|  |
| --- |
| Programming with ARTECTURE EE 2.0 |
| **../../Storage/podo_docs/회사문서/01.회사자료/02.회사로고/logo01_투명.png** |
| Rapid application development platform for the java. |

목차 정보

[1. 소개 4](#_Toc349731365)

[1.1. 소프트웨어 아키텍처 5](#_Toc349731366)

[1.1.1. 레이어드 아키텍처 5](#_Toc349731367)

[1.1.2. 이중화 컨테이너 아키텍처 7](#_Toc349731368)

[1.1.3. 플러그인 아키텍처 8](#_Toc349731369)

[1.1.4. 다이나믹 아키텍처 8](#_Toc349731370)

[1.1.5. 설정의 재사용 9](#_Toc349731371)

[1.2. 공개 소프트웨어 10](#_Toc349731372)

[1.3. 적용 시나리오 10](#_Toc349731373)

[2. 개발 및 운영 11](#_Toc349731374)

[2.1. 개발환경 12](#_Toc349731376)

[2.2. 개발도구 14](#_Toc349731377)

[2.2.1. Eclipse 14](#_Toc349731378)

[2.2.2. Subversion 17](#_Toc349731379)

[2.2.3. Maven 18](#_Toc349731380)

[2.3. 웹 응용프로그램 설치 19](#_Toc349731381)

[2.3.1. 웹 응용프로그램 홈 설정 19](#_Toc349731382)

[2.3.2. 다국어 지원 20](#_Toc349731383)

[2.4. 하드웨어 및 소프트웨어 호환성 22](#_Toc349731384)

[3. 핵심모듈 23](#_Toc349731385)

[3.1. Bootstrap 24](#_Toc349731387)

[3.2. Bootstrap ApplicationProperties 25](#_Toc349731388)

[3.3. ConfigService 27](#_Toc349731389)

[3.4. ApplicationHelper 28](#_Toc349731390)

[3.5. Logging 29](#_Toc349731391)

[3.5.1. 웹 응용프로그램에서 Log4j 사용하기 29](#_Toc349731392)

[3.6. 국제화 & 지역화 30](#_Toc349731393)

[3.6.1. Properties 파일 기반 국제화(I18N) 30](#_Toc349731394)

[3.6.2. 데이터베이스 기반 국제화(I18N) 32](#_Toc349731395)

[4. JDBC 프로그래밍 35](#_Toc349731396)

[4.1. 데이터베이스 핸들링 36](#_Toc349731398)

[4.1.1. DAO 스타일 37](#_Toc349731399)

[4.1.2. SqlQuery 38](#_Toc349731400)

[4.2. 데이터베이스 연결 40](#_Toc349731401)

[4.2.1. DataSource 40](#_Toc349731402)

[4.2.2. DBCP 44](#_Toc349731403)

[4.2.3. JDBC 드라이버 45](#_Toc349731404)

[4.3. SqlQuery 45](#_Toc349731405)

[4.3.1. 쿼리 xml 45](#_Toc349731406)

[4.3.2. DAO 구현하기 48](#_Toc349731407)

[4.3.3. 클래식 스타일 API 51](#_Toc349731408)

[4.3.4. 배치 작업 54](#_Toc349731409)

[4.3.5. LOB 데이터 55](#_Toc349731410)

[4.3.6. 페이징 57](#_Toc349731411)

[4.3.7. 동적 쿼리 59](#_Toc349731412)

[4.3.8. 작업단위패턴(Unit Of Work) 59](#_Toc349731413)

[4.3.9. 새로운 스타일 API 59](#_Toc349731414)

[4.3.10. 파라메터 매핑 60](#_Toc349731415)

[4.4. SqlQueryClient 사용하기 60](#_Toc349731416)

[4.4.1. 기본 API 61](#_Toc349731417)

[4.4.2. 콜백함수 61](#_Toc349731418)

[4.4.3. 단일 트랜젝션 처리 unitOfWork 62](#_Toc349731419)

[4.5. Springframework 기술 기반 트랜잭션 관리 64](#_Toc349731420)

[4.5.1. 프로그램적으로 트랜잭션 처리 65](#_Toc349731421)

[4.5.2. 선언형 트랜잭션관리 67](#_Toc349731422)

[4.5.3. 어노테이션을 통한 관리 67](#_Toc349731423)

[5. 서비스 71](#_Toc349731424)

[5.1. 서비스 구현 72](#_Toc349731426)

[5.2. 로깅 72](#_Toc349731427)

[5.3. 예외처리 73](#_Toc349731428)

[6. 통합 76](#_Toc349731429)

|  |  |
| --- | --- |
|  |  |
| 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. ARCHITECTURE 2.0 for JAVA 플랫폼은 웹을 기반으로 하는 애플리케이션 개발 및 유지보수를 위한 기반 기술을 제공하는 플랫폼으로 DRY(don’t repeat yourself) 원칙과 다양한 동적 프레임워크들의 생산성증대 아이디어를 도입하여 웹 애플리케이션 개발의 복잡성을 극적으로 줄이도록 설계되었다. |

## 소프트웨어 아키텍처

### 레이어드 아키텍처

복잡한 소트트웨어를 디자인할 때 공통적인 아키텍처 원칙을 적용하는 것이 유용하다는 것은 이미 널리 알려진 사실이다. ARCHITECTURE EE for JAVA 는 많은 응용프로그램들을 통하여 유용함이 증명된 레이어드 아키텍처 스타일을 따른다. 다음 그림은 이해를 돕기 위하여 업무측면과 개발 측면에서 바라본 레이어드 아키텍처 스타일을 보여주고 있다.



그림 1‑1 업무 및 개발 관점 레이어드 아키텍처 스타일

ARCHITECTURE 2.0 for JAVA 플랫폼은 업무관점에서 비즈니스 프레임워크 레이어에 위치하며 업무 레이어에서 공통 및 단위 업무 개발을 돕는 역할을 한다. 개발관점에서 레이어드 아키텍처 스타일은 프레젠테이션, 서비스/비즈니스, 퍼시스턴스 레이어로로 구성된다.



**그림 1‑2 레이어드 아키텍처 스타일**

이렇게 논리적인 분리를 통하여 응용프로그램을 구성하는 코드 조각들은 각각의 기능 또는 그 목적에 따라서 레이어 단위로 정리되어 새로운 컴포넌트 추가가 용의하고 향후 유지/보수 역시 더욱 쉽게 수행할 수 있게 한다.

### 이중화 컨테이너 아키텍처

ARCHITECTURE 2.0 for JAVA 플랫폼은 Spring[[1]](#footnote-1)의 경량 컨테이너 기술(IoC[[2]](#footnote-2))을 사용하여 이중화된 응용프로그램 컨텍스트(Application Context) 영역으로 구성된다. 여기에서 컨텍스트는 등록된 모듈(또는 객체)들에 대한 설정과 생명주기를 관리하는 실재적 컨테이너를 의미한다.

이중화된 켄텍스트는 크게 부모 컨텍스트와 자식 컨텍스트로 구성된다. 부모에 해당하는 부스트랩 응용프로그램 컨텍스트(Bootstrap Application Context)에는 코어 모듈(또는 객체)들이 존재하며 자식에 해당하는 웹 응용프로그램 컨텍스트(Application Context) 에는 업무 모듈(또는 객체)들이 존재한다. 또한 자식 컨텍스트는 부모 컨텍스트에 접근이 허용되지만 부모는 자식 컨텍스트에 접근이 허용되지 않는다. 이러한 컨텍스트의 분리는 비즈니스 모듈 개발자들의 설정을 간소화하고 주요한 코어 모듈들에 대한 설정을 임으로 변경하는 것을 보호하는 효과를 제공한다.



그림 1‑3 이중화된 컨텍스트 구조

### 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. ARCHITECTURE 2.0 for JAVA 플랫폼은 플러그인 기술 사용하여 응용프로그램의 확장성을 최대화한다.

### 다이나믹 아키텍처

ARCHITECTURE 2.0 은 스크립트 언어 Groovy 기술을 사용하여 코드를 수정하더라고 소스 컴파일과 서버의 재 시작 없이 적용이 가능한 기술을 지원한다.

**그림 1‑4 Groovy 기술을 이용한 동적 아키텍처**

### 설정의 재사용

ARCHITECTURE 2.0 for JAVA 플랫폼은 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 부스트랩 응용프로그램 컨텐스트(Bootstrap application context) 생성한다. 또한 startup-config.xml 에 정의된 값에 따라 코어 모듈을 생성한다.



그림 1‑5 미리 정의된 spring xml 을 재사용한다.

## 공개 소프트웨어

ARCHITECTURE 2.0 for JAVA 플랫폼는 널리 사용되는 공개소스 기술을 활용하여 구현되었다. 또한 최근 업데이트된 전자정부프레임워크 2.0 과 호환되는 공개소스 기술을 적용하여 전자 정부 프레임워크에서 동작하도록 지원하고 있다.

## 적용 시나리오

|  |  |
| --- | --- |
|  |  |
| 개발 및 운영 | |
|  | ARCHITECTURE 2.0 for JAVA 플랫폼 기반의 응용프로그램 개발을 위한 전반적인 개발 및 운영 환경에 대하여 기술한다. |



## 개발환경

ARCHITECTURE 2.0 for JAVA 플랫폼은 응용 프로그램을 개발을 위한 필수적 활동 (③설계, ④구현, ⑤테스트 및 통합 작업)을 위하여 요구되는 도구를 지원한다. 사용되는 도구들은 공개소프트웨어들로 선정하여 누구나 자유롭게 사용할 수 있도록 하였다.

  
 **그림 2‑1 소프트웨어의 생명주기를 8 단계로 구분할 수 있다. ARCHITECTURE 2.0 은 ③설계, ④구현, ⑤테스트 및 통합 작업을 위한 개발자 도구들을 지원한다.**

지원되는 개발 도구들은 활동 내용에 따라 다음의 4가지로 분류할 수 있다.

|  |  |
| --- | --- |
| **분류** | **설명** |
| Implementation Tools | 코딩 작업을 위한 도구 |
| Test Tools | 작성된 코드를 테스트하기 위한 도구 |
| Configuration Management Tools | 코드 및 산출물에 대한 형상관리를 위한 도구 |
| Build & Deployment Tools | 빌드 및 배포를 위한 도구 |

  
**그림 2‑2 ③설계, ④구현, ⑤테스트 및 통합 작업을 위한 도구로 eclipse, junit, coverture, maven, ant, subversion 을 사용한다.**

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **소프트웨어** | **버전** | **라이선스** |
| **Implementation Tools** | Eclipse | 4.2.0 | EPL & EDL |
| **Test Tools** | JUnit | 4.1.0 | BSD |
| Cobertura | 1.9.4.1 | GUN 2.0 |
| JMeter | 2.8 | Apache License 2.0 |
| **Build & Deployment Tools** | Maven | 3.0.3 | Apache License 2.0 |
| Ant | 1.8.3 | Apache License 2.0 |
| **Configuration**  **Management Tools** | Subversion |  | Subversion License |
| Subversive | 1.0.0 |  |
| TortoiseSVN | 1.7.7 |  |

표 2‑1 사용 소프트웨어 버전 및 라이선스

## 개발도구

### Eclipse

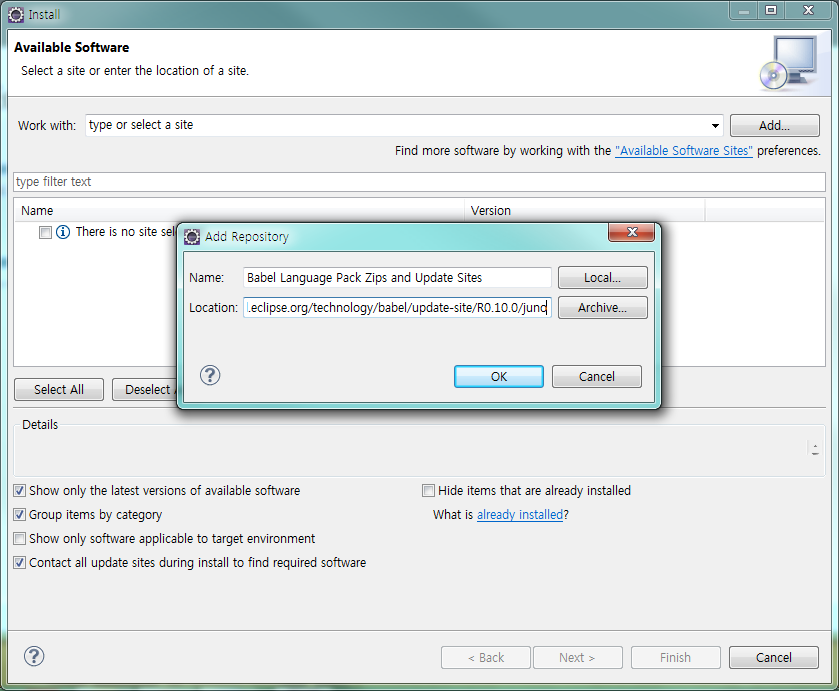
ARCHITECTURE EE 2.0 for JAVA 플랫폼은 코드 구현 도구로 공개 소프트웨어인 Eclipse 를 지원하고 있다. 프로그램은 [www.eclipse.org](http://www.eclipse.org) 사이트를 방문하여 개발자 환경에 맞는 Eclipse버전을 선택하여 다운로드하고 설치한다. 본 문서에서는 Eclipse IDE for Java EE Developers (Juno) 를 사용하였다



그림 2‑3 Eclipse 다운로드 페이지

Eclipse는 영어를 디폴트로 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 그림 2-4와 같이 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 사이트에서 제공하는 언어 팩을 사용한다.

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



**그림 2‑4 Babel Language Pack Update 사이트 추가 화면**

참고로 업데이트 사이트 추가 이후에 설치 가능한 소프트웨어 목록이 보여지는데 많은 시간이 소요된다. Eclipse 하단에서 보여주는 진행상태를 보면 얼마나 더 기다려야 하는가를 가늠할 수 있다.

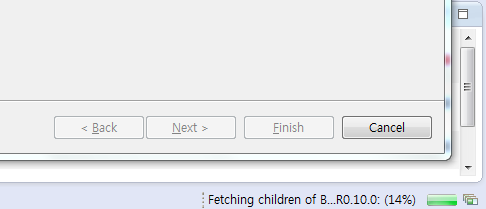


그림 2‑5 Eclipse 하단 상태바

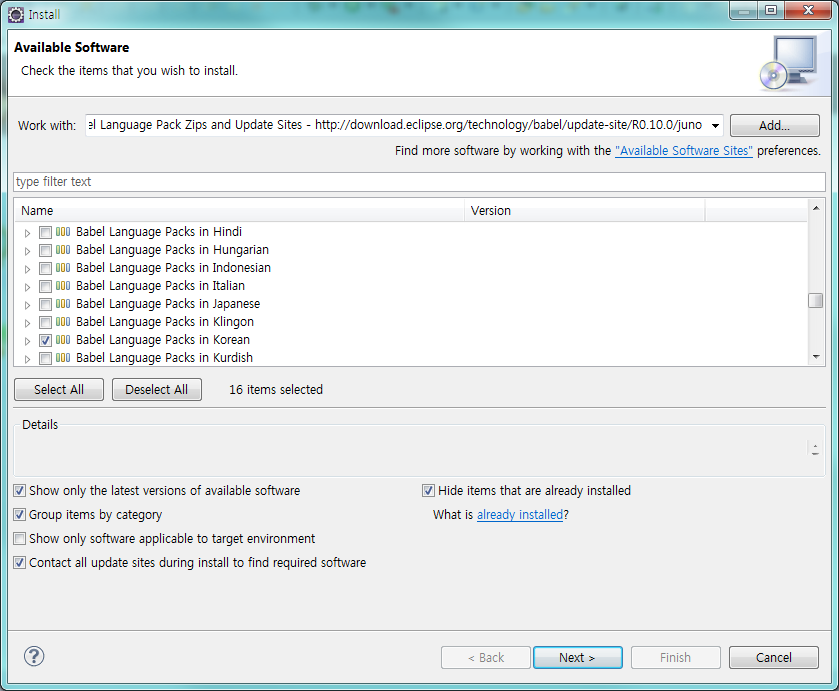


그림 2‑6 한국어 언어 팩 선택 화면

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

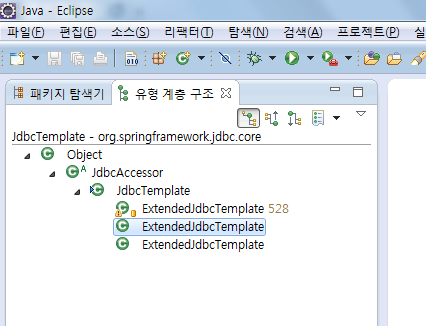


그림 2‑7 언어팩 설치 후 한글이 지원되는 화면

### Subversion

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.

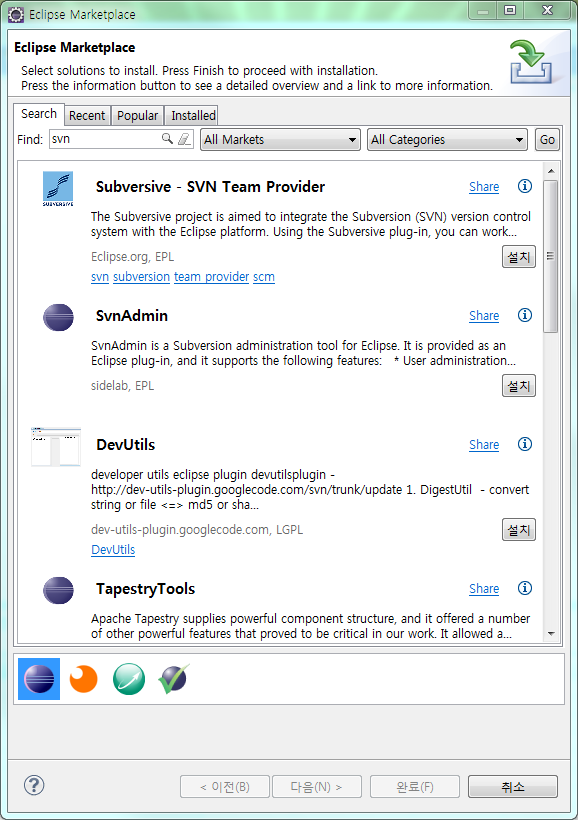


그림 2‑8 Eclipse 에서 Subversion 사용을 위한 플러그인 설치 화면

Eclipse 툴이 없이 Subversion 을 사용하는 경우 윈도우 환경이라면 Tortoise SVN 을 설치하여 사용한다. Tortoise SVN 프로그램은 <http://tortoisesvn.net/downloads.html> 사이트에서 다운로드 할 수 있다. 다운로드 페이지 하단에 위치하는 Language Pack 을 같이 설치하면 한글이 지원된다.

### Maven

빌드 도구로 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.

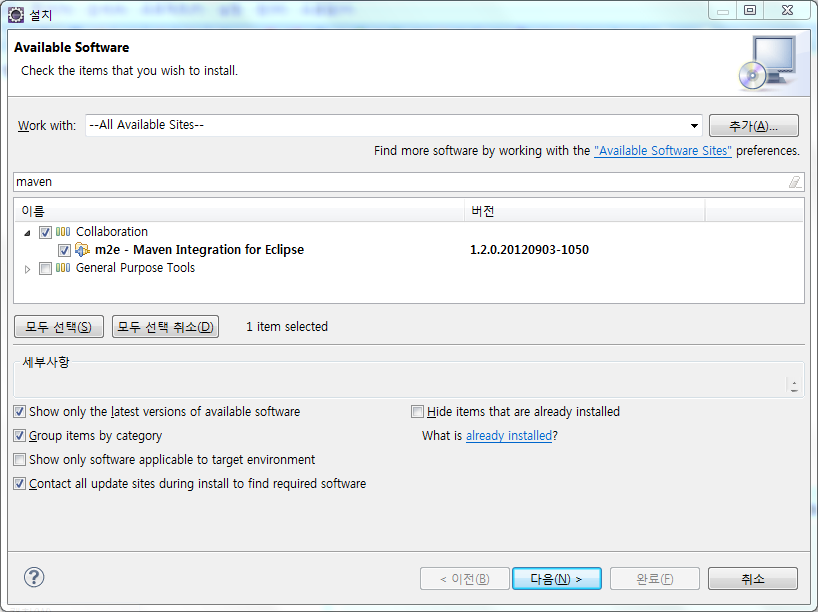


그림 2‑9 Maven 플러그인 설치 화면

또는 도움말(H) >Eclipse Marketplace…… 을 클릭하고 Eclipse Marketplace 창에서 maven 을 검색한다. 검색 결과에서 원하는 소프트웨어를 선택하여 설치한다. 문서에서는 Maven Integration for Eclipse WTP 를 설치하였다.

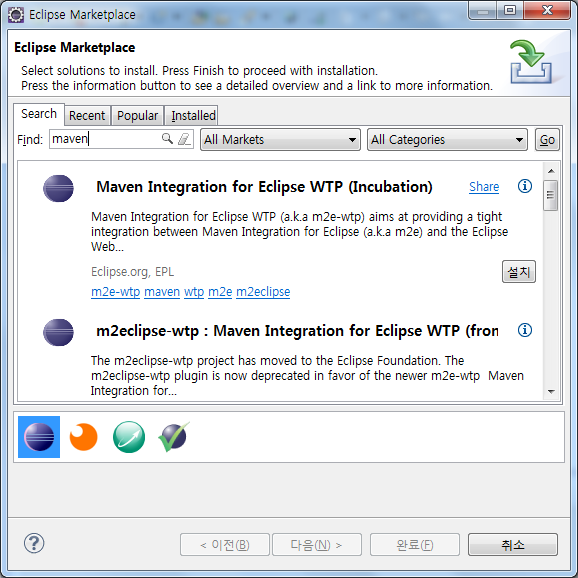


그림 2‑10 Maven 플러그인 설치 화면

## 웹 응용프로그램 설치

ARCHITECTURE 2.0 for JAVA 플랫폼은 손쉬운 웹 응용프로그램 개발을 지원하기 위하여 디폴트 웹 응용프로그램을 제공하고 있다. 개발자는 제공되는 디폴트 웹 응용프로그램을 기반으로 빠르게 목적하는 응용프로그램을 개발할 수 있다.

### 웹 응용프로그램 홈 설정

제공되는 디폴트 웹 어플리케이션은 동작을 위하여 반듯이 설정 파일이 위치하는 될 응용프로그램 홈 경로를 설정해 주어야 한다. 지정하지 않는 경우는 배포된 웹 응용프로그램의 /WEB-INF 경로를 홈으로 사용하도록 되어 있다. 응용프로그램 홈 경로는 아래와 같은 방법들로 설정할 수 있다.

**가. 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* |
| <?xml version="1.0" encoding="UTF-8"?>  <application>  <home>**file://C:\fuse\profile\default**</home>  </application> |

**나. 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> |

### 다국어 지원

다국어 지원을 위하여 요청된 request 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 JEUS 5.0 를 사용하는 경우, 웹 응용프로그램에서 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.

*/WEB-INF/web.xml*

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-container xmlns="http://www.tmaxsoft.com/xml/ns/jeus">  <context-group>  <group-name>ICAP</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> |

## 하드웨어 및 소프트웨어 호환성

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.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NO | H/W | OS | WEB | WAS | JAVA | JDBC | DBMS | 인증 |
| 1 | IBM p780 | IBM AIX 5.3 |  | Oracle Weblogic 10.3 | IBM Java 1.6 |  | Oracle 10g | △[[3]](#footnote-3) |
| 2 | IBM p740 | AIX 6.2 | Apache 2.2.22 | Tomcat 7.0.29 | IBM Java 1.6.0 (64-bit) | OJDBC 11.2.0.3.0 | Oracle 11g (11.2.0.3.0) | ○ |
| 3 |  |  |  | Oracle  Weblogic |  |  | Oracle 10g 64bit RAC (10.2.0.4.0) | △[[4]](#footnote-4) |

|  |  |
| --- | --- |
|  |  |
| 핵심모듈 | |
|  |  |



## Bootstrap Service

ARCHITECTURE EE 2.0 은 다양한 환경에서 개발 및 운용을 위한 설정파일들을 미리 가지고 있으며 최소한의 설정 만으로 웹 어플리케이션을 개발 할 수 있도록 지원하고 있다. ARCHITECTURE 2.0 for JAVA 의 코어 컴포넌트들은 Spring 이 제공하는 컨테이너 기술을 기반으로 하고 있으며 설정의 재사용을 위하여 미리 설정된 spring xml 파일들과 함께 제공된다. 또한 이들 컴포넌트들은 응용프로그램 홈 경로에 위치하고 있는 startup-config.xml 파일에 정의된 정보에 따라 동작하도록 되어 있다.

  
**그림 3‑1 응용프로그램은 startup-config.xml 의 설정에 따라 동작하게 된다**

부스트랩 응용프로그램 컨텍스트를 위한 spring xml (필수 코어 컴포넌트들을 정의하는)설정들은 자동으로 선택되어 로드 되며, 응용프로그램 컨텍스트(Application Context에) 해당하는 spring xml 들은 개발자 또는 운영자가 WEB-INF/context-config/webApplicaitonContext.xml 파일에 정의하여 선택적으로 사용할 수 있다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설정** | **내용** |
| Bootstrap  Context | META-INF/services/default-bootstrap-context.xml | 필수 컴포넌트 정의 |
| META-INF/services/default-plugin-context.xml | 플러그인 컴포넌트 정의 |
| META-INF/services/default-database-context.xml | 데이터베이스 연결 정의 |
| META-INF/services/default-dao-context.xml | DAO 컴포넌트 정의 |
| META-INF/services/default-transaction-context.xml | 로컬 트랜잭션 컴포넌트 정의 |
| Application  Context | context/transactionSubsystemContext.xml | 어노테이션 트랜잭션 컴포넌트 정의 |
| context/oracleSubsystemContext.xml | Oracle LOB 컴포넌트 정의 |
| context/i18nSubsystemContext.xml | 국제화 컴포넌트 정의 |
| context/utilSubsystemContext.xml | 유틸리티 컴포넌트 정의 |

표 3‑1 Bootstrap과 Application 컨텍스트를 구성하는 xml 파일

응용프로그램은 startup-config.xml 설정 정보에 따라 Bootstartp 클래스에의하여 시작되고 종료된다. Bootstrap 클래스가 언제 응용프로그램을 시작하고 종료할 것인가는 서블릿 리스너 기술을 사용하는 WebApplicationContextLoaderListener 클래스 전달되는 이벤트에 의하여 구현된다.

**그림 3‑2 Bootstrap 에 의하여 응용프로그램이 시작되는 것을 보여주는 시퀀스 다이어그램**

## Bootstrap ApplicationProperties

ARCHITECTURE 2.0 플랫폼이 제공하는 핵심 컴포넌트들은 로컬에 저장된 WEB-INF/startup-config.xml 파일의 설정 값에 따라 동작한다. WEB-INF/startup-config.xml 파일에 저장된 데이터는 architecture.common.lifecycle.ApplicationProperties 인터페이스 클래스를 통하여 제어가 가능하다. 다음은 그림은 WEB-INF/startup-config.xml 의 노드 구성을 보여주고 이다.

  
**그림 3‑3 startup-config.xml 구조**

ApplicationProperties 는 다음에 설명하는 architecture.ee.component.admin.AdminHelper 클래스을 사용하여 아래와 같은 방법으로 접근할 수 있다. 참고로 AdminHelper 을 통하여 접근되는 Repository 서비스는 Bootstarp 클래스를 통하여 가장 먼저 초기화되기 때문에 서비스 초기화를 위한 추가 정보가 필요한 경우에 사용된다.

|  |
| --- |
| ApplicationProperties setupProperties = AdminHelper.*getRepository*().getSetupApplicationProperties();  String path = setupProperties.get(  ApplicationConstants.*SCRIPTING\_GROOVY\_SOURCE\_LOCATION\_PROP\_NAME*);  String sourceEncoding = setupProperties.get(  ApplicationConstants.*SCRIPTING\_GROOVY\_SOURCE\_ENCODING\_PROP\_NAME*);  **boolean** recompileGroovySource = setupProperties.getBooleanProperty(  ApplicationConstants.*SCRIPTING\_GROOVY\_SOURCE\_RECOMPILE\_PROP\_NAME*, **false**); |

## ConfigService

architecture.common.lifecycle.ConfigService 는 WEB-INF/startup-config.xml 와 데이터베이스를 기반으로 하여 응용프로그램에서 필요한 다양한 설정들을 관리할 수 있도록 한다. ConfigService 서비스는 architecture.ee.component.admin.AdminHelper 또는 architecture.ee.util.ApplicationHelper 클래스의 getConfigService 함수를 호출하여 접근할 수 있다. ConfigService 는 내부적으로 3가지 종류의 architecture.common.lifecycle.ApplicationProperties 관리하고 있다.



콤포넌트 또는 응용 프로그램의 코드는 ConfigService 을 통하여 제공되는 프로퍼티 정보를 사용하여 코드를 작성할 수 있다.

## ApplicationHelper

Bootstrap 클래스에 의해 시작된 Bootstrap Context 와 Application Context 의 컴포넌트들은 ApplicationHelper 클래스의 getComponent 함수를 이용하여 사용할 수 있다.

* **public** **static** <T> T getComponent (Class<T> requiredType) **throws** ComponentNotFoundException
* **public** **static** <T> T getComponent (String requiredName, Class<T> requiredType) **throws** ComponentNotFoundException

또한 응용프로그램의 상태, License, Locale, TimeZone, Encoding 정보를 제공한다.

* **public** **static** **boolean** isSetupComplete
* **public** **static** **boolean** isReady()
* **public** **static** State getState()
* **public** **static** Locale getLocale()
* **public** **static** String getCharacterEncoding()
* **public** **static** TimeZone getTimeZone()

이 외에도 ApplicationHelper 는 setup-config.xml 로컬 파일과 데이터베이스을 사용하여 프로퍼티 서비스를 제공하는 ConfigService 을 사용하여 응용프로그램의 전반에 사용이 가능한 프로퍼티 서비스에 대한 인터페이스를 제공한다. 이들 함수들은 데이터베이스에 해당하는 값이 존재하지 않는 경우에 startup-config.xml 파일에 존재하는 값을 리턴한다.



그림 3‑4 ConfigService 를 이용한 프로퍼티 서비스

* **public** **static** String getApplicationProperty(String name, String defaultValue)
* **public** **static** **int** getApplicationIntProperty(String name, **int** defaultValue)
* **public** **static** **boolean** getApplicationBooleanProperty(String name, **boolean** defaultValue)

## Logging

ARCHITECTURE EE for JAVA 플랫폼을 구현하는 클래스들은 모두 [Apache Commons Logging API](http://commons.apache.org/logging) (이하 JCL)를 기반으로 로깅을 구현하고 있다. JCL 의 좋은 점은 로깅을 구현하는 라이브러리를 변경하더라도 코드에 변경을 요구하지 않는다는 것이다. 로깅 구현은 log4j 를 사용하고 있다.

### 웹 응용프로그램에서 Log4j 사용하기

Log4j 기술을 웹 응용프로그램에서 사용하기 위하여 스프링 프레임워크가 제공하고 있는 기능을 사용하고 있으며, Log4j 기반 로깅 서비스를 초기화하기 위하여 WEB-INF/web.xml 파일에 다음과 같은 설정을 추가한다. 사용되는 log4j.xml 설정파일은 log4j 설정 규칙을 따른다. [Log4j](http://logging.apache.org/log4j/1.2/) 는 현재 1.2.x 버전을 지원하고 있다.

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"*>  <context-param>  <param-name>log4jConfigLocation</param-name>  Log4j 는 서블릿 리스너 기술을 사용하여  초기화 된다. 컨텍스트 파라메터 값은  Log4j 을 초기화하기 위한 필수 정보이다.  **파라메터 키 정보**   * log4jConfigLocation = log4j 설정파일위치 * webAppRootKey = 웹 응용프로그램의 위치에 대한 알리아스 값 * log4jRefreshInterval = 파일을 모니터링하는 주기   <param-value>WEB-INF/log4j.xml</param-value>  </context-param>  <context-param>  <param-name>webAppRootKey</param-name>  <param-value>default.webapp.root</param-value>  </context-param>  <context-param>  <param-name>log4jRefreshInterval</param-name>  <param-value>120</param-value>  </context-param>  <!-- ============================================================ -->  <!-- Servlet Context Listeners (Executed on app startup/shutdown) -->  <!-- ============================================================ -->  <listener>  <listener-class>  org.springframework.web.util.Log4jConfigListener  </listener-class>  </listener>  </web-app> |

## 국제화 & 지역화

ARCHITECTURE EE for JAVA 플랫폼은 다국어 지원을 위하여 properties 파일 기반과 database 기반의 방식을 제공하고 있다.

### Properties 파일 기반 국제화(I18N)

Porperties 파일을 기반으로 하는 국제화는 ARCHITECTURE EE for JAVA 플랫폼 내부에서 코드 기반의 오류 메시지를 처리하기 위하여 사용되고 있으며 응용프로그램에서 역시 사용이 가능하다. 국제화 지원을 위하여 java.util.ResourceBundle 기술을 사용하고 있다. ResoueceBundel 에서 사용할 Properties 파일은 아래와 같은 형식으로 message + [‘000000’ 형식 코드 값] 키 / 값 형식으로 작성한다. 한국어는 [파일 이름] + “\_” + ko\_KR.properties 형식으로 파일을 생성한다.



*architecture.common.i18n.CommonLogLocalizer.properties*

|  |
| --- |
| version=1.0  message002001=Could not resolve type alias "{0}". Cause\: {1}  message002002=The alias "{0}" is already mapped to the value "{1}".  message002003=Error resolving JdbcType. Cause\: {0} |

*architecture.common.i18n.CommonLogLocalizer\_ko\_KR.properties*

|  |
| --- |
| version=1.0  message002001=별칭 "{0}"을 타입으로 분석할 수 없습니다. 원인\: {1}  message002002=별칭 "{0}" 은 이미 "{1}" 값으로 매핑되어 있습니다.  message002003=JdbcType 을 분석하는 과정에서 오류가 있습니다. 원인\: {0} |

국제화 메시지는 ResourceBundle 을 인자로 생성되는 architecture.common.i18n.Localizer 클래스 통하여 표현된다. Localizer 는 LocalizerFactory 또는 L10NUtils 를 클래스들의 getLocalizer(..) 함수를 사용하여 생성 또는 접근할 수 있다. Localizer 에서 코드 값을 인자로 getMessage(..) 함수를 호출하여 코드에 해당하는 메시지를 리턴 받는다.

자주 사용하는 코드들에 대한 국제화 메시지들은 i18n\_user.xml 파일을 생성하여 편리하게 사용할 수 있다. 리소스 파일이 어떤 코드에 해당하는 메시지들을 포함하는 가를 i18n\_user.xml 파일에 정의한다. (META-INF/i18n.xml 파일은 프레임워크 내부적으로 사용하며 코드 값은 역시 ~ 10000 까지는 프레임워크 모듈에서 사용하고 있다.) 다음 예의 i18n\_user.xml 은 코드 값 2001 ~ 3000 에 해당하는 메시지들은 “*architecture.common.i18n.CommonLogLocalizer”* 을 사용하라는 의미이다.

*i18n\_user.xml*

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <localizers>  <localizer bundleName=*"architecture.common.i18n.CommonLogLocalizer"* codes=*"002001-003000"* />  </localizers> |

이렇게 정의된 메시지들은 architecture.common.util.L10NUtils 의 getMessage(..) 또는 format(..) 함수를 사용하여 ResouceBundle 파일에 대한 정보 없이 코드에서 국제화(I18N) 기능을 구현할 수 있다.

|  |
| --- |
| log.debug( L10NUtils.*format*("002103", "a", "b", "c") );  log.debug( L10NUtils.*getMessage*("002099") ); |



그림 3‑5 L10NUtils

### 데이터베이스 기반 국제화(I18N)

데이터베이스 기반의 국제화는 웹 응용프로그램과 같은 응용프로그램 레벨에서 편리하게 사용할 수 있도록 한다. 오류 메시지에서부터 화면을 구성하는 텍스트 정보까지 다양하게 사용될 수 있다. 응용프로그램에서는 webApplicaitonContext.xml 파일에 다음과 같은 정의를 추가하는 것으로 사용이 가능하다.

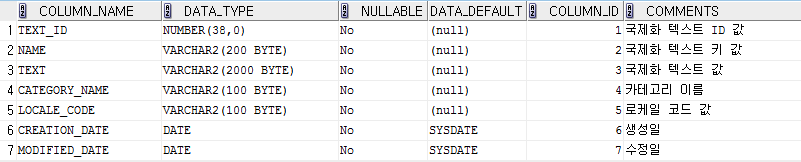
*WEB-INF/context-config/webApplicationContext.xml*

|  |
| --- |
| <?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/i18nSubsystemContext.xml"*/>  <import resource=*"classpath:context/freemarkerSubsystemContext.xml"*/>  <import resource=*"classpath:context/transactionSubsystemContext.xml"*/>  <import resource=*"classpath:context/oracleSubsystemContext.xml"*/>  <import resource=*"classpath:context/utilSubsystemContext.xml"*/>    </beans> |

응용프로그램에서는 architecture.ee.util.ApplicationHelper 의 getI18nTextManager() 함수 호출을 통하여 리턴되는 architecture.common.i18n.I18nTextManager 객체 또는 편리한 사용을 위하여 제공되는 유틸리티 architecture.common.util.I18nTextUtils 클래스를 사용하여 국제화를 구현할 수 있다. 국제화 메시지는 카테고리화 되어 관리되며 아래와 같은 형식으로 사용된다.

|  |
| --- |
| String text = I18nTextUtils.getText("login.buttion.label", "UI", Locale.KOREA ) ; |

이때 국제화 메시지는 데이터베이스의 2\_I18N\_TEXT 에 저장되어 있으며 테이블 구조는 아래와 같다. 편의를 위하여 Excel 또는 XML 파일을 IMPORT 하는 기능을 제공하고 있다. (1.0.5 이후부터 지원)



|  |  |
| --- | --- |
|  |  |
| JDBC 프로그래밍 | |
|  | 대부분의 응용프로그램들은 궁극적으로 비즈니스 정보를 데이터베이스에 저장하게 된다.  이장에서는 데이터베이스 작업에 대하여 알아볼 것이다. |



## 데이터베이스 핸들링

레이어 아키텍처 모델에 따르면 응용프로그램은 논리적 적으로 프레젠테이션, 비즈니스, 퍼시스턴스 레이어들로 구성된다. 이때 데이터베이스 접근이 발생되는 퍼시스턴스 레이어에서는 어떤 퍼시스턴스 프레임워크를 사용할 것인가에 대한 선택이 요구된다. 널리 사용되고 있는 것에는 Hibetnate, Oracle Toplink 그리고 [MyBatis](http://code.google.com/p/mybatis/wiki/Welcome?tm=6) 와 같은 ORM 기술들이 있다.

  
**그림 4‑1 웹 응용프로그램에서 일반적으로 적용되는 레이어드 아키텍처**



그림 4‑2 퍼시스턴스 레이어

그림 4‑3 대부분의 응용프로그램은 관계형 데이터베이스에서 정보를 꺼내고 저장한다. 응용프로그램에서 퍼시스턴스 층는 데이터 접근이 발생하는 곳을 의미한다.

### DAO 스타일

데이터베이스 접근은 Data Access Object[[5]](#footnote-5)(이하 DAO) 패턴을 사용한다. DAO 패턴을 사용하는 목적은 데이터베이스에서 데이터를 읽고 저장하는 것이며, 인터페이스를 통하여 데이터베이스 조작을 허용한다.



  
그림 4‑4 서비스 객체는 DAO 인터페이스를 통하여 DAO 객체에 접근한다.

**그림 4‑2** 에서 서비스 객체들은 DAO 인터페이스를 통하여 데이터베이스에 대한 조작이 허용되고 있는데, 이러한 접근 방식은 아래와 같은 몇 가지 이점이 있다.

첫째 특정 데이터 접근을 위한 구현에 직접적인 결합(coupling)이 이루어 지지 않기 때문에 서비스 객체의 테스트가 용의하게 된다. 만일 데이터 접근 인터페이스에 대한 Mock 구현체를 만들게 되면, 실제 데이터베이스에 대한 연결이 없이 서비스 객체에 대한 테스트가 가능하다.

또한 구체적으로 어떤 기술을 사용하였는가에 대하여 DAO 구현 이외의 로직에서는 관여할 이유가 없어진다. 따라서 퍼시스턴스 레이어와 다른 레이어들간의 결합도를 최소화하여 더욱 유연한 응용프로그램 디자인을 가능하게 한다.

 **그림 4‑5 퍼시스턴스 레이어에서 어떤 기술을 사용하더라도 다른 레이어에 영향을 주지 않는다.**

### SqlQuery

데이터베이스 조작을 위한 많은 프레임워크들이 존재하고 있지만 ARCHITECTURE EE for JAVA 는 전통적인 JDBC 프로그래밍 방식을 개선한 SqlQuery 컴포넌트를 제공하고 있다. SqlQuery 컴포넌트는 소스 코드와 SQL(structured query language)을 분리하여 전통적 JDBC 프로그래밍의 생산성 및 유지보수성을 개선하였으며 템플릿 언어를 지원하여 조건에 따른 동적인 쿼리 생성을 지원하고 있다.

  
**그림 4‑6 SqlQuery 는 xml 정보를 사용하여 쿼리, 입력 파라메터 값 매핑, 결과 데이터 값 매핑 등의 작업을 수행한다.**

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.

또한 Spring 에서 데이터베이스 접근을 위하여 제공되는 클래스들과 상응하는 클래스들을 제공하고 있다.

  
 **그림 4‑7 SqlQuery 지원 클래스와 스프링 클래스 연관도**

## 데이터베이스 연결

### DataSource

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. ARCHITECTURE EE 2.0 for JAVA 플랫폼 기반의 웹 어플리케이션은 WEB-INF/startup-config.xml 파일을 사용하여 응용프로그램에서 JNDI 또는 직접 커넥션 풀링 정보를 정의하여 DataSource 자원을 이용할 수 있다.

  
  
**그림 4‑8 startup-config.xml 파일에서 database 연결은 J2EE서버에 등록된 JNDI 또는 Apache commons dbcp 기반의 connection pool 을 사용할 수 있다.**

startup-config.xml 에서 Apache Commons dbcp 를 이용한 데이터소스 설정 속성은 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 | true일 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. |

**가) JNDI 을 이용한 데이터소스 설정**

WAS 에 등록된 데이터소스에 대한 JNDI 이름을 응용프로그램 홈 경로에 위치한 startup-config.xml 에 아래와 같이 기술하여 사용할 수 있다.

*WEB-INF/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"* /> |

기본적으로 ARCHITECTURE EE 2.0 for JAVA 기반의 응용프로그램은 WEB-INF/startup-config.xml 에 설정을 추가하면 default-database-context.xml 에 설정은 내부적으로 구현되기 때문에 추가로 정의할 필요가 없다.

**나) JNDI 을 이용한 사용자 정의 데이터소스 설정**

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 WEB-INF/context-config/webApplicationContext.xml 파일에 아래의 예제와 과 같이 정의하여 사용할 수 있다.

*WEB-INF/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. -->  새로운 노드 “external” 을 추가하고 JNDI 을 사용하여 DataSource 를 가져오도록 설정하였다.  <jndiDataSourceProvider order="1">  <jndiName>jdbc/DEFAULT\_DS</jndiName>  </jndiDataSourceProvider>  </default>  <external>  <jndiDataSourceProvider order="1">  <jndiName>jdbc/EXTERNAL\_DS</jndiName>  </jndiDataSourceProvider>  </external>  </database>  </startup-config> |

|  |
| --- |
| <bean id=*"dataSource2"* class=*"*architecture.ee.spring.jdbc.support*.*DefaultDataSourceProvider*"*  *WEB-INF/context-config/webApplicationContext.xml*  p:dataSourceName=”external” /> |

**다) 벤더에서 제공하는 클래스를 사용한 데이터소스 설정**

오라클을 사용하는 경우는 커낵션 풀링 기능을 지원하는 oracle.jdbc.pool.OracleDataSource 를 사용한다.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  *WEB-INF/startup-config.xml*  <startup-config>  <setup>  <complete>true</complete>  </setup>  <!-- JDBC DATASOURCE SETTING -->  <database>  <default>  <!-- 1. Use jndi datasource. -->  <oracleDataSourceProvider order=*"1"*>  <driverClassName>oracle.jdbc.pool.OracleDataSource</driverClassName>  <url>jdbc/DEFAULT\_DS</url>  <user>test</user>  <password>test</password>  < connectionCachingEnabled>true</connectionCachingEnabled>  </oracleDataSourceProvider>  </default> |

*default-database-context.xml (내부적으로 처리)*

|  |
| --- |
| <bean id=*"dataSource"* class=*"architecture.common.jdbc.datasource.DataSourceFactory"*  destroy-method=*"close"*  factory-method=*"getDataSource"* /> |

### DBCP

Apache commons-dbcp 플레임워크를 사용하여 직접 spring의 컨텍스트 파일에 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)

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

|  |
| --- |
| <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"  p:driverClassName=""  p:url="###"  p:username="###"  p:password="###" /> |

### JDBC 드라이버

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

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

|  |
| --- |
| <bean id="dataSource"  class="org.springframework.jdbc.datasource.DriverManagerDataSource"  p:driverClassName=""  p:url="###"  p:username="###"  p:password="###" /> |

## SqlQuery

### 쿼리 xml

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **이름** | | | | | **횟수** | **설명** |
| sqlset | | | | | 1 |  |
|  | **name** | | | | 1 | 이름 |
| description | | | | 1 | 설명 |
| vesrion | | | | 1 | 버전 |
| sql-query | | | | 1..n | 쿼리 |
|  | **name** | | |  | 쿼리 이름 |
| description | | |  | 쿼리 설명 |
| dynamic | | | 0..n | 동적 쿼리 문 |
| parameterMappings | | | 0..1 | 파라메터 매핑 정의 |
|  | parameterMapping | | 1..n | 파라메터 매핑 |
|  | index | 1 | 파라메터 인덱스 (1 부터 시작) |
| name | 1 | 파라메터 이름 |
| primary | 0..1 | Primary 키 여부 |
| encoding | 0..1 | 인코딩 값 |
| pattern | 0..1 | 변환 패턴 (javaType 의 옵션값) |
| cipher | 0..1 | 암호화 알고리즘 |
| cipherKey | 0..1 | 암호화 키 |
| chpherKeyAlg | 0..1 | 암호화 키 알고리즘 |
| size | 0..1 | 데이터 길이 |
| javaType | 1 | 자바 형식 |
| jdbcType | 1 | 데이터베이스 데이터 형식 |
| resultMappings | | | 0..1 | 결과 데이터 매핑 정의 |
|  | resultMapping | | 1..n | 결과 데이터 매핑 |
|  | index | 1 | 컬럼 인덱스 (1 부터 시작) |
| name | 1 | 컬럼 이름 |
| primary | 0..1 | Primary 키 여부 |
| encoding | 0..1 | 인코딩 값 |
| pattern | 0..1 | 변환 패턴 (javaType 의 옵션값) |
| cipher | 0..1 | 암호화 알고리즘 |
| cipherKey | 0..1 | 암호화 키 |
| chpherKeyAlg | 0..1 | 암호화 키 알고리즘 |
| size | 0..1 | 데이터 길이 |
| javaType | 1 | 자바 형식 |
| jdbcType | 1 | 데이터베이스 데이터 형식 |



**그림 4‑9 sqlset XML 파일의 노드 구성**

다음은 간단한 sqlset.xml 파일 사용 예이다.

*WEB-INF/sql/custom-sqlset.xml*

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <sqlset>  <name>DEFAULT</name>  <description>Queries for DEFAULT</description>  <version>1.0</version>  <sql-query name=*"SELECT\_ALL\_TABLE\_NAMES"* description=*"전체 테이블 이름"*><![CDATA[  SELECT TABLE\_NAME FROM tabs  ]]></sql-query>  </sqlset> |

custom-sqlset.xml 파일에 정의된 SQL들은 name 노드 값 “DEFAULT” 그룹으로 등록 된다. 주의할 것은 name 값이 “DEFAULT” 인 다른 XML 파일을 동시에 배포한다면 각 파일에 저장된 sql 들은 동일 그룹에 해당하게 된다. 응용프로그램에서는 DEFAULT.SELECT\_ALL\_TABLE\_NAMES 이름으로 쿼리를 사용할 수 있다.

startup-config.xml 에 sql 파일들이 정의된 xml 파일이 위치할 디렉터리를 지정하지 않았다면 디폴트로 웹 응용프로그램이 동작하는 WEB-INF/sql 경로를 검사하고 발견된 XML 파일들을 배포하게 된다. 이를 위하여 webApplicaitonContext.xml 파일에 다음과 같은 정의를 추가한다.

*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/i18nSubsystemContext.xml"*/>  <import resource=*"classpath:context/freemarkerSubsystemContext.xml"*/>  <import resource=*"classpath:context/transactionSubsystemContext.xml"*/>  <import resource=*"classpath:context/oracleSubsystemContext.xml"*/>  <import resource=*"classpath:context/utilSubsystemContext.xml"*/>    </beans> |

utilSubsystemContent.xml 은 xml 파일을 검색하고 모니터링 하여 xml파일에 정의된 쿼리들을SqlQuery 에서 사용할 수 있도록 미리 정의된 설정 파일이다.

### DAO 구현하기

SqlQuery 기술 기반의 DAO 구현하는 가장 손쉬운 방법은 기본적으로 제공하는 유틸리티 클래스architecture.ee.spring.jdbc.support.SqlQueryDaoSupport 을 상속하는 것이다.

  
 **그림 4‑10 응용프로그램 DAO 와 DAO 지원 클래스 연관도**

DAO 구현 클래스는 SqlQueryDaoSupport 을 상속하고, 데이터베이스 작업을 위하여 getSqlQuery() 함수를 호출하고 SqlQuery 객체를 사용하여 작업한다. SqlQueryDaoSupport 는 전통적인 JDBC 프로그래밍을 위한 인터페이스 역시 제공하고 있다. 예를 들어 getDataSource(), getConnection() 함수를 사용하여 직접 작업을 할 수 도 있다.

만일 SqlQuery 를 사용하지 않고 Springframework 의 JdbcTemplate 와 xml 에 정의된 쿼리 만들 사용하고자 한다면 architecture.ee.spring.jdbc.support.ExtendedJdbcDaoSupport 클래스 상속하여 DAO 를 구현하면된다. 퀴리는 getBoundSql (..) 함수를 사용하여 꺼낼 수 있다.

|  |
| --- |
| package architecture.ee.i18n.dao.impl;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Types;  import java.util.Date;  import java.util.List;  import java.util.Locale;  import org.springframework.dao.IncorrectResultSizeDataAccessException;  import org.springframework.jdbc.core.RowMapper;  import architecture.ee.i18n.I18nLocale;  import architecture.ee.i18n.dao.I18nLocaleDao;  import architecture.ee.model.impl.I18nLocaleModelImpl;  import architecture.ee.spring.jdbc.support.ExtendedJdbcDaoSupport;  **public** **class** JdbcI18nLocaleDao extends ExtendedJdbcDaoSupport implements I18nLocaleDao {  **class** I18nLocaleRowMapper implements RowMapper<I18nLocale> {  **public** I18nLocale mapRow(ResultSet rs, int rowNum) throws SQLException {  I18nLocaleModelImpl c = new I18nLocaleModelImpl();  c.setLocaleId(rs.getLong(1));  c.setLanguage(rs.getString(2));  c.setCountry(rs.getString(3));  c.setVariant(rs.getString(4));  c.setEncoding(rs.getString(5));  c.setCreationDate(rs.getDate(6));  c.setModifiedDate(rs.getDate(7));  **return** (I18nLocale)c;  }};    @Override  **protected** void initDao() throws Exception {  super.initDao();  }  **public** int getAvailableI18nLocaleCount() {  **return** getExtendedJdbcTemplate().queryForInt(  getBoundSql("ARCHITECTURE\_FRAMEWORK.COUNT\_ALL\_LOCALE").getSql());  }  **public** List<I18nLocale> getAvailableI18nLocales() {  **return** getExtendedJdbcTemplate().query(  getBoundSql("ARCHITECTURE\_FRAMEWORK.SELECT\_ALL\_LOCALE").getSql(), new I18nLocaleRowMapper() );  }  **public** I18nLocale getI18nLocaleById(long localeId) {  **return** getExtendedJdbcTemplate().queryForObject(  getBoundSql("ARCHITECTURE\_FRAMEWORK.SELECT\_LOCALE\_BY\_ID").getSql(),  new Object[]{localeId}, new int[]{Types.INTEGER}, new I18nLocaleRowMapper());  }  **public** I18nLocale getI18nLocale(Locale locale, boolean createIfNotExist) {  try {  **return** getI18nLocale(locale);  } catch (IncorrectResultSizeDataAccessException e) {  if(createIfNotExist){  **return** addI18nLocale(locale);  }  throw e;  }  }  **public** I18nLocale getI18nLocale(Locale locale) {  **return** getExtendedJdbcTemplate().queryForObject(  getBoundSql("ARCHITECTURE\_FRAMEWORK.SELECT\_LOCALE\_BY\_CODE").getSql(),  new Object[]{locale.getLanguage(), locale.getCountry() },  new int[]{Types.VARCHAR, Types.VARCHAR}, new I18nLocaleRowMapper());  }  **public** I18nLocale addI18nLocale(Locale locale) {  long now = System.currentTimeMillis();  I18nLocaleModelImpl impl = new I18nLocaleModelImpl();  impl.setLocaleId(getNextId("I18N\_LOCALE"));  impl.setLanguage(locale.getLanguage());  impl.setCountry(locale.getCountry());  impl.setVariant(locale.getVariant());  //impl.setEncoding(defaultEncoding);  impl.setCreationDate(new Date(now));  impl.setModifiedDate(new Date(now));  getExtendedJdbcTemplate().update(getBoundSql("ARCHITECTURE\_FRAMEWORK.INSERT\_LOCALE").getSql(),  new Object[]{  impl.getLocaleId(),  impl.getLanguage(),  impl.getCountry(),  impl.getVariant(),  new java.sql.Date(now),  new java.sql.Date(now)},  new int [] {Types.INTEGER, Types.VARCHAR, Types.VARCHAR, Types.VARCHAR, Types.DATE, Types.DATE}  );  **return** (I18nLocale)impl;  }  **public** void deleteI18nLocale(I18nLocale locale) {  getExtendedJdbcTemplate().update(  getBoundSql("ARCHITECTURE\_FRAMEWORK.DELETE\_LOCALE\_BY\_ID").getSql(),  new Object[]{locale.getLocaleId()}, new int[]{Types.INTEGER});  }  **public** I18nLocale updateI18nLocale(I18nLocale locale) {  java.sql.Date d = new java.sql.Date(System.currentTimeMillis());  locale.setModifiedDate(d);  getExtendedJdbcTemplate().update(getBoundSql("ARCHITECTURE\_FRAMEWORK.UPDATE\_LOCALE").getSql(),  new Object[]{locale.getEncoding(), d, locale.getLocaleId()},  new int []{ Types.VARCHAR, Types.DATE, Types.INTEGER } );  **return** locale;  }  } |

### 클래식 스타일 API

파라메터 배열과 JDBC TYPE 배열을 인자로 하는 queryForList() 함수를 사용하면 쉽게 등록된 쿼리에 대한 결과를 List<Map<String, Object>> 형식으로 리턴 받을 수 있다. (JDBC TYPE 값은 데이터베이스의 데이터타입을 표현하는 java.sql.Types 값을 의미한다)



쿼리 실행 결과가 String, Long, Integer 로 표현이 가능한 경우에는 해당하는 자바 타입의 List 객체로 결과를 받을 수 있다.



*TestDao.java*

|  |
| --- |
| **package** tests.dao;  **import** java.util.List;  **import** java.util.Map;  **public** **interface** TestDao {  **public** List<Map<String, Object>> queryForList(String statement );  **public** List<Map<String, Object>> queryForList(String statement , Object[] params, **int**[] jdbcTypes);  **public** List<String> queryForStringList(String statement );  **public** List<String> queryForStringList(String statement , Object[] params, **int**[] jdbcTypes);  } |

*TestDaoImpl.java*

|  |
| --- |
| **package** tests.dao.impl;  **import** java.util.List;  **import** java.util.Map;  **import** tests.dao.TestDao;  **import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;  **public** **class** TestDaoImpl **extends** SqlQueryDaoSupport **implements** TestDao {  @Override  **protected** **void** initDao() **throws** Exception {  // DAO 객체를 초기화할 때 함께 작업할 내용이 있다면 여기에 코드를 삽입한다.  **super**.initDao();  }  **public** List<Map<String, Object>> queryForList(String statement) {  **return** getSqlQuery().queryForList(statement);  }  **public** List<Map<String, Object>> queryForList(String statement, Object[] params, **int**[] jdbcTypes) {  **return** getSqlQuery().queryForList(statement, params, jdbcTypes);  }  **public** List<String> queryForStringList(String statement) {  **return** getSqlQuery().queryForList(statement, String.**class**);  }  **public** List<String> queryForStringList(String statement, Object[] params, **int**[] jdbcTypes) {  **return** getSqlQuery().queryForList(statement, params, jdbcTypes, String.**class**);  }  } |

쿼리 결과가 단일 열인 경우에는 queryForMap() 함수를 사용하면 쿼리 결과를 자바의 Map<String, Object) 형으로 리턴 받을 수 있다. 주의 할 점은 쿼리 결과는 반듯이 하나의 열에 해당하는 값을 가지고 있어야 한다는 점이다.



쿼리 실행 결과가 String, Long, Integer 로 표현이 가능한 경우에는 해당하는 자바 타입의 객체로 결과를 받을 수 있다.



데이터베이스에 새로운 값을 넣거나 수정하거나 삭제하는 작업은 update 함수를 사용하며 결과 값으로 처리된 열 수를 리턴한다.



### 배치 작업

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

*TestDaoImpl.java*

|  |
| --- |
| **package** tests.dao.impl;  **import** java.util.List;  **import** java.util.Map;  **import** tests.dao.TestDao;  **import** architecture.ee.jdbc.sqlquery.SqlQueryHelper;  **import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;  **public** **class** TestDaoImpl **extends** SqlQueryDaoSupport **implements** TestDao {  @Override  **protected** **void** initDao() **throws** Exception {  // DAO 객체를 초기화할 때 함께 작업할 내용이 있다면 여기에 코드를 삽입한다.  **super**.initDao();  }    **public** Object batchUpdate(String statement, List<Object[]> parameters) {  **return** getSqlQuery().batchUpdate(statement, parameters);  }  **public** Object batchUpdate(String statement, List<Object[]> parameters, **int**[] jdbcTypes) {    SqlQueryHelper helper = **new** SqlQueryHelper();  **for**( Object[] params : parameters ){  파라메터를 추가한 다음에 반듯이 inqueue() 함수를 실행한다.  helper.parameters(params, jdbcTypes).inqueue();  }  **return** helper.executeBatchUpdate( getSqlQuery() , statement );  }  } |

### 

### LOB 데이터

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

1. webApplicationContext.xml을 수정하여 LOB 조작을 위한 객체를 생성한다.
2. 오라클을 사용하는 경우는 oracleSubsystemcontext.xml을 사용한다.
3. DAO 구현체를 정의하는 XML을 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>  오라클 LOB 데이터 조작이 가능하도록 LOB 관련 컴포넌트를 정의하는 xml 파일을 추가한다. |

*WEB-INF/context-config/testSubsystemContext.xml*

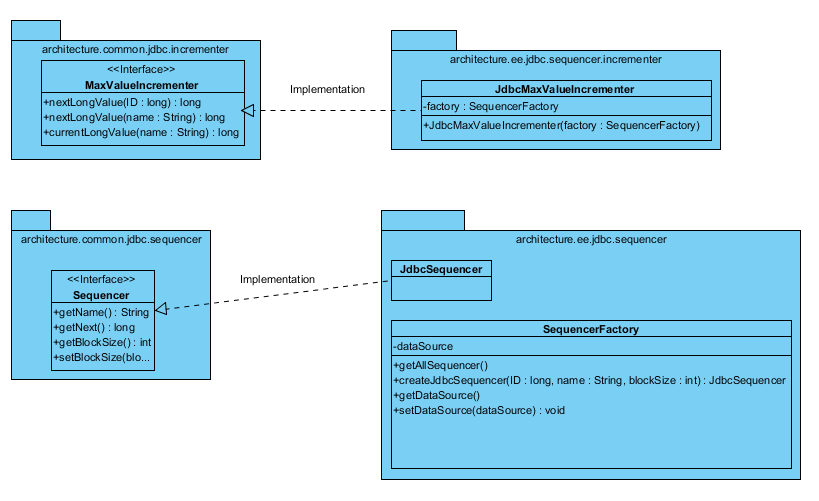
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:c=*"http://www.springframework.org/schema/c"* xmlns:aop=*"http://www.springframework.org/schema/aop"*  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*  서비스 객체에서 LOB 데이터 조작이 가능하게 설정을 추가한다.  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd"*>  <bean id=*"testDao"* class=*"tests.dao.impl.TestDaoImpl"*  parent=*"sqlQuerySupport"* p:dataSource-ref=*"dataSource"*  lobHandler-ref=*"lobHandler"*/>  </beans> |

*TestDaoImpl.java*

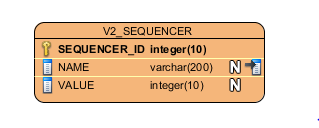
|  |
| --- |
| **package** tests.dao.impl;  **import** java.io.File;  **import** java.io.IOException;  **import** java.sql.Types;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.UUID;  **import** tests.dao.TestDao;  **import** architecture.ee.jdbc.sqlquery.SqlQueryHelper;  **import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;  **public** **class** TestDaoImpl **extends** SqlQueryDaoSupport **implements** TestDao {  lob() 함수를 사용하여 CLOB, BLOB 데이터를 처리한다. 파라메터 값이 String 인 경우는 CLOB 로 바이너리 데이터인 경우는 BLOB로 처리된다.  @Override  **protected** **void** initDao() **throws** Exception {  // DAO 객체를 초기화할 때 함께 작업할 내용이 있다면 여기에 코드를 삽입한다.  **super**.initDao();  }  **public** String save(String statement, File file) **throws** IOException {  UUID uuid = UUID.*randomUUID*();  SqlQueryHelper helper = **new** SqlQueryHelper(getLobHandler());  helper.parameter(uuid.toString());  helper.lob(file);  getSqlQuery().update( statement, helper.values(), **new** **int**[] {Types.*VARCHAR*, Types.*BLOB*});  **return** uuid.toString() ;  }  } |

### 유니크 아이디 값 생성

오라클의 Sequence 와 유사하게 number 형태의 유니크 한 값을 생성하는 기능을 제공하고 있어 코딩 시에 손쉽게 유일한 ID 값 생성이 가능하다. 또한 내부적으로 캐시(Cache) 기능을 가지고 있기 때문에 성능에서 역시 유리한 점이 있다.



사용의 편의성을 위하여 SqlQueryDaoSupport 클래스의 getMaxValueIncrementer() 함수를 사용하여 MaxValueIncrementer 객체에 접근하고 인자에 해당하는 최대 값을 리턴 받는다. 데이터베이스에는 V\_SEQUENCER 테이블을 통하여 유니크 값이 관리된다.



### 페이징

setStartIndex() 와 setMaxResults() 함수를 사용하면 사용하는 데이터베이스에 따른 별도의 처리 없이 쉽게 페이지 처리가 가능하다. setStartIndex 값은 쿼리 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 MaxResults 값을 6 으로 하면 index 값이 19 에 해당하는 위치에서부터 6개의 데이터를 리턴하게 된다. 이러한 원칙을 이용하면 손쉬운 페이징 구현이 가능할 것이다.

  
그림 4‑11 startIndex 부터 시작하여 MaxResults에 해당하는 행까지 데이터를 리턴한다.

*TestDaoImpl.java*

|  |
| --- |
| **package** tests.dao.impl;  **import** java.io.File;  **import** java.io.IOException;  **import** java.sql.Types;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.UUID;  **import** tests.dao.TestDao;  **import** architecture.ee.jdbc.sqlquery.SqlQueryHelper;  **import** architecture.ee.spring.jdbc.support.SqlQueryDaoSupport;  **public** **class** TestDaoImpl **extends** SqlQueryDaoSupport **implements** TestDao {  다른 queryForList() 함수에서 역시 동일하게 처리하면 페이지 처리가 된다.  @Override  **protected** **void** initDao() **throws** Exception {  // DAO 객체를 초기화할 때 함께 작업할 내용이 있다면 여기에 코드를 삽입한다.  **super**.initDao();  }  **public** List<String> queryForStringList(String statement, **int** startIndex, **int** maxResults) {  **return** getSqlQuery().setStartIndex(startIndex).setMaxResults(maxResults).queryForList(statement, String.**class**);  }  } |

### 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, "DEFAULT.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  이 경우 다음 SQL 이 동적으로 생성되어 실행된다. V2\_I18N\_LOCALE WHERE LOCALE\_ID <> 0  </#if>  ]]></dynamic>  </sql-query> |

### 작업단위패턴(Unit Of Work)

작업단위패턴을 스크립트 언어인 Groovy 와 함께 사용될 수 있도록 지원하고 있다. 이를 위하여 UnitOfWorkForSqlQuery 또는 SqlQueryCallback 인터페이스를 구현하는 Groovy 파일을 생성하고 이를 호출하여 사용할 수 있다.

### 새로운 스타일 API

스프링의 API 형식과 다른 새로운 스타일의 API 을 사용하는 경우, 입력 파라메터와 출력 결과를 xml 에 정의된 매핑에 따라 변환하는 기능들을 사용할 수 있다.

### 파라메터 매핑

파라메터 매핑은 쿼리 XML 에 정의된 쿼리를 실행할 때 입력된 파라메터 값들을 어떻게 처리할 것인가를 정의하는 것을 의미하며 <parameterMapping> 태크를 사용하여 정의한다.

## SqlQueryClient 사용하기

DAO 패턴을 사용하게 되면 기본적으로 비즈니스 로직을 구현하는 서비스 객체를 구현하기 위하여 최소 4개의 소스파일을 작성하여야 한다. architecture.ee.services.SqlQueryClient 는 일반적인 데이터베이스 핸들링을 위한 기능들을 제공하는 DAO 구현체로 서비스 객체에서 DAO 생성 없이 간편하게 사용할 수 있다.

  
**그림 4‑12 DAO 패턴을 적용하는 경우 비즈니스 로직이 필요로 하는 모든 데이터베이스 핸들링 작업을 수행하는 DAO를 구현하여야 한다.**

  
**그림 4‑13 서비스 객체는 이미 구현되어 있는 DAO 인 SqlQueryClient 을 사용하여 비즈니스 로직이 필요로 하는 모든 데이터베이스 핸들링 작업을 처리할 수 있다.**

또한 동적 스크립트로 작성된 프로그램을 단일 트랜잭션 단위로 실행할 수 있기 때문에 비즈니스 요구사항에 따라 여러 가지 데이터 작업을 단일 트랜잭션으로 처리할 수 있다.

### 기본 API

SqlQueryClient 는 단일 로(row)를 조회하는 경우는 uniqueResult(), 다수의 로를 조회하는 경우는 list() 함수를 사용한다.

### 콜백함수

일반적으로 DAO 객체를 구현하는 경우 단일 쿼리의 실행 결과가 아닌 여러 쿼리 결과와 데이터 조작이 요구되는 경우가 있다. SqlQueryClient 는 이러한 작업 수행을 위하여 SqlQueryCallback 클래스을 인자로 하는 execute 함수를 제공하고 있다. 개발자는 아래와 같이 서비스 클래스에서 SqlQueryCallback 클래스를 생성하고 SqlQuery 객체를 인자로 받는 doInSqlQuery 함수를 작성하여 사용할 수 있다.

|  |
| --- |
| SqlQueryClient client = getComponent(SqlQueryClient.**class**);  사용자가 SqlQueryCallback 클래스를 구현하여 필요한 작업을 처리하도록 한다.  client.execute(**new** SqlQueryCallback<List<Map<String, Object>>> (){  **public** List<Map<String, Object>> doInSqlQuery(SqlQuery sqlQuery) {  // sqlQuery 를 사용하여 필요한 작업을 수행한다.  **return** **null**;  }  }  ); |

### 단일 트랜젝션 처리 unitOfWork

SqlQueryClient 의 가장 큰 특징은 비즈니스 로직을 포함하는 스크립트를 단일 트랜잭션 단위로 처리하는 기능이다. 특히 동적 스크립트 언어인 Groovy 기반의 스크립트를 지원하여 코드의 컴파일 또는 서버의 재 시작 없이 동적으로 수정된 코드를 실행할 수 있다. 참고로 이 함수는 단일 트랜잭션 단위로 실행되기 때문에 트랜잭션이 선언된 서비스 함수에서 호출하는 경우는 unitOfWorkWithoutTransaction() 함수를 호출하여야 한다.

**public** Object unitOfWorkWithoutTransaction ( String scriptName, String methodName , Object... parameters );

**public** Object unitOfWork ( String scriptName , String methodName , Object... parameters );

호출할 스크립트 내부의 함수

함수 호출시 전달할 파라메터 값

호출할 스크립트 파일

**그림 4‑14 스크립트를 이용한 SqlQueryClient**

*jdbc-test.jsp*

|  |
| --- |
| <%@ page contentType=*"text/html; charset=utf-8"* pageEncoding=*"utf-8"* %>  <%@ page import=*"architecture.ee.services.\*,architecture.ee.web.util.WebApplicationHelper"*%>  <html>  <head>  <title>unitOfWork 테스트</title>  </head>  스크립트 이름, 호출할 함수 이름, 그리고 호출 함수 파라메터를 인자로 unitOfWork 를 호출한다  <body>  <p>unitOfWork 테스트</p>  <%  SqlQueryClient client = WebApplicationHelper.getComponent(SqlQueryClient.**class**) ;  %>  <%= client.unitOfWork("unitofwork.SqlQueryTest", "getTableNames", "V%") %>  </body>  </html> |

스크립트 파일을 명시적으로 지정하지 않으면 디폴트로 WEB-INF/groovy 경로를 검색하도록 되어 있다. 스크립트는 수정의 경우에 동적으로 반영되며 그 외의 경우는 서버를 재 기동하여야 한다. 스크립트는 자바 언어와 호환되며 아래와 같이 UnitOfWorkForSqlQuery 클래스을 확장하여 사용한다.

*WEB-INF/groovy/unitofwork/SqlQueryTest.groovy*

|  |
| --- |
| package unitofwork  import groovy.io.FileType  import architecture.ee.services.support.UnitOfWorkForSqlQuery  import architecture.ee.jdbc.sqlquery.\*;  class SqlQueryTest extends UnitOfWorkForSqlQuery {    def getTableNames(String tableName){  println( "hello what the hell !!" );  println( "getTableNames called with params : " + tableName )  return sqlQuery.list("BUSINESS.SELECT\_TABLENAMES\_LIKE\_NAME", tableName)  }    def getTableData(String tableName){  SqlQueryHelper helper = new SqlQueryHelper();  helper.additionalParameter("TABLE\_NAME", tableName);  List<Map<String, Object>> list = helper.list( sqlQuery, "BUSINESS.SELECT\_TABLE");  return list;  }  } |

## Springframework 기술 기반 트랜잭션 관리

Springframework 는 트랜잭션을 지원하기 위하여 다음 3개의 인터페이스를 사용하고 있다.

* PlatformTransactionManager : 실질적으로 트랜잭션을 수행하는 인터페이스
* TransactionDefinition : 트랜젝션의 네가지 속성[[6]](#footnote-6)을 제어하기 위한 인터페이스.
* TransactionStatus : 트랜잭션의 상태를 관리하는 인터페이스.

**격리 수준(Isolation Level)**

트랜잭션 격리 수준에는 다음이 있다.

|  |  |
| --- | --- |
| 속성 | 설명 |
| ISOLATION\_DEFAUTL | 디폴트 격리 수준으로 Commit 되지 않는 데이터를 읽을 수 없다. |
| ISOLATION\_READ\_COMMITTED | 대부분의 데이터베이스들이 디폴트로 지원하는 격리 레벨이다. 이 격리수준은 다른 트랜잭션에서 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. |
| ISOLATION\_READ\_UNCOMMITTED | 가장 낮은 격리수준으로 다른 트랜잭션에서 commit 하지 않은 변경데이터를 읽을 수 있다. 다른 트랜잭션들간 잠금/해제가 일어나지 않기 때문에 데이터 일관성이 보장되지 않는다. |
| ISOLATION\_REPEATABLE\_READ | 다소 엄격한 격리수준으로 다른 트랜잭션에서 commit 하지 않은 변경데이터를 읽을 수 없다. 동일 객체를 여러 번 조회하는 경우 같은 값을 읽는 것을 보장한다. |
| ISOLATION\_SERIALIZABLE | 가장 신뢰할 만한 격리수준으로 어떤 간섭도 허용하지 않는다. 잠금/해제로 인한 과부하가 높으나 데이터의 일관성을 보장한다. 트랜잭션이 순차적으로 실행되는 것처럼 지원한다. |

표 5‑2 트랜잭션 격리수준(ISOLATION) 속성

**전달행위(Propagation Behavior)**

트랜잭션 전달행위에는 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 commit, 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. |

표 5‑3 트랜잭션 PROPAGATION 속성

### 프로그램적으로 트랜잭션 처리

앞에서 언급된 인터페이스들을 사용하여 다음 예와 간이 프로그램적으로 구현할 수 있다. 직접 org.springframework.transaction.PlatformTransactionManager 인터페이스를 사용하여 직접 트랜잭션을 관리할 수도 있으나 TransactionTemplate 을 사용하여 간편하게 트랜잭션을 관리할 수 있다.

TransactionTemplate 을 이용한 트랜잭션 관리를 위하여 개발자는 트랜잭션 처리할 실행 코드들을 모두 포함하고 있는 TransactionCallback 를 구현하고 TransactionTemplate 의 execute() 함수의 인자로 호출하면 된다.

*SqlQueryService.java*

|  |
| --- |
| **public** **class** SqlQueryService **extends** SqlQueryDaoSupport {  **private** **final** TransactionTemplate transactionTemplate;  **public** SqlQueryService(PlatformTransactionManager transactionManager) {  **this**.transactionTemplate = **new** TransactionTemplate(transactionManager);  }  **public** Object someServiceMethod() {  **return** transactionTemplate.execute(**new** TransactionCallback<List<Map<String, Object>>>() {  **public** List<Map<String, Object>> doInTransaction(TransactionStatus status) {  **return** resultOfSelectOperation();  }  });  }  **public** List<Map<String, Object>> resultOfSelectOperation(){  SqlQueryHelper helper = **new** SqlQueryHelper();  helper.additionalParameter("TABLE\_NAME", "V2\_I18N\_LOCALE");  List<Map<String, Object>> list = helper.list(getSqlQuery(), "COMMON.SELECT\_TABLE\_ROWS");  **return** list;  }  } |

또한 리턴 값이 없는 경우는 제공되는 TransactionCallbackWithoutResult 클래스를 사용한다.

|  |
| --- |
| **return** transactionTemplate.execute(**new** TransactionCallbackWithoutResult () {  **public** List<Map<String, Object>> doInTransactionWithoutResult(TransactionStatus status) {  updateOperation();  }  }); |

콜백(Callback)[[7]](#footnote-7) 코드에서 TransactionStatus 객체가 제공하는 setRollbackOnly() 함수를 호출하여 트랜잭션 롤백처리가 가능하다.

|  |
| --- |
| **return** transactionTemplate.execute(**new** TransactionCallbackWithoutResult () {  **public** List<Map<String, Object>> doInTransactionWithoutResult(TransactionStatus status) {  **try{**  updateOperation();  } **catch** (Exception e){  status.setRollbackOnly();  }  }  }); |

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 자바5 이후 지원하는 어노테이션을 사용하는 경우는 많은 자원을 사용하기 때문이다.

### 선언형 트랜잭션관리

### 어노테이션을 통한 관리

선언형 트랜젝션 관리 기능을 사용하기 위하여 XML 파일에 트랜잭션 설정을 기술하는 방법 이외에 자바 5 이후부터 지원되는 어노테이션(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** java.util.Map;  **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 } )

Transaction 의 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) |

표 5‑1 Transaction 속성

|  |  |
| --- | --- |
|  |  |
| 1. 서비스 | |
|  |  |



## 서비스 구현



서비스에서 데이터베이스 조작은 DAO 객체를 사용하거나 사용되었던 SqlQueryDaoSupport 클래스를 상속하여 구현한다.

## 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.

## 예외처리

서비스에서 예외처리는 에러코드 기반의 예외 처리 패턴을 따르며, 손쉬운 구현을 위하여ApplicationException 클래스 와 Codeable 인터페이스를 사용한다. 에러코드는 반듯이 60000 번 이상을 사용한다. 0 ~ 59999 오류 코드는 ARCHITECTURE EE 2.0 for JAVA 플랫폼 시스템 내부용으로 예약되어 있다. 다음은 이미 구현되어 있는 ApplicationException 클래스를 사용한 예이다.

*tests.service.impl.DataServiceImpl.java*

|  |
| --- |
| **package** tests.service.impl;  **import** java.util.List;  **import** java.util.Map;  **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> 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);  }  }  } |

다음은 Action 에서 Codeable 인터페이스를 구현하는 예외클래스를 사용하여 오류를 처리하는 예이다.

*tests.service.impl.DataServiceImpl.java*

|  |
| --- |
| **package** tests.service.impl;  **import** java.util.List;  **import** java.util.Map;  **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);  }  }  } |

예외 클래스는 Codeable 인터페이스를 직접 구현하거나 ApplicationException 을 상속하여 구현할 수 있다.

*tests.service.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);  }  } |

C:\Users\donghyuck\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\5ZB9TKGX\MC900237453[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.

|  |  |
| --- | --- |
|  |  |
| 1. 통합 | |
|  |  |

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 [↑](#footnote-ref-2)
3. 이슈: 자바 Class Loader가 create와 같은 특정 이름의 함수를 검색하지 못함. 리턴 값이 숫자가 아닌 프로시저 호출에 오류가 발생. [↑](#footnote-ref-3)
4. 이슈: 스트럿츠1 관련 web.xml 에서 설정 파일을 기술할 때 공백 없이 기술하여야 함. [↑](#footnote-ref-4)
5. DAO 패턴은 표준 J2EE 디자인 패턴들 중 하나이며, 이 패턴을 사용하여 저 수준 데이터 액세스와 고급 비즈니스 로직을 분리할 수 있다 [↑](#footnote-ref-5)
6. 트랜잭션은 Atomicity(원자성), Consistency(일관성), Isolation(격리성), Durability(지속성) 네 가지 속성을 갖는다. [↑](#footnote-ref-6)
7. **콜백**(callback)은 다른 코드의 인수로서 넘겨받는 서브루틴이다. 이를 통해 높은 수준의 층에 정의된 서브루틴(또는 함수)을 낮은 수준의 추상화 층이 호출할 수 있게 된다. 일반적으로 먼저 높은 수준의 코드가 낮은 수준의 코드에 있는 함수를 호출할 때, 다른 함수의 포인터나 핸들을 넘겨준다. 낮은 수준의 함수를 실행하는 동안에 그 넘겨받은 함수를 적당히 회수, 호출하고, 부분 작업을 실행하는 경우도 있다. 다른 방식으로는 낮은 수준의 함수는 넘겨받은 함수를 '핸들러'로서 등록하고, 낮은 수준의 층에서 비동기적으로(어떠한 반응의 일부로서) 다음에 호출하는데 사용한다. 콜백은 폴리모피즘과 제네릭프로그래밍의 단순화된 대체 수법이며, 어떤 함수의 정확한 동작은 그 낮은 수준의 함수에 넘겨주는 함수 포인터(핸들러)에 의해 바뀐다. 이것은 코드 재사용을 하는 매우 강력한 기법이라고 말할 수 있다. [↑](#footnote-ref-7)