelMap model = **new** ModelMap();

model.put("list", list);

request.setAttribute(ApplicatioinConstants.*MODEL\_ATTRIBUTE*, model);

**return** (mapping.findForward("success"));

}catch(Exception e){

if(isCodeable(e))

throw e;

else

throw **new** ApplicationException(200001, ex.getMessage(), e);

}

}

}

### JSP를 이용한 뷰 구현

전형적인 PUSH 방식의 Struts 에서 Action 은 뷰에 해당하는 JSP에게 ModelMap 객체를 넘겨주고 JSP 는 이미 약속된 이름으로 ModelMap 을 꺼내어 사용하게 된다.

ModelMap model = **new** ModelMap();

model.put("message", “hello”);

request.setAttribute(ApplicatioinConstants.*MODEL\_ATTRIBUTE*, model);

JSP 는 동일한 키를 사용하여 request 에서 ModelMap 객체를 꺼내어 사용한다.

<%@ page import=*"architecture.ee.web.util.ApplicatioinConstants"*%>

<%@ page import=*"architecture.ee.web.util.ParamUtils"*%>

<%@ page import=*"architecture.ee.web.util.ModelMap"* %>

<%

ModelMap model = (ModelMap)request.getAttribute(

ApplicatioinConstants.MODEL\_ATTRIBUTE );

%>

<%= model.get("message") %>

## Struts 2 기반 MVC 프레임워크 사용하기



### Struts 2 소개

FUSE™ ARCHITECTURE 2.0 플랫폼은 Model-View-Controller[[4]](#footnote-4) (이하 MVC) 구현을 위하여 Struts2를 사용하고 있다. Struts2는 2001년 공개된 이후 자바 영역에서 웹 어플리케이션 개발을 위한 표준 프레임워크로 널리 사용되었던 Struts1 과는 사실상 다른 프레임워크이다. (Struts2는 공개 소프트웨어 WebWork 2.2를 계승하는 프로젝트로 Struts1 보다는 WebWork 2.2 와 유사하다)



그림 IV‑1 Front Controller MVC Model

초기 대부분의 웹 어플리케이션들은 Page Controller 패턴을 사용하였다. Page Controller 패턴은 요청이 많을 경우 부하가 걸린다는 것이고 하나의 페이지마다 하나씩의 Page Controller 가 존재한다는 문제점을 가지고 있었다. Struts 1 과 Struts 2 는 Page Controller 패턴의 문제점을 해결한 Front Controller 디자인 패턴[[5]](#footnote-5)을 이용한 MVC 모델이라는 공통점을 가지고 있다. Struts 1 과 Struts 2 의 MVC 모델의 차이는 Push Model & Pull Model 방식으로 설명할 수 있다. Struts1 에 적용된 Push Model 은 초기 MVC 모델에 주로 채택되었던 방법으로 사용자의 요청에 의해 처리된 결과를 View 에 밀어 넣는 (Push) 방식이고, Struts 2 에 적용된 Pull 방식은 View 에서 다수의 Controller 로 부터 필요에 따라 처리 결과를 댕겨오는 (Pull) 방식이다. Pull Model 은 필요한 컴포넌트를 호출하여 처리하는 방식이 가능하기 때문에 이를 Component MVC 라고도 부르며, 최근에는 많은 웹 어플리케이션 프레임워크들이 이 방식을 사용하고 있다.



그림 IV‑2 Push Model



그림 IV‑3 Pull Model

### Struts 2 아키텍처

그림 IV‑4는 Struts 2 프레임워크의 아키텍처를 보여준다.



그림 IV‑4 Struts 2 아키텍처

**FilterDispatcher**

표준 필터 FilterDispatcher 는 ActionMapper 를 사용하여 Acton 을 불러낼 것인가를 결정한다. 만일 불러내야 한다면 ActionProxy 에게 제어를 위임한다.

**ActionProxy**

ActionProxy 는 Struts.xml 파일을 사용하여 초기화되는 ConfigurationManager 는 참조하여 명령 패턴을 구현하는 ActionInvocation 을 생성한다. ActionInvocation 은 (설정된 경우) Inteceptor들을 호출 하고 Action 을 호출한다. ActionInvocation 는 Action 호출 결과에 해당하는 Result 를 찾는다. 랜더링될 JSP 또는 Freemarker 같은 템플릿을 포함하는 Result 를 실행한다. 이제 반대 순서로 Inteceptor 들을 호출한다.

### Struts 2 환경설정

Struts 2 프레임워크를 사용하기 위하여 아래와 같이 WEB-INF/web.xml 파일에 filter 와 filter-mapping 노드를 추가한다.

|  |
| --- |
| /WEB-INF/web.xml |

/WEB

<filter>

<filter-name>struts2</filter-name>

<filter-class>

org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>\*.do</url-pattern>

</filter-mapping>

다음으로 URL 과 자바 클래스 그리고 View 페이지간의 관계를 정의하기 위하여 어노테이션(Annotation) 또는struts.xml 파일 사용한다. 본 문서에서는 struts.xml 파일을 사용한다.

## Sitemesh 기반 UI 구현



### Sitemesh 소개

FUSE™ ARCHITECTURE 2.0 플랫폼은 Gang of Four 의 Decorator Pattern 을 적용하여 UI 화면에서 콘텐츠 만을 분리하는 것을 가능하게 하는 Sitemesh 프레임워크를 지원한다.

Sitemesh 을 사용하면 오직 콘텐츠에 해당하는 영역만을 작업하여 웹 응용프로그램을 구현하는 것이 가능하게 된다.



### Sitemesh 설치

Sitemesh 를 사용할 수 있도록 web.xml에 설정을 추가한다.

|  |
| --- |
| WEB-INF/web.xml |

<filter>

<filter-name>sitemesh</filter-name>

<filter-class>com.opensymphony.sitemesh.webapp.SiteMeshFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>sitemesh</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

이제 모든 요청에 SiteMeshFilter 가 적용된다. 이제 WEB-INF/decorations.xml 파일을 생성하여 SiteMeshFilter 가 어떻게 페이지들을 꾸밀 것인가를 설정한다. 아래의 파일과 같은 설정에서는 index.jsp 를 호출하는 경우 /decorators/main.jsp를 사용하여 index.jsp를 완전하게 한다.



|  |
| --- |
| WEB-INF/decorations.xml |

<?xml version=*"1.0"* encoding=*"ISO-8859-1"*?>

<decorators defaultdir=*"/decorators"*>

<excludes>

<pattern>/css/\*</pattern>

<pattern>/images/\*</pattern>

<pattern>/includes/\*</pattern>

</excludes>

<decorator name=*"main"* page=*"main.jsp"*>

<pattern>/\*</pattern>

</decorator>

<decorator name=*"panel"* page=*"panel.jsp"*/>

<decorator name=*"printable"* page=*"printable.jsp"*/>

<decorator name=*"black"* page=*"black.jsp"*/>

</decorators>

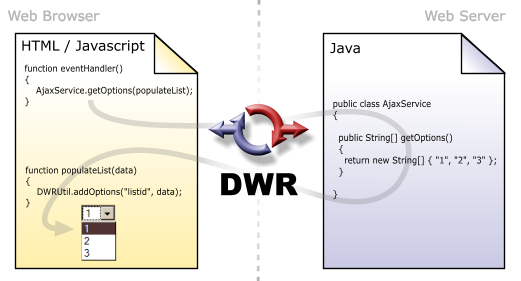
## DWR을 이용한 동적 웹 구현



### DWR 소개

DWR은 AJAX(Asynchronous JavaScript and XML)기술을 웹 사이트에 적용하기 위한 공개소프트웨어이다. DWR 는 크게 2개의 파트로 구성된다.

* 서버측의 자바 서블릿은 서버로 요청 데이터를 브라우저로 전달한다.
* 브라우저에서 구동되는 JavaScript는 동적으로 서버의 자바 코드를 호출하고 그 결과를 리턴한다.



### DWR 설치

DWR 를 사용할 수 있도록 web.xml에 설정을 추가한다.

|  |
| --- |
| WEB-INF/web.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app id=*"WebApp\_ID"*>

<servlet>

<servlet-name>dwr</servlet-name>

<servlet-class>

org.directwebremoting.spring.DwrSpringServlet

</servlet-class>

<init-param>

<param-name>debug</param-name>

<param-value>true</param-value>

</init-param>

<init-param>

<param-name>crossDomainSessionSecurity</param-name>

<param-value>false</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>dwr</servlet-name>

<url-pattern>/dwr/\*</url-pattern>

</servlet-mapping>

</web-app>

DWR 에서 사용할 서비스 및 데이터 타입을 설정하기 위하여 dwrSubsystemContext.xml 파일을 생성하고 webApplicaitonContext.xml 에 추가한다.

|  |
| --- |
| WEB-INF/context-config/dwrSubsystemContext.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xmlns:dwr=*"http://www.directwebremoting.org/schema/spring-dwr"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd*

*http://www.directwebremoting.org/schema/spring-dwr http://www.directwebremoting.org/schema/spring-dwr-2.0.xsd"*

default-autowire=*"no"* default-init-method=*"init"*

default-destroy-method=*"destroy"*>

<dwr:configuration>

<dwr:convert class=*"java.lang.Exception"* type=*"exception"*>

<dwr:include method=*"message"*/>

</dwr:convert>

<dwr:signatures><![CDATA[

import java.util.Map;

import java.util.List;

]]></dwr:signatures>

</dwr:configuration>

<bean id=*"dwrDataService"* class=*"tests.service.DwrDataService"* p:dataService-ref=*"dataService"*>

<dwr:remote javascript=*"dataservice"*>

<dwr:include method=*"queryForList"*/>

</dwr:remote>

</bean>

</beans>

# 서비스 & 비즈니스 계층





## 비즈니스 서비스



### 서비스 구현



비즈니스 서비스는 인터페이스, 구현체 디자인 스타일을 따른다. 구현된 클래스는 다음과 같이 WEB-INF/context-config/serviceSubsystemContext.xml 에 정의하여 사용하게 된다.

### 서비스에서 로깅

서비스에서 로깅 구현을 위하여 서비스 클래스는 apache commons 의 [logging](http://commons.apache.org/logging/) 을 사용한다. 로깅은 클래스에 다음과 같이 Log 객체를 멤버 변수로 생성한 다음 사용한다.

**import** org.apache.commons.logging.Log;

**import** org.apache.commons.logging.LogFactory;

**public** **class** ICapServiceImpl **implements** ICapService {

**protected** Log log = LogFactory.*getLog*(getClass());

..

로그를 남길 때는 성능을 고려하여 반듯이 사용하는 로그 레벨에서 로깅이 가능한 가를 확인하여 처리하도록 한다.

**if**( log.isDebugEnabled() )

log.debug( "logging here .." );

참고로 제공되는 Support 클래스를 확장하여 구현하는 경우는 Log 객체 생성 과정 없이 부모에서 제공하는 log 멤버를 사용하면 된다.

### 서비스에서 에러 처리

서비스에서 에러 처리는 에러 코드 기반의 예외 처리 패턴을 따르며, 손쉬운 구현을 위하여ApplicationException 클래스 와 Codeable 인터페이스를 사용한다. 에러코드는 반듯이 60000 번 이상을 사용한다. 0 ~ 59999 오류 코드는 FUSE™ ARCHITECTURE 2.0 플랫폼 내부용으로 예약되어 있다.

**ApplicationException 을 이용한 에러 처리**

|  |
| --- |
| DataServiceImpl.java |

**package** tests.service.impl;

**import** java.util.List;

**import** tests.service.DataService;

**import** architecture.ee.exception.ApplicationException;

**import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;

**public** **class** DataServiceImpl **extends** SqlQueryDaoSupport **implements** DataService {

**public** List<String> getTablenames() **throws** ApplicationException {

**try** {

String statement = "ICAP.SELECT\_ALL\_TABLE\_NAMES";

**return** getSqlQuery().list(statement, String.**class** );

} **catch** (Exception e) {

// 80001 : 테이블 이름 조회 오류

**throw** **new** ApplicationException (80001, e);

}

}

}

**ApplicationException 을 확장한 Exception 객체를 이용한 에러 처리**

|  |
| --- |
| DataServiceException.java |

DataServiceException.java

**package** tests.service;

**import** architecture.ee.exception.ApplicationException;

**public** **class** DataServiceException **extends** ApplicationException {

**public** DataServiceException() {

**super**();

}

**public** DataServiceException(**int** errorCode, String msg, Throwable cause) {

**super**(errorCode, msg, cause);

}

**public** DataServiceException(**int** errorCode, String msg) {

**super**(errorCode, msg);

}

**public** DataServiceException(**int** errorCode, Throwable cause) {

**super**(errorCode, cause);

}

**public** DataServiceException(**int** errorCode) {

**super**(errorCode);

}

**public** DataServiceException(String msg, Throwable cause) {

**super**(msg, cause);

}

**public** DataServiceException(String msg) {

**super**(msg);

}

**public** DataServiceException(Throwable cause) {

**super**(cause);

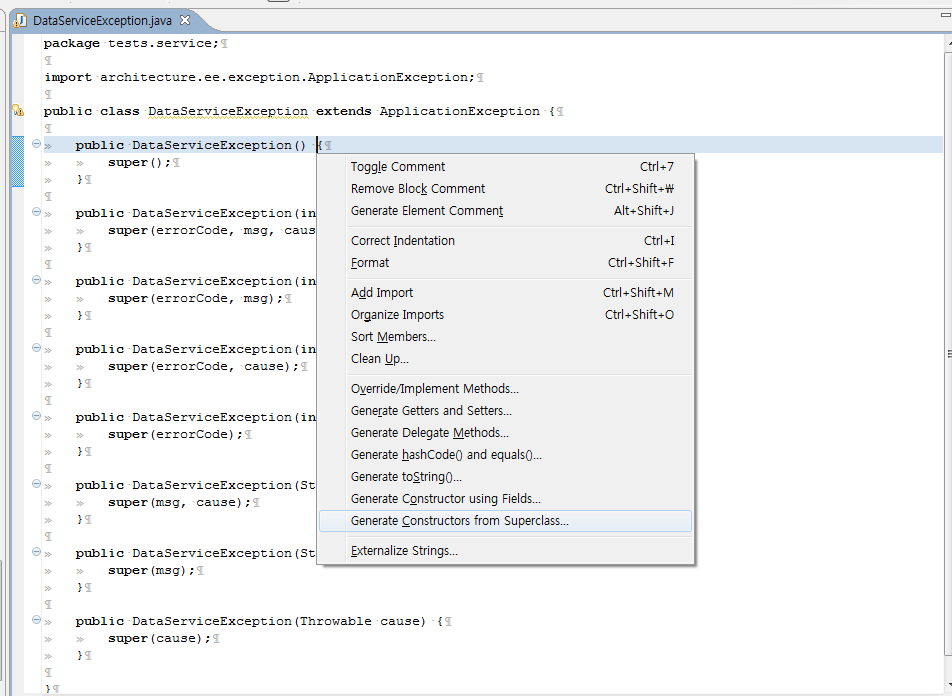
}

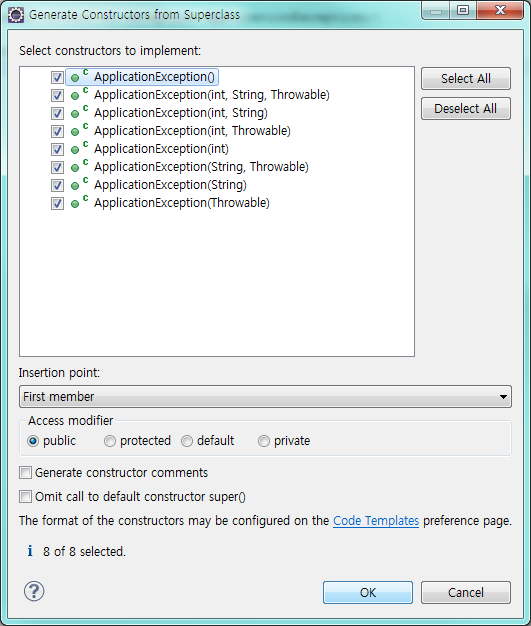
}

ApplicationException 클래스를 확장하는 에러 클래스는 반듯이 슈퍼클래스에 정의된 생성자들을 생성하여야 한다.

C:\Users\donghyuck\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\K0O7W0UI\MC900311758[1].wmf

Eclipse 소스 에디터에서 단축키 “Alt + Shift + S” 키를 누면 보여지는 메뉴에서 “Generate Constructors from Superclass……” 를 클릭하여 Generate Constructors from Superclass 창을 띄운다. 창의 OK 버튼을 클릭하면 자동으로 슈퍼 클래스에 정의된 생성자들이 생성된다.





|  |
| --- |
| DataServiceImpl.java |

DataServiceException DataServiceImpl.java

**package** tests.service.impl;

**import** java.util.List;

**import** tests.service.DataService;

**import** tests.service.DataServiceException;

**import** architecture.ee.exception.ApplicationException;

**import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;

**public** **class** DataServiceImpl **extends** SqlQueryDaoSupport **implements** DataService {

**public** List<String> getTablenames2() **throws** DataServiceException {

// 80001 : 테이블 이름 조회 오류

**try** {

String statement = "ICAP.SELECT\_ALL\_TABLE\_NAMES";

**return** getSqlQuery().list(statement, String.**class** );

} **catch** (Exception e) {

**throw** **new** DataServiceException (80001, e);

}

}

}

### 서비스에서 데이터베이스 접근

FUSE™ ARCHITECTURE 2.0 플랫폼은 데이터베이스 접근을 위하여 SqlQuery 라 불리는 컴포넌트를 사용한다. 서비스 역시 데이터베이스 접근은 SqlQuery 를 사용하며, 이를 위하여 서비스 클래스는 SqlQueryDaoSupport 클래스를 상속하여 사용한다.

|  |
| --- |
| DataServiceException.java |

**package** tests.service.impl;

**import** java.util.List;

**import** tests.service.DataService;

**import** tests.service.DataServiceException;

**import** architecture.ee.exception.ApplicationException;

**import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;

**public** **class** DataServiceImpl **extends** SqlQueryDaoSupport **implements** DataService {

**public** List<String> getTablenames2() **throws** DataServiceException {

// 80001 : 테이블 이름 조회 오류

**try** {

String statement = "ICAP.SELECT\_ALL\_TABLE\_NAMES";

**return** getSqlQuery().list(statement, String.**class** );

} **catch** (Exception e) {

**throw** **new** DataServiceException (80001, e);

}

}

}

구현된 서비스 클래스는 다음과 같이 WEB-INF/context-config/serviceSubsystemContext.xml 에 정의하여 사용하게 된다. 여기에서 주의할 것은 데이터베이스 접근의 경우 반듯이 parent 와 p:dataSource-ref 속성의 값을 지정해 주어야 한다.

|  |
| --- |
| WEB-INF/context-config/servicesSubsystemContext.xml |

WEB-INF/context-config/servicesSubsystemContext.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:c=*"http://www.springframework.org/schema/c"* xmlns:aop=*"http://www.springframework.org/schema/aop"*

xmlns:jee=*"http://www.springframework.org/schema/jee"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:lang=*"http://www.springframework.org/schema/lang"* xmlns:util=*"http://www.springframework.org/schema/util"*

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

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd"*>

<bean id=*"dataService"* class=*"tests.service.impl.DataServiceImpl"*

parent=*"sqlQuerySupport"* p:dataSource-ref=*"dataSource"* />

</beans>

SqlQuery 에 대한 자세한 설명은 6.2 을 참조한다.

# 데이터 계층





## Spring 프레임워크 기반의 트랜잭션 관리



### 선언형 트랜잭션 관리 (Declarative transaction Management)

### @Tranasctional 어노테이션을 사용하기

선언형 트랜젝션 관리 기능을 사용하기 위하여 XML 파일에 트랜잭션 설정을 기술하는 방법 이외에 자바의 Annotation 기능을 사용하여 트랜잭션 설정하는 방법이 있다. 어노테이션 기능은 webApplicationContext.xml을 아래와 같이 수정하여 사용할 수 있다.

|  |
| --- |
| WEB-INF/context-config/webApplicationContext.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:jee=*"http://www.springframework.org/schema/jee"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:context=*"http://www.springframework.org/schema/context"*

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

*http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<description><![CDATA[ .. ]]></description>

<import resource=*"classpath:context/transactionSubsystemcontext.xml"*/>

</beans>

**추가하여 어노테이션 기반 트랜잭션 기능을 활성화 한다.**

아래의 예제를 보면 @Transaction 을 트랜잭션 처리하고자 하는 함수에 기술하여 트랜잭션을 관리하는 것을 알 수 있다. 예제에서는 RuntimeException 또는 Error 유형의 예외가 발생되면 rollback 처리 된다.

|  |
| --- |
| tests.service.impl.DataServiceImpl.java |

**package** tests.service.impl;

**import** java.util.List;

**import** org.springframework.transaction.annotation.Transactional;

**import** tests.service.DataService;

**import** tests.service.DataServiceException;

**import** architecture.ee.exception.ApplicationException;

**import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;

**@Transactional**

**public** **class** DataServiceImpl **extends** SqlQueryDaoSupport **implements** DataService {

**public** List<String> getTablenames2() **throws** DataServiceException {

// 80001 : 테이블 이름 조회 오류

**try** {

String statement = "ICAP.SELECT\_ALL\_TABLE\_NAMES";

**return** getSqlQuery().list(statement, String.**class** );

} **catch** (Exception e) {

**throw** **new** DataServiceException (80001, e);

}

}

**public** List<String> getTablenames() **throws** ApplicationException {

// 80001 : 테이블 이름 조회 오류

**try** {

String statement = "ICAP.SELECT\_ALL\_TABLE\_NAMES";

**return** getSqlQuery().list(statement, String.**class** );

} **catch** (Exception e) {

**throw** **new** ApplicationException (80001, e);

}

}

}

@Transaction 에 추가적인 속성을 정의하여 사용할 수도 있는데 가장 유용하게 사용되는 것이 rollbackFor 이다. 기본적으로 rollback 는 RuntimeException 또는 Error 유형의 예외가 발생되는 경우에만 수행된다. 만일 사용자가 정의한 특정 Exception 이 발생되는 경우에 rollback 처리하고자 한다면 아래와 같이 사용하면 된다.

@Transaction( rollbackFor { Exception.class } )

추가적인 속성들에는 다음과 같은 것들이 있다.

|  |  |
| --- | --- |
| 속성 | 설명 |
| isolation | 트랜잭션의 고립수준(독립성 수준 = 트랜잭션 격리 수준)을 정의하며 디폴트는 데이터베이스의 격리수준을 따른다. |
| noRollbackFor | 정의된 예외들에 대해서는 rollback 을 수행하지 않는다. |
| noRollbackForClassName | 정의된 예외들에 대해서는 rollback 을 수행하지 않는다. |
| propagation | 트랜잭션 영역 내에서 실행되는 모든 코드는 행당 트랜잭션 내에서 실행된다. 만일 트랜잭션 컨텍스트가 이미 존재하는 경우에는 트랜잭션 동작을 지정하는 옵션을 의미한다. |
| readOnly | 트랜잭션을 읽기 전용으로 처리한다.  @Transaction (readOnly=true) |
| rollbackFor | 정의된 예외들에 대해서는 rollback 을 수행한다. |
| rollbackForClassName | 정의된 예외들에 대해서는 rollback 을 수행한다. |
| timeout | 지정된 시간 내에 프로그램 실행이 완료되지 않는다면 rollback를 수행한다. 디폴트 값은 -1 이며, -1은 타임아웃이 없음을 의미한다.  @Transaction ( timeout=10) |

트랜잭션 격리 수준에는 다음이 있다.

|  |  |
| --- | --- |
| 속성 | 설명 |
| ISOLATION\_DEFAUTL | Commit 되지 않는 데이터를 읽을 수 없다. |
| ISOLATION\_READ\_COMMITTED | 다른 트랜젝션에서 commit 하지 않은 변경데이터를 읽을 수 있다. 잠금/해제가 일어나지 않기 때문에 데이터 일관성이 보장되지 않는다. |
| ISOLATION\_READ\_UNCOMMITTED | 다른 트랜젝션에서 commit 하지 않은 변경데이터를 읽을 수 없다. 동일 객체를 여러 번 조회하는 경우 다른 값을 읽을 수 있다. 즉 매번 객체의 값이 변경될 수 있다. 대부분의 데이터베이스들이 디폴트로 지원하는 격리 레벨이다. |
| ISOLATION\_REPEATABLE\_READ | 다른 트랜잭션에서 commit 하지 않은 변경데이터를 읽을 수 없다. 동일 객체를 여러 번 조회하는 경우 같은 값을 읽는 것을 보장한다. |
| ISOLATION\_SERIALIZABLE | 어떤 간섭도 허용하지 않는다. 잠금/해제로 인한 과부하가 높으나 데이터의 일관성을 보장한다. 트랜잭션이 순차적으로 실행되는 것처럼 지원한다. |

트랜잭션 Propagation 행위에는 다음이 있다.

|  |  |
| --- | --- |
| 속성 | 설명 |
| PROPAGATION\_MADATORY | 반듯이 트랜잭션 내에서 함수가 실행되어야 한다. |
| PROPAGATION\_NESTED | 트랜잭션이 있는 경우, 기존 트랜잭션 내의 nested transaction 형태로 함수를 실행하고, nested transaction 자체적으로 commt, rollback 이 가능하다.  트랜잭션이 없는 경우 PROPAGATION\_REQUIRED 속성으로 행동한다. Nested Transaction 에서 변경사항이 commit 된 이후에 기존 트랜잭션에서 조회가 가능하다. |
| PROPAGATION\_NEVER | MANDATORY 와 반대로 트랜잭션 없이 실행된다. |
| PROPAGATION\_NOT\_SUPPORTED | 트랜잭션 없이 함수를 실행하며 기존 트랜잭션이 있는 경우에는 이 트랜잭션을 호출한 함수가 처리될 때까지 대기한다. |
| PROPAGATION\_REQUIRED | 기존 트랜잭션이 있는 경우 기존 트랜잭션에서 실행하고, 없는 경우 새로운 트랜잭션을 생성한다. |
| PROPAGATION\_REQUIRED\_NEW | 호출되는 함수는 자신 만의 트랜잭션을 가지고 실행하며, 기존 트랜잭션들은 보류된다. |
| PROPAGATION\_SUPPORTS | 새로운 트랜잭션을 필요로 하지 않지만 기존의 트랜잭션이 있는 경우 트랜잭션 내에서 실행한다. |

### 프로그래밍을 통한 트랜잭션 관리

응용프로그램이 업데이트와 같은 특정 작업에만 트랜잭션 관리가 요구되고 그 수가 많지 않는 경우라면 프로그래밍을 통한 트랜잭션 관리를 권장한다. 이는 프록시 기술 기반의 선언형 트랜잭션 관리 또는 어노테이션을 사용하는 경우는 많은 자원을 사용하기 때문이다.

## SqlQuery



### SqlQuery 소개

SqlQuery 컴포넌트는 소스 코드와 SQL(structured query language)을 분리하여 전통적 JDBC 프로그래밍의 생산성 및 유지보수성을 개선하였다.



1. SQL(Structured query language)들을 XML(Extensible Markup Language) 형식으로 관리하여 소스 코드와 SQL을 분리하여 개발하는 것을 지원.
2. XML 파일 디플로이 성능 개선.
3. SQL 상의 기법을 사용하지 않고 특정 범위 내의 쿼리 결과조회 지원. (예를 들면 조회 결과 중 처음 1000 번째 부터 2000 째 까지 형식으로)
4. 대용량 데이터 Update, Insert, Delete 작업을 위한 Batch 작업 지원.
5. SQL들이 정의된 XML 파일들은 로컬 디스크를 포함하여 HTTP, HTTPS, FTP, FTPS, RAM 등 여러 파일 시스템 지원한다.  추가로 JAR, ZIP,  TGZ, TBZ2,  GZIP, BZIP2 압축 파일들을 지원.
6. 파라메터 또는 임의로 주어진 값에 따라 동적으로 여러 형태의 쿼리를 생성을 지원한다. 템플릿 언어 기술을 사용하여 강력한 동적 쿼리 지원.
7. SQL 을 실행할 때, 스크립트 모드를 지원. (여러 쿼리를 한꺼번에 실행할 수 있다. 테이블 생성 및 데이터 입력 작업이 가능)
8. 새로운 데이터타입 Recordset 추가.
9. 보다 손쉬운 데이터베이스 작업을 지원하기 위하여 SqlQueryClient 제공.
10. [Core J2EE 패턴](http://java.sun.com/blueprints/corej2eepatterns/index.html)에서 소개하는 DAO(Data Access Object) 디자인 패턴 지원.

### 쿼리 XML

SqlQuery 에서 사용하는 SQL 들이 정의되어 있는 XML 파일은 다음과 같은 규칙을 따른다.

|  |
| --- |
| custom-sqlset.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<sqlset>

<name>FRAMEWORK\_V2</name>

<description>Queries for COMMON</description>

<version></version>

<sql-query name=*"SELECT\_ALL\_LOCALIZED\_PROPERTY"*

description=*"전체 다국어 지원 프로퍼티 값"*><![CDATA[

SELECT

PROPERTY\_NAME,

PROPERTY\_VALUE,

LOCALE\_CODE

FROM V2\_LOCALIZED\_PROPERTIES

]]></sql-query>

</sqlset>

custom-sqlset.xml 파일에 정의된 SQL들은 name 노드 값 “FRAMEWORK\_V2” 그룹으로 등록 된다. 주의할 것은 name 노드 값이 “FRAMEWORK\_V2” 인 다른 XML 파일을 동시에 배포한다면 각 파일에 저장된 SQL 들은 동일 그룹에 해당하게 된다.

### SELECT, UPDATE

SqlQuery 는 입력 파라메터 값으로 java.util.Map 또는 배열을 사용한다. 다음은 배열 값을 사용하는 예시이다.

**하나 이상의 ROW SELECT 하기**

SqlQuery query = getSqlQuery();  
List<Map<String, Object>> rows =   
 query.queryForList("COMMON.SELECT\_ALL\_ENT\_APP");

List<Map<String, Object>> rows =   
 queryForList("COMMON.SELECT\_TABLE\_NAMES\_BY\_LIKE",   
 **new** Object[]{ "V2\_USER%" },  
 **new** **int**[] {Types.*VARCHAR*}, String.**class**);

**단일 ROW SELECT 하기**

SqlQuery query = getSqlQuery();

Integer value = query.queryForObject("COMMON.SELECT\_TABLE\_COUNT",   
Integer.**class**);

String value2 = query.queryForObject("COMMON.SELECT\_TABLE\_NAME",   
**new** Object[]{ "I18N\_COUNTRY" }, **new** **int**[] {Types.*VARCHAR*}, String.**class**);

Map<String, Object> value3 = query.queryForMap("COMMON.SELECT\_TABLE\_NAME",   
**new** Object[]{ "I18N\_COUNTRY" }, **new** **int**[] {Types.*VARCHAR*} );

**UPDATE**

String statement = "BOHUN.INSERT\_IMAGE";

SqlQuery query = getSqlQuery();

query.update( statement,

new Object[] {1, “filename” },

**new** **int**[] {Types.*NUMERIC*, Types.*VARCHAR*}

);

### 배치 업데이트

배치 업데이트 기능을 사용하면 많은 데이터를 변경하고자 하는 경우 유용하게 사용할 수 있다. 또한 단일 트랜잭션으로 처리되기 때문에 성능에 큰 이점을 갖는다. 배치 작업은 SqlQueryHelper 클래스를 사용하여 처리한다.

SqlQuery query = getSqlQuery();  
SqlQueryHelper helper = **new** SqlQueryHelper();

helper.parameters(

**new** Object[]{"1", "name1", "value1"},

**new** **int** [] {Types.*INTEGER*, Types.*VARCHAR*, Types.*VARCHAR*}).inqueue();

helper.parameters(

**new** Object[]{"2", "name2", "value2"},

**new** **int** [] {Types.*INTEGER*, Types.*VARCHAR*, Types.*VARCHAR*}).inqueue();

helper.executeBatchUpdate( query, "COMMON.INSERT\_ENT\_APP\_PROPERTY");

### LOB 데이터

바이너리 또는 텍스트 데이터를 지원하기 위하여 사용되는 LOB 데이터는 SqlQueryHelper 클래스를 사용하여 쉽게 조작할 수 있다. 먼저 다음의 작업을 통하여 서비스에서 LOB 데이터 조작이 가능하도록 한다.

1. webApplicationContext.xml 을 수정하여 LOB 조작을 위한 객체를 생성한다.
2. 구현된 서비스가 정의된 XML ( servicesSubsystemContext.xml ) 을 수정하여 LOB 데이터 조작이 가능하게 한다.

|  |
| --- |
| WEB-INF/context-config/webApplicationContext.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:jee=*"http://www.springframework.org/schema/jee"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:context=*"http://www.springframework.org/schema/context"*

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

*http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<description><![CDATA[ .. ]]></description>

<import resource=*"classpath:context/oracleSubsystemcontext.xml"*/>

</beans>

**추가하여 오라클 LOB 데이터 조작이 가능하게 한다.**

|  |
| --- |
| WEB-INF/context-config/servicesSubsystemContext.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:c=*"http://www.springframework.org/schema/c"* xmlns:aop=*"http://www.springframework.org/schema/aop"*

xmlns:jee=*"http://www.springframework.org/schema/jee"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:lang=*"http://www.springframework.org/schema/lang"* xmlns:util=*"http://www.springframework.org/schema/util"*

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

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd"*>

<bean id=*"dataService"* class=*"tests.service.impl.DataServiceImpl"*

parent=*"sqlQuerySupport"* p:dataSource-ref=*"dataSource"*

p:lobHandler-ref=*"lobHandler"*/>

</beans>

**서비스 객체에서 LOB 데이터 조작이 가능하게 설정을 추가한다.**

서비스에서 사용 예

String statement = "BOHUN.INSERT\_IMAGE";

SqlQueryHelper helper = **new** SqlQueryHelper(getLobHandler());

helper.parameter(104394);

helper.lob(**new** File("C:/Users/donghyuck/Pictures/feather.gif"));

getSqlQuery().update(

statement,

helper.values(),

**new** **int**[] {Types.*NUMERIC*, Types.*BLOB*}

);

### 페이징 처리

SqlQuery 에서 페이징 처리는 setStartIndex 와 setMaxResults 함수를 사용하여 구현할 수 있다. setStartIndex 값은 SQL 결과 데이터 중에서 어디에서부터 데이터를 가져올 것인가를 의미한다 (이 값은 0부터 시작한다). setMaxResults 값은 최대 몇 개의 데이터를 가져올 것인가를 의미한다.

아래의 코드에서 SQL 은 46 개 데이터가 조회 된다. 이때 startIndex 값을 19 로 하고 MasResults 값을 6 으로 하면 index 값이 19 에 해당하는 위치에서부터 6개의 데이터를 리턴하게 된다. 이러한 원칙을 이용하면 손쉬운 페이징 구현이 가능할 것이다.



SqlQuery query = getSqlQuery();  
query.setStartIndex(19);  
query.setMaxResults(6);

List<Map<String, Object>> rows =  
 query.queryForList("COMMON.SELECT\_ALL\_ENT\_APP");

### 동적 쿼리

SqlQuery 는 입력된 파라메터 값 또는 추가적인 파라메터 값에 따라 동적으로 Sql 을 생성하여 사용할 수 있다. 과거 버전과 기능과 비교하면 템플릿 엔진 기술을 적용하여 더욱 강력하게 다양한 형식의 쿼리를 동적으로 처리할 수 있다. 쿼리 XML 에서 동적 쿼리를 적용하는 부분은 반듯이 <dynamic> 태그로 지정을 해주여야 한다.

SqlQuery query = getSqlQuery();  
SqlQueryHelper helper = **new** SqlQueryHelper();

helper.additionalParameter("TABLE\_NAME", "V2\_I18N\_LOCALE");

List<Map<String, Object>> list = helper.list(  
query, "COMMON.SELECT\_TABLE\_ROWS");

<sql-query name=*"SELECT\_TABLE\_ROWS"*

description=*"테이블 명에 해당하는 데이터 조회"*><![CDATA[

select \* from

]]><dynamic><![CDATA[

${TABLE\_NAME}

<#if TABLE\_NAME = "V2\_I18N\_LOCALE" >

WHERE LOCALE\_ID <> 0

</#if>

]]></dynamic>

</sql-query>

이 경우 다음 SQL 이 동적으로 생성되어 실행된다.  
  
V2\_I18N\_LOCALE WHERE LOCALE\_ID <> 0

## SqlQueryClient

SqlQueryClient는 비즈니스 로직이 요구되지 않으며 조회 위주의 작업을 손쉽게 처리할 수 있도록 구현된 서비스로 Action, JSP 및 원격으로 자유롭게 사용할 수 있다.





### 테이블 데이터 엑셀 Export

예제 코드

SqlQueryClient client = …

String catelogName = “”; // 카탈로그 명 또는 null

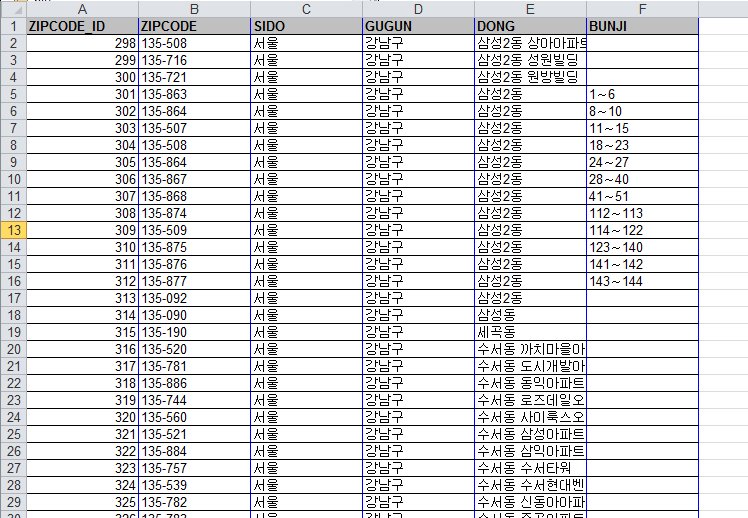
String schemaName = “”; // 스키마 명 또는 null

String tableName = “”; // 테이블 명

Boolean executeAsUnsync = true; // 비동기방식으로 실행 여부

client.exportFromExcel(catelogName, schemaName, tableName, executeAsUnsync);

엑셀파일은 응용프로그램 홈 (웹 응용프로그램인 경우 디폴트 경로는 WEB-INF) 을 기준으로 database/export/[테이블명]\_[yyyyMMddHHmm].xls 형식으로 저장된다.



### 테이블 데이터 엑셀 Import

## 데이터베이스 연결



### DataSource

데이터베이스에 대한 연결은 DataSource 를 통하여 처리된다. DataSource 는 JDBC 표준의 일부로 Servlet/JSP 컨테이너 또는 프레임워크들이 애플리케이션 코드에서 커낵션 풀링과 트랜젝션 관리 이슈들을 숨기는 것을 가능하게 한다. 개발자는 어떻게 데이터베이스에 연결해야 하는가에 대한 세부 사항들을 알 필요가 없다는 의미이다. DataSource 을 얻는 가장 손쉬운 방법은 WAS (Web Application Server) 에 등록된 DataSource 자원을 JNDI 를 통하여 가져오는 것이다.

WAS 에 등록된 데이터소스에 대한 JNDI 이름을 응용프로그램 홈 경로에 위치한 startup-config.xml 에 아래와 같이 기술하여 사용할 수 있다. FUSE™ ARCHITECTURE 2.0 플랫폼 기반의 웹 어플리케이션은 반듯이 하나의 default 데이터소스를 정의하여야 한다.

|  |
| --- |
| startup-config.xml |

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<startup-config>

<setup>

<complete>true</complete>

</setup>

<!-- JDBC DATASOURCE SETTING -->

<database>

<default>

<!-- 1. Use jndi datasource. -->

<jndiDataSourceProvider order=*"1"*>

<jndiName>jdbc/DEFAULT\_DS</jndiName>

</jndiDataSourceProvider>

</default>

|  |
| --- |
| default-database-context.xml |

<bean id=*"dataSource"* class=*"architecture.common.jdbc.datasource.DataSourceFactory"*

destroy-method=*"close"*

factory-method=*"getDataSource"* />

새로운 데이터소스를 추가하는 경우는 default 이름이 아닌 다른 이름으로 노드를 생성하고 default 와 동일한 형태로 xml 정의하면 된다. 응용프로그램의 다른 비즈니스 코드에서 새로운 데이터소스를 사용하기 위해서는 webApplicationContext.xml 파일에 아래의 예제와 과 같이 정의하여 사용할 수 있다.

|  |
| --- |
| startup-config.xml |

<?xml version="1.0" encoding="UTF-8"?>

<startup-config>

새로운 노드 “external” 을 추가하고 JNDI 을 사용하여 DataSource 를 가져오도록 설정하였다.

<setup>

<complete>true</complete>

</setup>

<!-- JDBC DATASOURCE SETTING -->

<database>

<default>

<!-- 1. Use jndi datasource. -->

<jndiDataSourceProvider order="1">

<jndiName>jdbc/DEFAULT\_DS</jndiName>

</jndiDataSourceProvider>

</default>

<external>

<jndiDataSourceProvider order="1">

<jndiName>jdbc/EXTERNAL\_DS</jndiName>

</jndiDataSourceProvider>

</external>

</database>

</startup-config>

|  |
| --- |
| webApplicationContext.xml |

<bean id=*"dataSource2"* class=*"*architecture.ee.spring.jdbc.support*.*DefaultDataSourceProvider*"*

p:dataSourceName=”external” />



커낵션 풀관련 XML 노드와 관련 속성정보는 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **이름** | | | | **횟수** | **설명** |
| database | | | | 1 |  |
|  | default | | | 1..n | 데이터소스 이름 |
|  | jndiDataSoruceProvider | | 1 | JNDI 데이터소스 프로바이더 |
|  | jndiName | 1 | JNDI 이름 |
| pooledDataSoruceProvider | | 1 | 커낵션 풀 데이터소스 프로바이더 |
|  | driverClassName | 1 | 드라이버 클래스 |
| url | 1 | 접속 URL |
| username | 1 | 계정 |
| password | 1 | 비밀번호 |
| maxIdle | 1 | 사용되지 않고 풀에 저장될 수 있는 최대 커넥션 개수. 음수일 경우 제한이 없다. |
| maxActive | 1 | 커넥션 풀이 제공할 최대 커넥션 개수 |
| maxWait | 1 | whenExhaustedAction 속성의 값이 1일 때 사용되는 대기 시간. 단위는 1/1000초이며, 0 보다 작을 경우 무한히 대기한다. |
| validationQuery | 1 |  |
| testOnBorrow | 1 | true일 경우 커넥션 풀에서 커넥션을 가져올 때 커넥션이 유효한지의 여부를 검사한다. |
| testWhileIdle | 1 | rue일 경우 비활성화 커넥션을 추출할 때 커넥션이 유효한지의 여부를 검사해서 유효하지 않은 커넥션은 풀에서 제거한다 |
| timeBetweenEvictionRunsMillis | 1 | 사용되지 않은 커넥션을 추출하는 쓰레드의 실행 주기를 지정한다. 양수가 아닐 경우 실행되지 않는다. 단위는 1/1000 초이다. |
| minEvictableIdleTimeMillis | 1 | 사용되지 않는 커넥션을 추출할 때 이 속성에서 지정한 시간 이상 비활성화 상태인 커넥션만 추출한다. 양수가 아닌 경우 비활성화된 시간으로는 풀에서 제거되지 않는다. 시간 단위는 1/1000초이다. |
| numTestsPerEvictionRun | 1 | 사용되지 않는 커넥션을 몇 개 검사할지 지정한다. |

### 커넥션 풀링 기반의 DataSource

Apache commons-dbcp 를 사용하여 직접 커낵션 풀링을 지원하는 데이터소스를 생성할 수 있다. 주의할 점은 JAVA 에 따라 제공되는 버전이 다르다는 점이다.

* DBCP 1.4 compiles and runs under JDK 1.6 only (JDBC 4)
* DBCP 1.3 compiles and runs under JDK 1.4-1.5 only (JDBC 3)

|  |
| --- |
| webApplicationContext.xml |

<bean id="dataSource"

class="org.apache.commons.dbcp.BasicDataSource"

p:driverClassName=""

p:url=""

p:username=""

p:password="" />

### JDBC 드라이버 기반의 DataSoruce

DriverManagerDataSource 클래스는 풀링 기능을 제공하지 않기 때문에 반듯이 테스트 목적으로 사용되어야 한다.

|  |
| --- |
| webApplicationContext.xml |

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource" p:driverClassName=""

p:url=""

p:username=""

p:password="" />



# 통합 계층



## 웹 서비스

## 메일 서비스

## 메시지 서비스

# 보안

# 참고 문헌

“Core J2EE Pattern Catalog - Front Controller.” (날짜 정보 없음). Oracle: http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html에서 검색됨

GearyDavid. (날짜 정보 없음). “Is Struts Dead? Part III: I can't kill Struts (Struts is Deprecated).” http://www.jroller.com/RickHigh/entry/is\_struts\_dead\_part\_iii에서 검색됨

*ISO 639 Language Codes.* (n.d.). Retrieved from W3C: http://www.w3.org/WAI/ER/IG/ert/iso639.htm

RutenbergGuy. (날짜 정보 없음). “Pull vs. Push MVC Architecture.” Guy Rutenberg: http://www.guyrutenberg.com/2008/04/26/pull-vs-push-mvc-architecture/에서 검색됨

“Struts 2 Architecture - Detail information on Struts 2 Architecture.” (날짜 정보 없음). http://www.roseindia.net/: http://www.roseindia.net/struts/struts2/struts-2-architecture.shtml에서 검색됨

1. 자바 개발자들에게 가장 널리 알려진 공개 소프트웨어 Spring은 Rod Johnson 의 저서 “Expert One-on-One J2EE Design and Development, 2002” 의 코드에 기반하는Layered Java/J2EE 애플리케이션 플랫폼이다. FUSE™ ARCHITECTURE 2.0 은 Spring 의 경량 컨테이너, 트랜잭션 관리를 위한 추상화 계층(Abstract Layer), JDBC 추상화 계층(Abstract Layer), AOP 기능들을 사용하고 있다. [↑](#footnote-ref-1)
2. 제어구조의 반전 (IoC, Inversion of Control) 이라 알려진 테크닉을 통하여 컨테이너는 컨테이너에 포함된 객체의 인스턴스가 생성되는 시점에 이 객체가 의존하고 있는 것들을 자동으로 검색하여 객체에게 넘겨 준다. 예를 들어 특정 데이터소스를 요구하는 A 라는 컴포넌트가 있을 때, 컨테이너는 A 라는 객체의 인스턴스가 생성되는 시점에 A 의 의존성을 판별하고 컨테이너에 등록되어 있는 것들 중 검색을 통하여 데이터소스를 A 에게 넘겨주게 된다. [↑](#footnote-ref-2)
3. MVC 디자인을 수용한 웹 디자인 패턴으로 Model 2 또는 MVC Model 2 라 불린다. [↑](#footnote-ref-3)
4. MVC 디자인 패턴은 어플리케이션을 Model, View, Controller으로 나누어 각 계층간의 결합력이 최소화 될 수 있도록 디자인하는 방식으로, 대부분의 웹 프레임워크들은 UI 와 비즈니스 로직을 분리하여 개발자와 디자이너 사이에 작업을 분리하고 컴포넌트 형태로 개발할 수 있도록 MVC 디자인 패턴을 지향하고 있다. 일반적인 웹 어플리케이션들에서, 도메인 모델들은 서버에 저장되며 사용자는 클라이언트에서 동작하는 소프트웨어 (보통 클라이언트 측에서 응답을 랜더링하는 브라우저)와 낮은 수준의 상호작용을 한다. 따라서 전형적인 웹 어플리케이션 개발에서, Model, View 그리고 Controller는 네트워크를 넘어서 서버와 클라이언트에 위치하게 된다. [↑](#footnote-ref-4)
5. Page Controller 패턴은 Controller를 사용자의 요청을 다루기 위한 최초 진입점(initial Point)로 정의하고, 사용하는 패턴. 최초 진입점이 되는 Controller를 FrontController라 하며 이 컨트롤러는 authentication, authorization, buisiness delegating, managing view 등의 작업을 통합해서 제공하는 책임을 갖는다. Controller가 모든 클라이언트 요청의 진입점이 되면서 얻을 수 있는 장점은 요청 관리에 대한 표준화(a standard framework for handling request)이다. 요청을 분산되서 받을 경우에는 중복된 코드가 생산이 되기 때문에 비효율적이다. 진입점을 한 곳으로 모음으로서 클라이언트의 요청을 일관된 방법으로 관리할 수 있게 된다. [↑](#footnote-ref-5)