Developing web application

with APPLICATION ARCHITECTURE for JAVA

목차

[I. 소개 5](#_Toc328140240)

[1.1 주요 특징 6](#_Toc328140241)

[1.1.1 레이어드 아키텍처 스타일 6](#_Toc328140242)

[1.1.2 공개소스 활용 8](#_Toc328140243)

[1.1.3 이중화된 컨테이너 구조 8](#_Toc328140244)

[1.1.4 플러그인 아키텍처 9](#_Toc328140245)

[1.1.5 동적 아키텍처 10](#_Toc328140246)

[1.1.6 설정의 재사용 11](#_Toc328140247)

[1.2 시나리오 14](#_Toc328140248)

[1.2.1 AJAX 와 RESTful 기반의 웹 어플리케이션 14](#_Toc328140250)

[1.2.2 모바일 웹 어플리케이션 14](#_Toc328140251)

[1.2.3 웹 서비스 기반의 미들웨어 15](#_Toc328140252)

[II. 환경 설정 17](#_Toc328140253)

[2.1 개발 환경 설정 18](#_Toc328140255)

[2.2 응용프로그램 환경 설정 18](#_Toc328140256)

[2.3 어플리케이션 서버 환경 설정 19](#_Toc328140257)

[2.4 클러스터 설정 19](#_Toc328140258)

[2.5 싱글사인온(Single Sign On) 설정 20](#_Toc328140259)

[III. 주요 기능 21](#_Toc328140260)

[3.1 보안 22](#_Toc328140261)

[3.2 다국어 & 지역화 22](#_Toc328140265)

[3.1.1 Spring MessageSource지원 22](#_Toc328140266)

[IV. 프레젠테이션 계층 23](#_Toc328140267)

[4.1 Struts 2 기반 MVC 프레임워크 24](#_Toc328140269)

[4.1.1 Struts 2 소개 24](#_Toc328140272)

[4.1.2 Struts 2 아키텍처 26](#_Toc328140273)

[4.1.3 Struts 2 환경설정 28](#_Toc328140274)

[4.2 Struts 기반의 MVC 프레임워크 28](#_Toc328140275)

[4.2.1 Action 지원을 위한 클래스 28](#_Toc328140277)

[4.2.2 JSP를 이용한 뷰 구현 29](#_Toc328140278)

[4.3 View 기술 30](#_Toc328140279)

[V. 서비스 & 비즈니스 계층 31](#_Toc328140280)

[5.1 비즈니스 서비스 32](#_Toc328140282)

[5.1.1 서비스 구현 32](#_Toc328140285)

[5.1.2 서비스에서 로깅 32](#_Toc328140286)

[5.1.3 서비스에서 에러 처리 33](#_Toc328140287)

[5.1.4 서비스에서 데이터베이스 접근 36](#_Toc328140288)

[VI. 데이터 계층 38](#_Toc328140289)

[6.1 Spring 프레임워크 기반의 트랜잭션 관리 39](#_Toc328140291)

[6.1.1 선언형 트랜잭션 관리 (Declarative transaction Management) 39](#_Toc328140294)

[6.1.2 @Tranasctional 사용하기 39](#_Toc328140295)

[6.2 SqlQuery 40](#_Toc328140296)

[6.2.1 SqlQuery 소개 40](#_Toc328140298)

[6.2.2 쿼리 XML 41](#_Toc328140299)

[6.2.3 SELECT, UPDATE 41](#_Toc328140300)

[6.2.4 페이징 처리 42](#_Toc328140301)

[6.2.5 동적 쿼리 43](#_Toc328140302)

[6.3 SqlQueryClient 44](#_Toc328140303)

[VII. 통합 계층 46](#_Toc328140305)

[7.1 웹 서비스 47](#_Toc328140306)

[7.2 메일 서비스 47](#_Toc328140307)

[7.3 메시지 서비스 47](#_Toc328140308)

[VIII. 보안 48](#_Toc328140309)

[참고 문헌 50](#_Toc328140310)

# 소개

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

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

## 주요 특징

### 레이어드 아키텍처 스타일

FUSE™ ARCHITECTURE 2.0 플랫폼은 가장 널리 사용되는 레이어드 아키텍처 스타일을 따른다. 다음은 그림은 이해를 돕기 위하여 업무측면과 개발 측면에서 바라본 레이어드 아키텍처 스타일을 보여주고 있다.



그림 1‑1 업무 및 개발 관점 레이어드 아키텍처 스타일

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



그림 1‑2 레이어드 아키텍처 스타일

### 공개소스 활용

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

### 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)들의 생명주기를 관리하는 실재적 컨테이너로 동작한다.

**

부모에 해당하는Bootstrap Application Context 에는 코어 모듈들이 존재하며 자식에 해당하는 Web Application Context 에는 비즈니스 모듈들이 존재한다. 자식 컨텍스트는 부모 컨텍스트에 정의된 모듈들에 접근이 허용되지만 부모는 자식 컨텍스트에 접근이 허용되지 않는다. 이러한 컨텍스트의 분리는 비즈니스 모듈 개발자들의 설정을 간소화하고 주요한 코어 모듈들에 대한 설정을 보호하는 효과를 제공한다.



### 플러그인 아키텍처

플러그인 기술은 어플리케이션이 동작 중에 모듈들 또는 어플리케이션의 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

Groovy 기술을 사용하여 비즈니스 코드를 수정하더라고 소스 컴파일과 서버의 재 시작 없이 적용될 수 있도록 지원한다. FUSE™ ARCHITECTURE 2.0 은 Struts2 Action 과 비즈니스 서비스 구현 클래스에 대하여 groovy 언어를 사용 하여 구현하는 것을 지원하고 있다.



### 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.

<?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>

## 시나리오



### AJAX 와 RESTful 기반의 웹 어플리케이션



### 모바일 웹 어플리케이션



### 웹 서비스 기반의 미들웨어



# 환경 설정



## 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 | maven | <http://maven.apache.org> |
| IED 툴 | eclipse | <http://www.eclipse.org/> |
| 형상관리 | subversion | <http://subversion.tigris.org/> |

## 응용프로그램 환경 설정

FUSE™ ARCHITECTURE 2.0 플랫폼 기반의 웹 어플리케이션은 동작을 위하여 필요한 환경 설정 정보 및 데이터들을 저장하게 될 응용프로그램 홈 경로를 설정해야 한다. 설정은 다음의 2 방법을 사용하여 설정하거나 웹 기반 어플리케이션 setup 프로그램을 사용하여 설정할 수 있다.

**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. 경로를 지정하지 않는 경우는 디폴트로 WEB-INF/ 위치를 응용 프로그램이 동작하기 위한 홈 경로로 사용하게 된다.

/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

…

<context-param>

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

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

</context-param>

</web-app>

## 어플리케이션 서버 환경 설정

테스트된 프레임워크 기반의 응용프로그램 운영환경

|  |  |  |  |
| --- | --- | --- | --- |
| **환 경** | | | **검증 여부** |
| H/W | IBM p780 | | ○ |
| S/W | IBM AIX 5.3 Oracle Weblogic 10.3 with Java 1.6.x Oracle 10g | |
| H/W | | HP DL380 G7 | ○ |
| S/W | | Tomcat 6.0.30 with Java 1.6.0 Redhat Enterprise Linux Server release 5.5  Oracle 11g(11.2.0) |

기술되지 않은 운영시스템 또는 검증 여부에 체크되지 않는 경우지 설치가 불가능할 수 도 있다. 일반적으로 JDK 1.5 이상을 지원하는 운영시스템에서는 설치가 가능하다.

## 클러스터 설정

## 싱글사인온(Single Sign On) 설정

# 주요 기능

## 

## 보안



## 다국어 & 지역화

### Spring MessageSource지원

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**);

# 프레젠테이션 계층





## Struts 2 기반 MVC 프레임워크



### Struts 2 소개

FUSE™ ARCHITECTURE 2.0 플랫폼은 Model-View-Controller[[3]](#footnote-3) (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 4‑1 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[[4]](#footnote-4)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 4‑2 Push Model



그림 4‑3 Pull Model

### Struts 2 아키텍처

그림 4‑4는 Struts 2 프레임워크의 아키텍처를 보여준다.



그림 4‑4 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.

<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.

## Struts 기반의 MVC 프레임워크



### 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.

QueryAction.java

**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.ICapService;

**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** QueryAction **extends** FrameworkDispatchActionSupport {

**public** ActionForward getList(

ActionMapping mapping,

ActionForm form,

HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

String statement = ParamUtils.*getParameter*(

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

ICapService service = getComponent(ICapService.**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"));

}

}

### JSP를 이용한 뷰 구현

Struts Action 은 JSP 에 ModelMap 객체를 사용하여 넘겨주게 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") %>

## View 기술

# 서비스 & 비즈니스 계층





## 비즈니스 서비스



### 서비스 구현



비즈니스 서비스는 인터페이스, 구현체 디자인 스타일을 따른다. 구현된 클래스는 다음과 같이 WEB-INF/context-config/serviceSubsystemContext.xm 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

**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);

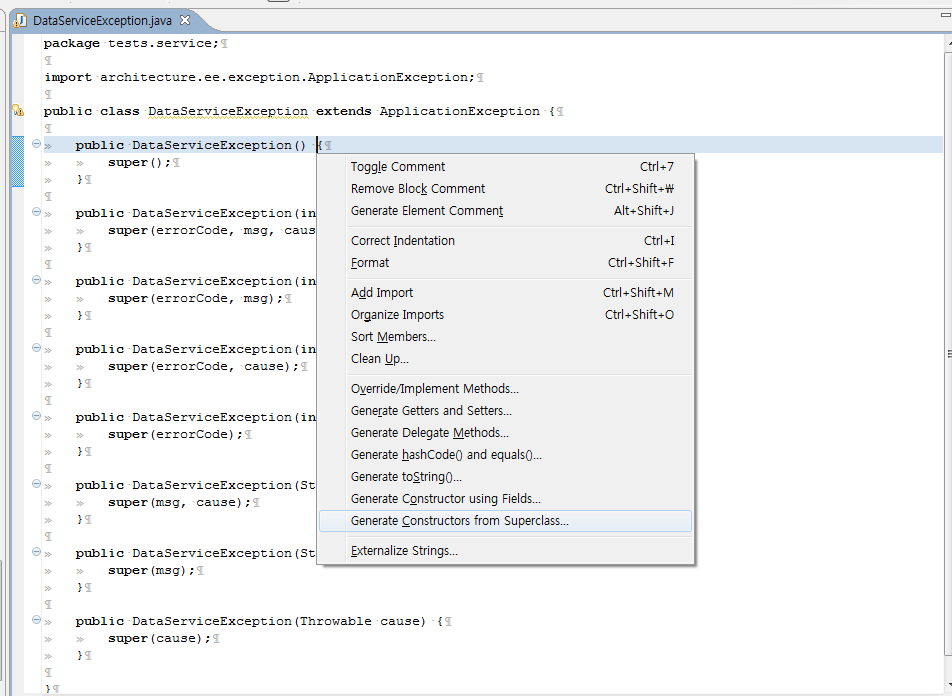
}

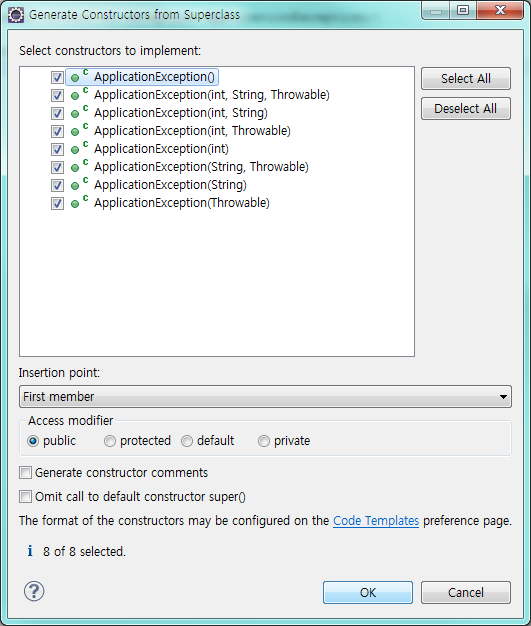
}

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 뛰운다. 창의 OK 버튼을 클릭하면 자동으로 슈퍼 클래스에 정의된 생성자들이 생성된다.





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.

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);

}

}

}

구현된 서비스 클래스는 다음과 같이 WEB-INF/context-config/serviceSubsystemContext.xm 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

<?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 5.1.6 을 참조한다.

# 데이터 계층





## Spring 프레임워크 기반의 트랜잭션 관리



### 선언형 트랜잭션 관리 (Declarative transaction Management)

### @Tranasctional 사용하기

선언형 트랜젝션 관리 기능을 사용하기 위하여 XML 파일에 트랜잭션 설정을 기술하는 방법 이외에 자바의 Annotation 기능을 사용하여 트랜잭션 설정하는 방법이 있다.

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);

}

}

}

## 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 Map 형식을 허용하는 함수들과 오직 객체 배열만을 허용하는 함수들로 구성되어 있다.

**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*} );

**배치 업데이트**

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");

### 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 입력 파라메터 값 또는 추가적인 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>

이경우 다음 SQL 이 실행된다.  
  
V2\_I18N\_LOCALE WHERE LOCALE\_ID <> 0

## SqlQueryClient

SqlQueryClient는 비즈니스 로직이 요구되지 않으며 조회 위주의 작업을 손쉽게 처리할 수 있도록 구현된 서비스이며, SqlQuery와 같이 별도의 서비스 클래스 구현을 요구하지 않는다.





# 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 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-3)
4. 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-4)