Real Alternative DBMS ALTIBASE, Since 1999

# Spring 연동 가이드

2014. 10

# Document Control

## Change Record

| Date | Author | Change Reference |
|---|---|---|
| 2010-08 | snkim | Created |
| 2014-10 | Dbpark | Updated |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## Reviews

| Date | Name (Position) |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

## Distribution

| Name | Location |
|---|---|
| | |
| | |
| | |
| | |

# 목차

# 개요

본 문서는 Spring에서 ALTIBASE와 연동하는 방법에 대해 기술한다.

Spring Framework 3.2.x, ALTIBASE는 6.3.1, 개발 IDE로는 Eclipse를 사용하고, 문서 이외에 각 chapter 별로 예제가 제공되므로 해당 예제를 참고하면 된다.

본 문서와 더불어 개발 시 참고해야 할 문서들은 다음과 같다.

1. 『ALTIBASE 개발가이드』
2. 『JAVA 개발가이드』
3. 『ALTIBASE_JBOSS 연동가이드』
4. 『ALTIBASE_TOMCAT 연동가이드』
5. 『ALTIBASE_WEBSPERE 연동가이드』
6. 『ALTIBASE_WEBLOGIC 연동가이드』
7. 『ALTIBASE_iBATIS 연동가이드』
8. 『ALTIBASE_HIBERNATE 연동가이드』

# Spring이란?

Spring Framework은 Enterprise Application에서 필요로 하는 기능들을 제공하는 오픈 소스 프레임워크이다. EJB 컨테이너에 비해 lightweight한 컨테이너로서 POJO(Plain Old Java Object: 기존의 일반 자바 객체)의 생성, 소멸 등 라이프사이클을 관리해주며 트랜잭션 관리, 로깅, 보안 등과 같은 모듈을 제공한다.

## Spring의 특징

Spring은 다음과 같은 특징이 있다.

1. J2EE에 비해 lightweight한 컨테이너이다.
2. 자바 객체의 생성 소멸과 같은 라이프 사이클을 관리하며, 프로그램에서는 객체를 생성할 필요 없이, Spring으로부터 필요한 객체를 가져와 사용할 수 있다.
3. IoC(Inversion of Control)와 DI(Dependency Injection)을 지원한다. 즉, bean 설정 파일을 통해 객체들의 의존관계를 설정할 수 있다..
4. AOP(Aspect Oriented Programming)를 지원하므로 여러 모듈에서 공통으로 사용되지만, 실제 모듈의 핵심은 아닌 기능(예) 트랜잭션이나, 로깅, 보안과 같은 기능)을 분리하여 각 모듈에 적용할 수 있다.
5. EJB처럼 특정 인터페이스를 구현하거나 특정 클래스를 상속받지 않은 일반 자바 객체인 POJO를 지원하다. 따라서 기존에 작성한 소스들을 수정하지 않고 Spring에서 사용이 가능하다.
6. 트랜잭션 처리를 위한 일관된 방법을 제공한다.
7. 다양한 API를 제공한다.

# 개발 전 설정사항

Spring Framework를 사용하기 위해서는 Spring 관련 jar파일이 필요하다. 또한 ALTIBASE와 연동하기 위해서는 ALTIABSE JDBC Driver가 필요하다. 본 장에서는 Spring 환경을 구축하는 방법과 ALTIBASE JDBC Diver를 얻는 방법에 대해 설명한다.

## Spring Framework 환경 구축

Eclipse 스프링 개발 도구(STS) 와 Maven 을 이용하여 손쉽게 Spring 환경을 구축할 수 있다.

Maven 이란? 프로젝트 관리 도구이다.

프로젝트 오브젝트 모델, 표준 집합, 프로젝트 라이프사이클, 의존성관리 시스템, 라이프사이클에 정의된 단계에서 플러그인 골을 실행하기 위한 로직을 포함하는 관리 툴이다.

Maven 예시

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>3.2.xx.RELEASE</version>
    </dependency>
</dependencies>
```

*** ALTIBASE MyBatis 연동가이드 부록 2 를 참조하도록 한다.**

*** 붉은색 표시는 패치 버전에 맞게 설정하도록 한다.**

## ALTIBASE JDBC Driver 얻는 방법

ALTIBASE에서 제공하는 JDBC driver는 Altibase.jar 파일이다. 이 파일은 ALTIBASE가 설치되어있는 서버의 $ALTIBASE_HOME/lib 디렉토리 안에 존재한다.

ALTIBASE 5 버전부터는 $ALTIBASE_HOME/lib 디렉토리에 Altibase.jar 파일과 Altibase5.jar 파일이 존재하는데, Altibase.jar는 일반 JDBC Driver 파일이고, Altibase5.jar는 ALTIBASE 5 버전과 그 이하의 버전을 함께 연동하고 싶을 때 사용하는 JDBC Driver 파일이다. 따라서 하나의 ALTIBASE DB와 연동하거나, 또는 버전이 동일한 여러 대의 ALTIBASE와 연동할 경우에는 $ALTIBASE_HOME/lib/Altibase.jar 파일을 사용하면 된다.

연동하려는 ALTIBASE DB Server와 ALTIBASE JDBC Driver가 호환 가능한지 확인을 위해 ALTIBASE JDBC Driver 버전 확인이 필요하다.

ALTIBASE JDBC Driver 버전을 확인하는 방법은 다음의 명령어를 수행하면 된다.

```
$ java –jar Altibase.jar

JDBC Driver Info : Altibase Ver = 6.3.1.2.6 for JavaVM v1.4, CMP:7.1.1, Oct  6 2014 13:54:56
```

이때, ALTIBASE DB Server의 cm protocol version과 ALTIBASE JDBC Driver의 CMP가 동일하면 호환 가능하다.

```
$ altibase -v version 6.3.1.2.7 X86_64_LINUX_redhat_Enterprise_ES4-64bit-6.3.1.2.7-release-GCC3.4.6 (x86_64-unknown-linux-gnu) Oct  8 2014 09:16:24, binary db version 6.2.1, meta version 6.3.1, cm protocol version 7.1.1, replication protocol version 7.4.1
```

버전이 UP 되면서 JDBC 관련 버그가 fix되었을 가능성이 있으므로, 일반적으로 ALTIBASE DB Server의 버전과 같거나 이 보다 더 최신의 ALTIBASE JDBC Driver 파일을 사용하는 것을 권장한다.

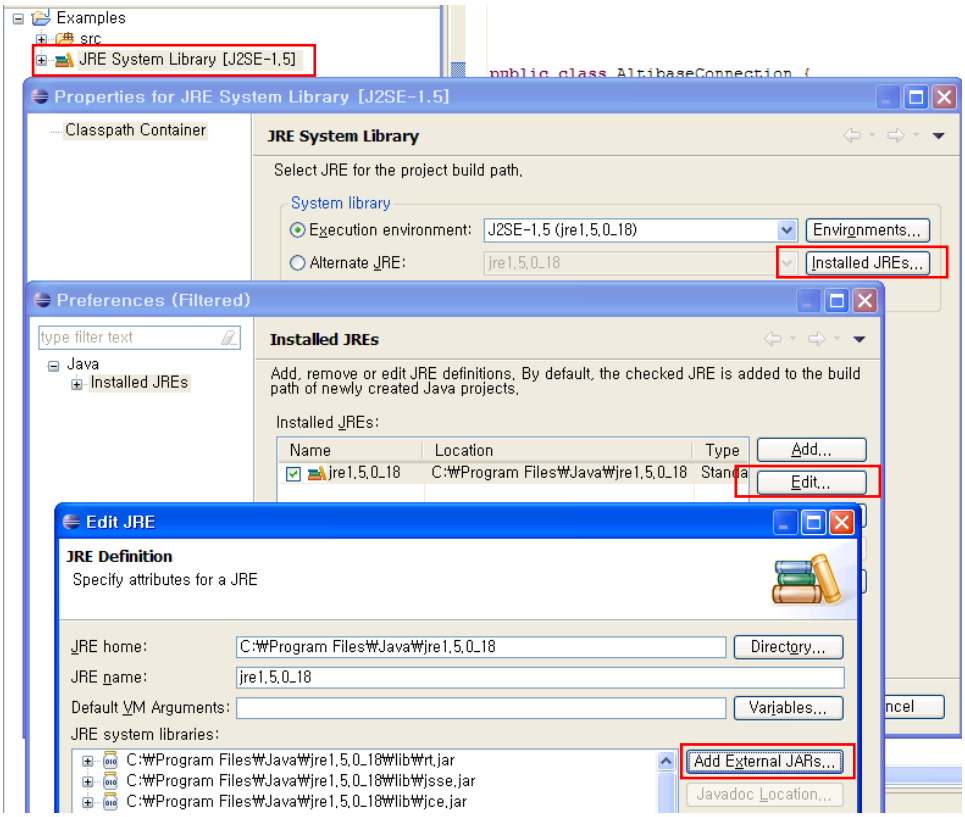
# Setting up JDBC Driver

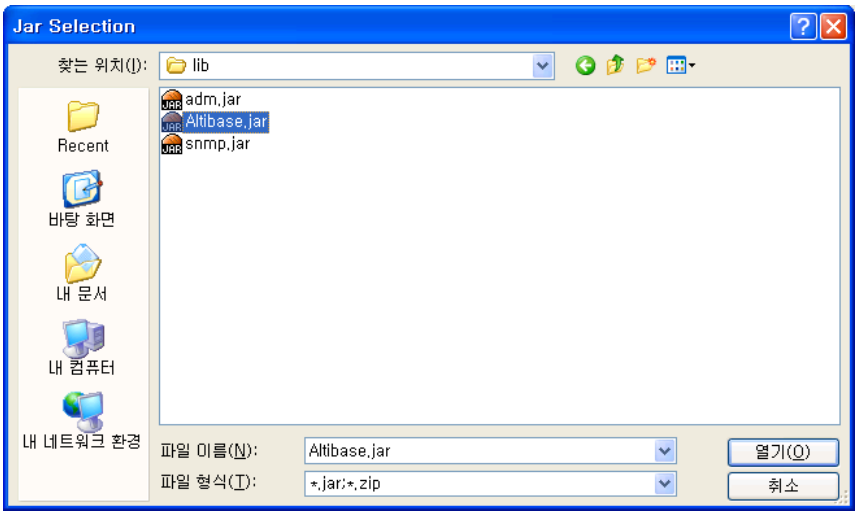
ALTIBASE JDBC Driver를 setting하는 방법에 대해 설명한다.

## 프로젝트에 설정하는 방법(Eclipse)

Eclipse에서 해당 프로젝트에 ALTIBASE JDBC Driver를 추가하는 방법은 다음과 같다.

프로젝트 – JRE System Library - Properties – Installed JREs – 항목 중 jre를 클릭 – Edit – Add External JARs 를 클릭하여 ALTIBASE JDBC Driver인 Altibase.jar를 추가한다.





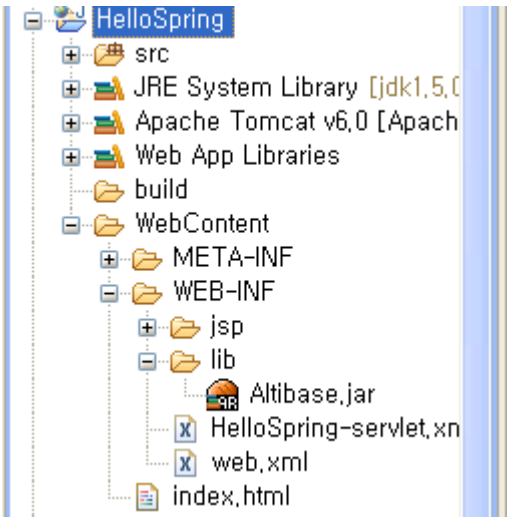

## web application에 설정하는 방법

만약 웹 어플리케이션을 구현할 경우에는 ALTIBASE JDBC Driver를 두 가지 방법으로 설정할 수 있는데, 첫 번째 방법은 각 웹 서버의 적절한 디렉토리에 Altibase.jar 파일을 위치시켜, 모든 웹 어플리케이션에서 전역적으로 ALTIBASE JDBC Driver를 사용할 수 있는 방법이고(웹 서버 별 연동가이드 문서를 참고한다.), 다른 방법은 웹 어플리케이션 별로 ALTIBASE JDBC Driver를 설정하여 해당 웹 어플리케이션에서만 ALTIBASE JDBC Driver를 사용하는 것이다. 만약 웹 어플리케이션에 ALTIBASE JDBC Driver를 설정한다면 다음의 위치에 Altibase.jar를 위치시키면 된다.

```
Web_application\WEB-INF\lib
```

만약, Eclipse에서 Web Project로 프로젝트를 생성하였다면 다음의 디렉토리에 Altibase.jar를 위치시킨다.

```
프로젝트\WebContent\WEB-INF\lib
```



# DataSource 설정

Spring에서 DB와 연동하기 위해서는 bean 설정 파일에 DataSource 관련 bean을 설정해야 한다. (본 문서에서 bean 설정 파일의 이름은 applicationContext.xml으로 정의하였다.)

DataSource를 설정하는 방법 중에서 Spring Framework에서 제공하는 DriverManagerDataSource를 이용하는 방법, Jakarta에서 제공하는 DBCP를 이용하는 방법, DB 벤더에서 제공하는 ConnectionPool을 이용하는 방법 등이 있다. 본 장에서는 이 방법들을 이용하여 ATLIBASE와 연동하는 방법에 대해 설명한다.

## DriverManagerDataSource 이용

Spring Framework에서 제공하는 DriverManagerDataSource 클래스를 이용하여 DB와 연동할 경우에는 bean 설정파일에서 dataSource bean을 설정할 때 class 속성 값을 org.springframework.jdbc.datasource.DriverManagerDataSource로 지정하면 된다.

이때 dataSource bean에 DB와 연동하기 위한 여러 가지 property들을 정의 할 수 있는데, 다음과 같은 Property들을 ALTIBASE에 맞게 정의하면 된다.

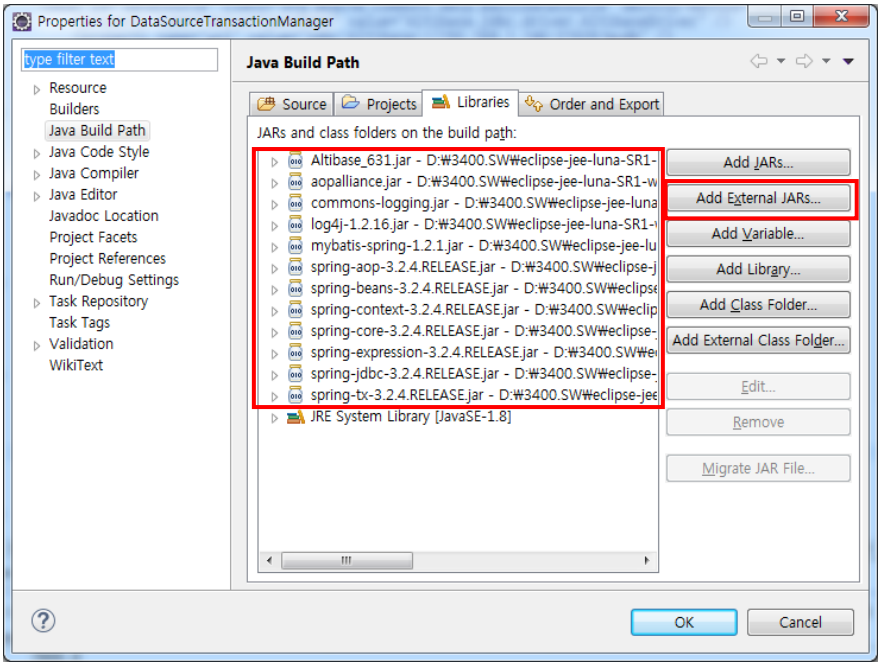
| Property | 설명 |
|---|---|
| driverClassName | ALTIBASE JDBC driver class Name |
| url | ALTIBASE와 연결을 위한 Connection string정보 “jdbc:Altibase://IP:port_no/db_name” 형태로 기입 |
| username | 데이터베이스 계정 |
| password | 데이터베이스 패스워드 |

다음은 예제로 제공되는 DataManagerDataSourceConnection의 acpplicationContext.xml 파일의 일부이다.

예) DataManagerDataSourceConnection의 applicationContext.xml 파일

```
…
<bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <!-- JDBC Driver 클래스 명 설정 -->
        <property name="driverClassName"
value="Altibase.jdbc.driver.AltibaseDriver" />
        <       <!-- connection url -->
        <property name="url" value="jdbc:Altibase://192.168.1.35:21129/mydb" />
        <!-- DB 사용자 계정 설정 -->
        <property name="username" value="sys" />
        <!-- DB 사용자 패스워드 설정 -->
        <property name="password" value="manager" />
</bean>
…
```

예제에 포함된 DataManagerDataSourceConnection 프로젝트를 실행하기 위해서는 Altibase.jar, spring.jar, spring-jdbc.jar, common-loggings.jar 파일이 필요하므로 해당 jar 파일을 추가해주어야 한다.




## DBCP를 이용

Spring에서 Jakarta에서 제공하는 DBCP(Jakarta Commons Database Connection Pool) API를 이용하여 ConnectionPool 기반의 DataSource를 설정할 수 있다. 이 때 사용하는 DataSource 클래스는 org.apache.commons.dbcp.BasicDataSource 이다. 설정하는 방법은 위에서 설명한 DriverManagerDataSource 클래스를 이용하는 것처럼 applicationContext.xml 파일에 BasicDataSource 클래스를 사용하여 dataSource bean을 작성하면 된다.

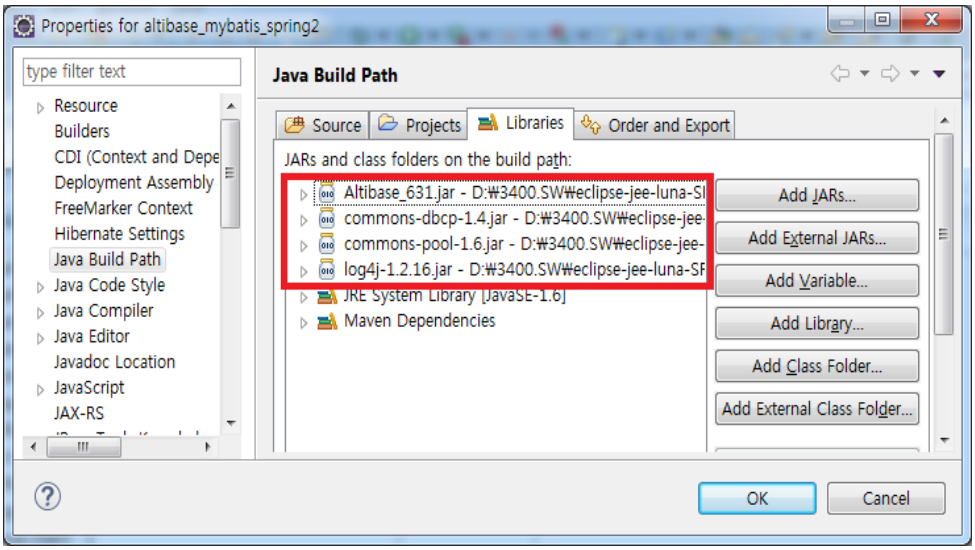
예) DBCPConnection의 applicationContext.xml 파일

```
…
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <!-- JDBC Driver 클래스 명 설정 -->
        <property name="driverClassName"
value="Altibase.jdbc.driver.AltibaseDriver" />
        <!-- connection url -->
        <property name="url" value="jdbc:Altibase://192.168.1.35:21129/mydb" />
        <!-- DB 사용자 계정 설정 -->
        <property name="username" value="sys" />
        <!-- DB 사용자 패스워드 설정 -->
        <property name="password" value="manager" />
</bean>
…
```

BasicDataSource 클래스는 ConnectionPool을 관리하기 위해 다양한 Property들을 제공한다.

| Property | 설명 |
|---|---|
| driverClassName | ALTIBASE JDBC driver class Name |
| url | ALTIBASE와 연결을 위한 Connection string정보<br>jdbc:Altibase://IP:port_no/db_name" 형태로 기입 |
| username | 데이터베이스 계정 |
| password | 데이터베이스 패스워드 |
| maxActive | 최대 Connection 수, 0 은 무제한. default는 8 |
| initialSize | 초기 Connection 수. default는 0 |
| maxIdle | Pool에 idle하게 유지하는 최대 연결 수. default는 8 |
| maxWait | 최대 연결 시도 시간 (단위 : millisec) -1 은 무한 대기<br>Default는 무한 대기 |
| validationQuery | 연결의 validation을 체크하기 위해 사용하는 SQL문<br>반드시 최소한 1 개이상의 row가 return되는<br>select문으로 지정<br>ex) select 1 from dual |
| defaultAutoCommit | autocommit 모드를 설정. default는 true |
| defaultTransactionIsolation | Transaction Isolation level을 설정<br>NONE, REPEATABLE_READ, SERIALIZABLE의 값을<br>설정할 수 있고, default는 DB서버의 default 값을<br>따른다. ALTIBASE의 isolation level은 default 로<br>READ COMMITTED 이다. |

예제에 포함된 DBCPConnection 프로젝트를 실행하기 위해서는 Spring 설정 library 파일 이외에 Altibase.jar, common-logging.jar, common-dbcp.jar, common-pools.jar 파일이 필요하다.



## ALTIBASE의 ConnectionPool을 이용

ALTIBASE에서 제공하는 ABConnectionPoolDataSource 클래스를 사용하면 ALTIBASE의 ConnectionPool을 이용할 수 있다. 위에서 설명한 다른 DataSource처럼 applicationContext.xml에 ABConnectionPoolDataSource 클래스를 이용하여 dataSource bean을 정의하면 된다.

예) AltibaseConnectionPool의 applicationContext.xml 파일

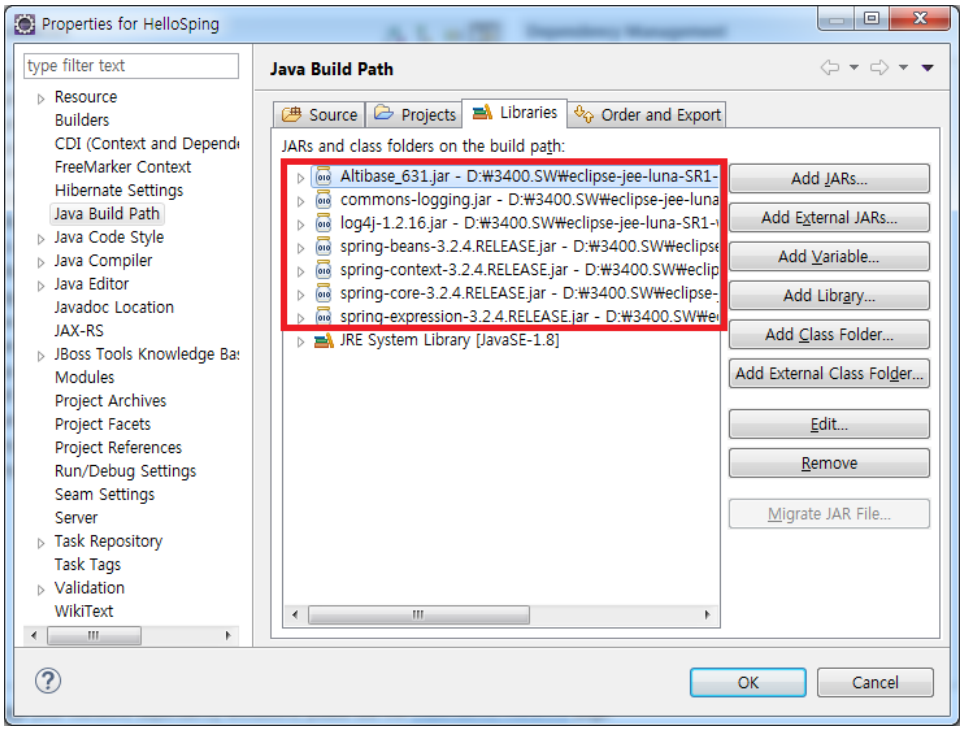
```
…
<bean id="dataSource"
class="Altibase.jdbc.driver.AltibaseConnectionPoolDataSource">
   <!-- connection  URL  대문자 주의 -->
   <property name="URL" value="jdbc:Altibase://192.168.1.35:21129/mydb" />
   <!-- DB 사용자 계정 설정 -->
   <property name="user" value="sys" />
   <!-- DB 사용자 패스워드 설정 -->
<property name="password" value="manager" />
</bean>
…
```

ABConnectionPoolDataSource을 이용할 경우 ABConnectionPoolDataSource 내부에서 자동으로 ALTIBASE JDBC Driver class를 로딩하므로 driverClassName property를 지정할 필요가 없다. 또한 DB 사용자 계정을 나타내는 property가 username이 아니고 user임을 주의해야 한다.

ABConnectionPoolDataSource 클래스는 ConnectionPool을 관리하기 위해 다양한 property들을 제공한다.

| Property | 설명 |
|---|---|
| URL | ALTIBASE와 연결을 위한 Connection string정보 jdbc:Altibase://IP:port_no/db_name" 형태로 기입 |
| user | 데이터베이스 계정 |
| password | 데이터베이스 패스워드 |
| maxPoolSize | 최대 Connection 수. 기본값 10. |
| minPoolSize | 최소 Connection 수. 기본값 0. |
| initialPoolSize | 초기 Connection 수. 기본값 1. |
| maxIdleTime | idle 대기 시간 |
| propertyCycle | ConnectionPool이 다 찼을 때 대기 시간(millisec) |

예제에 포함된 AltibaseConnectionPool 프로젝트를 실행하기 위해서는 Altibase.jar, spring.jar, spring-jdbc.jar, common-logging.jar 파일이 필요하다.


Properties for HelloSping
type filter text
Resource
Builders
CDI (Context and Depend
FreeMarker Context
Hibernate Settings
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
JAX-RS
JBoss Tools Knowledge Ba
Modules
Project Archives
Project Facets
Project References
Run/Debug Settings
Seam Settings
Server
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
Altibase_631.jar - D:₩3400.SW₩eclipse-jee-luna-SR1-
commons-logging.jar - D:₩3400.SW₩eclipse-jee-luna
log4j-1.2.16.jar - D:₩3400.SW₩eclipse-jee-luna-SR1-
spring-beans-3.2.4.RELEASE.jar - D:₩3400.SW₩eclipse
spring-context-3.2.4.RELEASE.jar - D:₩3400.SW₩eclip
spring-core-3.2.4.RELEASE.jar - D:₩3400.SW₩eclipse-
spring-expression-3.2.4.RELEASE.jar - D:₩3400.SW₩e
JRE System Library [JavaSE-1.8]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel

# Additional Connection

ALTIBASE에서 제공하는 FailOver와 Multi-version ALTIBASE 연동하는 방법에 대해 설명한다.

## FailOver를 이용한 Connection

ALTIBASE 5.3.3 부터 FailOver를 지원하는데, FailOver를 사용하기 위해서는 Connection URL부분에 FailOver 관련 속성을 넣어주면 된다.

다음은 FailOver를 이용하여 ALTIBASE에 연결하는 예제이다. DataSource는 위에서 기술한 방법 중 DBCP를 이용하였다.

예) FailOverConnection의 applicationContext.xml 파일

```
…
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
<!-- JDBC Driver 클래스 명 설정 -->
<property name="driverClassName" value="Altibase.jdbc.driver.AltibaseDriver" />
<property name="URL"
   value="jdbc:Altibase://192.168.1.62:21020/mydb?AlternateServers=(192.168.1.146:21020)
&amp;ConnectionRetryCount=1&amp;ConnectionRetryDelay=1&amp;SessionFailOver=on
&amp;LoadBalance=off"/>
" />
<property name="username" value="test" />
<property name="password" value="test" />
</bean>
…
```

Connection URL 부분에 정의할 수 있는 FailOver 관련 property는 다음과 같다.

| Property | 설명 |
|---|---|
| AlternateServer | 장애 발생시 접속하게 될 가용 서버를 나타내며 (IP Address1:Port1, IP Address2:Port2,...) 형식으로 기술한다. |
| ConnectionRetryCount | 가용 서버 접속 실패 시, 접속 시도 반복 횟수 |
| ConnectionRetryDelay | 가용 서버 접속 실패 시, 다시 접속을 시도하기 전에 대기하는 시간(초 단위) |
| LoadBalance | on으로 설정하면 최초 접속 시도 시에 기본 서버와 가용 서버를 포함하여 랜덤으로 선택한다. off로 설정하면 최초 접속 시도 시에 기본 서버에 접속하고, 접속에 실패하면 AlternateServer로 기술한 서버에 접속한다. |
| SessionFailOver | STF(Service Time Fail-Over)를 할 것인지 여부를 나타낸다.<br>on : STF, off : CTF |

| Property | 설명 |
|---|---|
| | CTF(Connection Time Fail-Over)는 DBMS 접속 시점에 장애를 인식하여 장애가 발생한 DBMS대신 다른 가용 STF(Service Time Fail-Over)는 DBMS 접속에 성공하여 노드의 DBMS로 접속하고 서비스를 진행한다.<br><br>서비스하는 도중에 장애가 발생하는 것으로, 다른 가용 노드의 DBMS에 다시 접속하여 세션의 프로퍼티를 복구한 후 사용자 응용 프로그램의 업무 로직을 다시 수행하도록 하는 것을 의미한다. 즉 장애가 발생한 DBMS에서 수행된 작업을 다시 한 번 수행할 필요가 있는 경우이다. |

FailOver는 첨부된 예제 중 FailOverConnection 프로젝트를 참조하면 된다. FailOverConnection 프로젝트를 실행하기 위해서는 Spring 설정 library 파일 이외에 Altibase.jar, spring-jdbc.jar, common-logging.jar 파일이 필요하다.

## ALTIBASE와 이전 버전을 동시에 Connection

ALTIBASE 는 하나의 어플리케이션에서 ALTIBASE 하위 버전을 동시에 연결할 수 있다.

기존의 Altibase.jar와 구별하기 위해 별도로 명명만 바뀐 ALTIBASE 5 버전의 JDBC Altibase5.jar 가 필요하다.

다음은 Altibase.jar와 Altibase5.jar 파일을 이용하여 두 버전의 ALTIBASE의 드라이버를 로딩하는 예제이다.

예) MultiVersionConnection의 applicationContext.xml 파일

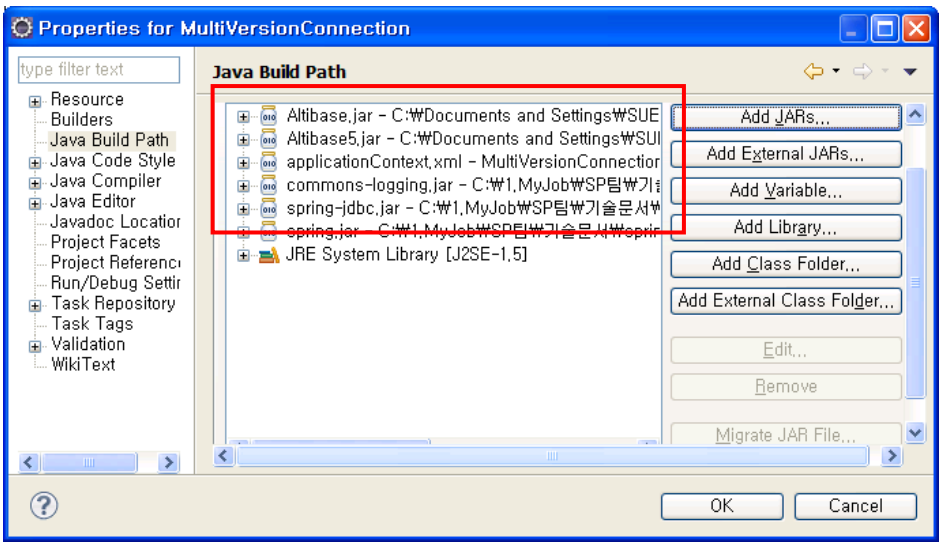
```
<!—ALTIBASE 6 버전에 대한 DataSource 설정 -->
<bean id="dataSource1"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<!--  JDBC Driver 클래스 명 설정 -->
    <property name="driverClassName" value="Altibase.jdbc.driver.AltibaseDriver"/>
    <!-- connection URL  대문자 주의-->
    <property name="URL" value="jdbc:Altibase://127.0.0.1:20300/mydb"/>
    <!-- DB 사용자 계정 설정 -->
    <property name="username" value="sys"/>
    <!-- DB 사용자 패스워드 설정 -->
<property name="password" value="manager" />
</bean>

<!—ALTIBASE 6 이전 버전에 대한 DataSource 설정 -->
<bean id="dataSource2"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<!--  JDBC Driver 클래스 명 설정 -->
<property name="driverClassName" value="Altibase.jdbc.driver.AltibaseDriver"/>
    <!-- connection URL  대문자 주의-->
    <property name="URL" value="jdbc:Altibase://192.168.1.35:21129/mydb"/>
    <!-- DB 사용자 계정 설정 -->
    <property name="username" value="sys"/>
```

```
    <!-- DB 사용자 패스워드 설정 -->
    <property name="password" value="manager" />
</bean>

<!-- DAO 클래스의 bean 설정 -->
<bean id="accountDao1" class="com.altibase.banking.AccountDao">
    <property name="dataSource" ref="dataSource1"/>
</bean>

<!-- DAO 클래스의 bean 설정 -->
<bean id="accountDao2" class="com.altibase.banking.AccountDao">
    <property name="dataSource" ref="dataSource2"/>
</bean>

…
```

MultiVersionConnection의 AccountApp.java파일

```
…
Resource resource = new ClassPathResource("applicationContext.xml");
BeanFactory factory = new XmlBeanFactory(resource);

AccountDao accountDao1 = (AccountDao)factory.getBean("accountDao1");
AccountDao accountDao2 = (AccountDao)factory.getBean("accountDao2");

…
```

예제에 포함된 MultiVersionConnection프로젝트를 실행하기 위해서는 기존에 사용했던 jar 파일 뿐만 아니라, Altibase.jar와 Altibase5.jar 파일이 필요하다. 이 파일들은 ALTIBASE가 설치된 디렉토리($ALTIBASE_HOME)의 lib 디렉토리 안에 존재하는데 ALTIBASE 5 버전의 Altibase5.jar 파일, 그 이전 버전의 Altibase.jar 파일을 사용하면 된다.



# 트랜잭션 관리

Spring Framework를 이용하면 다양한 방법으로 트랜잭션을 처리할 수 있다. 먼저, 트랜잭션 관리자(TransactionManager)를 bean 설정 파일에 지정해주고, 트랜잭션 처리를 해주면 된다. 트랜잭션 처리는 필요에 따라 소스 코드 내에서 직접 프로그램을 구현할 수 있고, 또 다른 방법으로 bean 설정 파일에서 선언적으로 처리할 수 있다.

Spring Framework는 데이터베이스 연동 기술에 따라 여러 PlatformTransactionManager 인터페이스의 구현 클래스를 제공하는데 이 클래스들을 TransactionManager로 지정하면 된다. 예를 들어, 로컬 트랜잭션을 처리하기 위해서 DataSourceTransactionManager 클래스를 TransactionManager로 지정하고, 분산 트랜잭션을 처리하기 위해서는 JtaTransactionManager 클래스를 TransactionManager로 지정하면 된다.

TransactionManager를 지정한 후 트랜잭션을 선언적으로 처리하기 위해서는 bean 설정 파일에 트랜잭션을 처리하는 bean을 정의하면 된다. 이때, 이 bean에 TransactionProxyFactoryBean 클래스를 지정하여 각 메소드이름 별로 트랜잭션을 처리하는 방법을 지정해주면 된다.

본 문서에서는 이 클래스들을 TransactionManager를 설정하는 방법과 bean 파일을 이용해서 선언적으로 트랜잭션을 처리하는 방법에 대해 살펴본다.

## DataSourceTransactionManager 설정

PlatformTransactionManager 구현 클래스 중 DataSourceTransactionManager 클래스를 이용하여 TransactionManager를 지정할 수 있다. bean 설정 파일(applicationConext.xml)에 transactionManager bean에 org.springframework.jdbc.datasource.DataSourceTransactionManager 클래스를 지정해 주면 된다.

이 후, 선언적으로 트랜잭션을 처리할 때 트랜잭션을 처리하는 bean의 property중 서 transactionManager에 위해서 정의한 transactionManager bean을 지정해주면 된다.

예) DataSourceTransactionManager의 applicationContext.xml 파일

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

<bean id="txProxyTemplate" abstract="true"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"    ref="transactionManager" />
    <property name="transactionAttributes">
        <props>
            <prop key="get*">PROPAGATION_REQUIRED, readOnly </prop>
            <prop key="add*">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>

<bean id="accountService" parent="txProxyTemplate">
<property name="target">
        <bean class="com.altibase.banking.AccountService">
            <property name="accountDao" ref="accountDao"/>
```

```
        </bean>
    </property>
</bean>
…
```

트랜잭션을 처리하기 위해서는 트랜잭션을 관리(TransactionManager)하는 방법 뿐만 아니라, 트랜잭션을 처리하는 방법도 기술해야 하는데 위의 예제는 트랜잭션 처리 방법 중 springframework.transaction.interceptor.TransactionProxyFactoryBean 클래스를 이용하여 bean 설정 파일(applicationConext.xml)에 선언적으로 트랜잭션을 처리하는 방법으로 작성되었다.(해당 내용은 아래에 자세히 설명한다.)

## TransactionProxyFactoryBean을 이용한 선언적 트랜잭션 처리

TransactionProxyFactoryBean 클래스는 트랜잭션을 처리하기 위한 여러 속성들을 제공하는 클래스이다. bean 설정 파일(applicationContext.xml)에 TransactionProxyFactoryBean 클래스의 bean을 설정해주고, 트랜잭션을 처리하는 방법을 각 메소드 별로 정의하여 선언적으로 트랜잭션을 처리할 수 있다. 이 트랜잭션 처리를 위한 bean의 <property>에 지정하는 트랜잭션 관련 속성들은 다음과 같다.

1. target

   프로퍼티를 통해 트랜잭션을 적용할 대상 객체를 설정한다.

2. transactionManager

   위에서 정의한 TransactionManager를 설정한다.

3. transactionAttrubutes

   트랜잭션 속성을 설정할 때 사용되는 Properties 객체를 설정한다. 이때 트랜잭션을 적용할 메소드 단위로 <prop> 태그를 이용하여 트랜잭션 속성을 정의한다.

   <prop>의 태그 값은 다음과 같은 형식으로 지정한다.

```
PROPAGATION, ISOLATION_NAME, readOnly, timeout, +Exception, -Exception
```

<prop> 태그 값의 각 항목에 대한 설명은 다음과 같다.

| 항목 | 설명 |
|---|---|
| PROPAGATION | 트랜잭션 전파 규칙을 명시. 필수항목.<br><br>PROPAGATION_REQUIRED(기본값),<br>PROPAGATION_REQUIRES_NEW,<br>PROPAGATION_MANDATORY,<br>PROPAGATION_SUPPORTS,<br>PROPAGATION_NOT_SUPPORTED,<br>PROPAGATION_NEVER,<br>PROPAGATION_NESTED |
| ISOLATION_NAME | ISOLATION 속성을 설정. 선택항목.<br><br>DEFAULT,<br>READ_COMMITTED, |

| | |
|---|---|
| | READ_UNCOMMITTED,<br>REPETABLE_READ,<br>SERIALIZABLE.<br>ALTIBASE는 이 중 READ_COMMITTED, SERIALIZABLE, REPETABLE_READ만 지원한다. |
| readOnly | readOnly가 사용되면 트랜잭션을 읽기 전용으로 설정 |
| timeout | 트랜잭션의 타임 아웃 시간을 초 단위로 설정한다. |
| +,-Exception | ROLLBACK 규칙을 설정. +Exception은 해당 Exception이 발생하더라도 COMMIT을 수행하고 -Exception은 해당 Exception이 발생하면 ROLLBACK을 수행한다. |

다음은 TransactionProxyFactoryBean 클래스를 이용하여 트랜잭션 속성들을 정의한 bean 설정 파일(applicaionContext.xml)이다.

예) DataSourceTransactionManager의 applicationContext.xml 파일

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

<bean id="txProxyTemplate" abstract="true"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"    ref="transactionManager" />
    <property name="transactionAttributes">
        <props>
            <prop key="get*">PROPAGATION_REQUIRED, readOnly </prop>
            <prop key="add*">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>

<bean id="accountService" parent="txProxyTemplate">
<property name="target">
        <bean class="com.altibase.banking.AccountService">
            <property name="accountDao" ref="accountDao"/>
        </bean>
    </property>
</bean>

…
```

## 분산 트랜잭션 처리

Spring에서 JOTM(Java Open Transaction Manager)을 이용하면 분산 트랜잭션을 처리할 수 있다. 이 때 JOTM과 연동하기 위해서는 먼저, Spring에서 제공하는 JotmFactoryBean을 등록해야 한다. 그리고 TransactionManager로는 분산 트랜잭션을 제공하는 JtaTransactionManager를 사용하면 된다.

뿐만 아니라, 분산 트랜잭션을 처리하기 위해서는 분산트랜잭션을 제공하는 DataSource를 사용해야 하는데, JOTM에서 제공하는 org.enhydra.jdbc.pool.StandardXAPoolDataSource를 사용할 수도 있지만, 본 문서에서는 ALTIBASE에서 제공하는 Altibase.jdbc.driver. ABXADataSource를 이용하여 분산 트랜잭션을 처리하는 방법에 대해 설명한다.

다음은 org.springframework.transaction.jta.JtaTransactionManager와 Altibase.jdbc.driver.`AltibaseXADataSource`를 이용하여 분산 트랜잭션을 처리하는 예제이다.

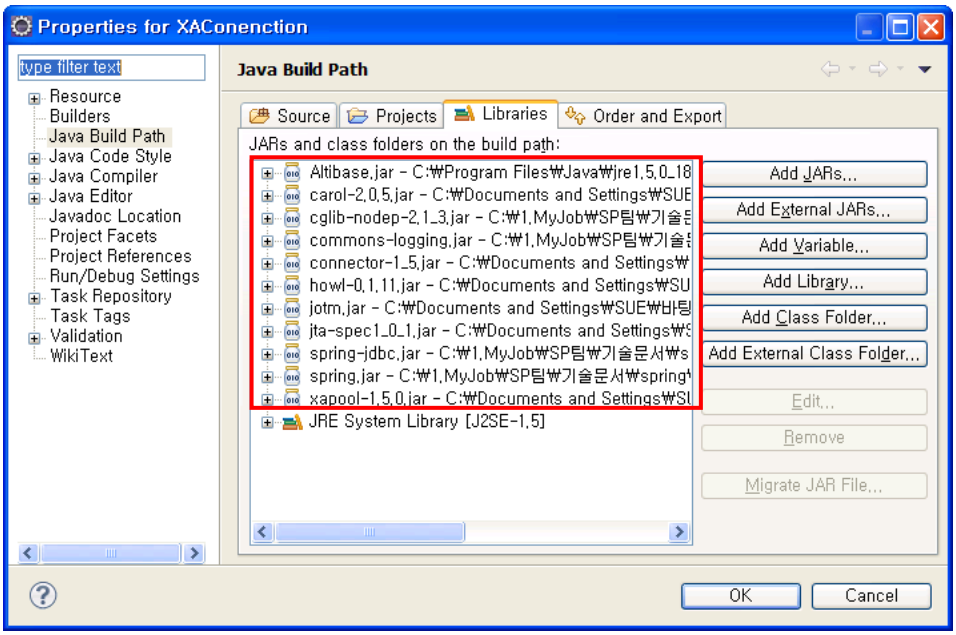
1. `AltibaseXADataSource`를 이용하여 DataSource를 지정
2. JotmFactoryBean을 지정
3. TransactionManager로 JtaTransactionManager 지정
4. TransactionProxyFactoryBean 클래스를 이용하여 선언적 트랜잭션 처리하는 bean을 지정

예) XAConnection의 applicationContext.xml 파일

```
…
<bean id="dataSource1" class="Altibase.jdbc.driver.AltibaseXADataSource">
    <!-- connection url-->
    <property name="url" value="jdbc:Altibase://192.168.1.35:21129/mydb"/>
    <!-- DB 사용자 계정 설정 -->
    <property name="user" value="sys"/>
    <!-- DB 사용자 패스워드 설정 -->
    <property name="password" value="manager" />
</bean>

<bean id="dataSource2" class="Altibase.jdbc.driver. AltibaseXADataSource ">
    <!-- connection url-->
    <property name="url" value="jdbc:Altibase://127.0.0.1:20300/mydb"/>
    <!-- DB 사용자 계정 설정 -->
    <property name="user" value="sys"/>
    <!-- DB 사용자 패스워드 설정 -->
    <property name="password" value="manager" />
</bean>

<bean id="accountDao1" class="com.altibase.banking.AccountDao">
    <property name="dataSource" ref="dataSource1"/>
</bean>

<bean id="accountDao2" class="com.altibase.banking.AccountDao">
    <property name="dataSource" ref="dataSource2"/>
</bean>

<bean id="jotm" class="org.springframework.transaction.jta.JotmFactoryBean"/>

<bean id="transactionManager"
          class="org.springframework.transaction.jta.JtaTransactionManager">
    <property name="userTransaction" ref="jotm"/>
</bean>

<bean id="txProxyTemplate" abstract="true"
 class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"   ref="transactionManager" />
    <property name="transactionAttributes">
          <props>
```

```
                <prop key="get*">PROPAGATION_REQUIRED, readOnly</prop>
                <prop key="add*">PROPAGATION_REQUIRED</prop>
            </props>
    </property>
</bean>

<bean id="accountService" parent="txProxyTemplate">
    <property name="target">
        <bean class="com.altibase.banking.AccountService">
                <property name="accountDao1" ref="accountDao1"/>
                <property name="accountDao2" ref="accountDao2"/>
        </bean>
    </property>
</bean>
…
```

예제에 포함되어 있는 XAConnection 프로젝트를 실행하기 위해서는 기존에 추가했던 jar 파일 이외에 JOTM에 대한 jar 파일들을 더 추가해야 한다.




JOTM에 관련된 파일들은 http://forge.ow2.org/projects/jotm/ 사이트에서 다운로드 받을 수 있다. 다운로드 후 압축을 풀면 jar 파일들이 있는데 이중 carol-2.0.5.jar, connector-1_5.jar, howl-0.1.11.jar, jotm.jar, jta-spec1_0_1.jar, xapool-1.5.0.jar 파일을 프로젝트에 추가해주면 된다. (위의 jar 파일의 이름은 다운로드 받은 JOTM 버전에 따라 이름이 다를 수 있다.)

그 외 cglib-nodep-2.1.3.jar 파일이 별도로 필요하다.

# Spring 연동 시 주의사항

Spring에서 ALTIBASE에 연동할 경우 주의해야 할 사항에 대해 설명한다.

## LOB 데이터 처리

ALTIBASE에서 LOB 데이터를 처리하기 위해서는 반드시 autocommit 모드를 false로 바꾼 후 명시적으로 트랜잭션을 관리해줘야 한다. 따라서 Spring에서 LOB을 처리하기 위해서는 반드시 TransactionManager bean을 명시해줘야 한다.

또한 선언적 트랜잭션을 사용하는 경우에는 propagation을 PROPAGATION_REQUIRED, PROPAGATION_REQUIRES_NEW, PROPAGATION_NESTED 중 하나로 지정해줘야 한다.

만약 TransactionManager를 지정해주지 않았거나, 또는 선언적 트랜잭션을 사용하는데 propagation을 위에 설명한 값 이외의 다른 값으로 지정했을 경우에는 LOB 데이터 조회 시 null 값이 리턴 되거나, "java.sql.SQLException: [0]:LobLocator can not span the transaction 101858625." 과 같은 에러가 발생한다.

그리고 LOB 데이터를 입력 시에도 "java.sql.SQLException: [0]:LobLocator can not span the transaction 101858625." 에러가 발생하게 된다.

예) LobConnection의 applicationContext.xml 파일

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

<bean id="txProxyTemplate" abstract="true"
      class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
      <property name="transactionManager"    ref="transactionManager" />
      <property name="transactionAttributes">
            <props>
                  <prop key="get*">PROPAGATION_REQUIRED</prop>
                  <prop key="add*">PROPAGATION_ REQUIRED </prop>
            </props>
      </property>

</bean>

<bean id="lobSampleService" parent="txProxyTemplate">
      <property name="target">
            <bean class="com.altibase.lob.LobSampleService">
                  <property name="lobSampleDao" ref="lobSampleDao"/>
            </bean>
      </property>
</bean>

…
```

# 부록

sample 예제를 바탕으로 Spring을 사용하는 방법에 대해 좀 더 자세하게 설명한다.

부록에서 설명하는 sample 프로젝트의 이름은 HelloSpring이며 Eclipse에서 작성하였다.

HelloSpring 프로젝트는 인사말 메시지를 print 하는 간단한 프로그램으로 Spring 컨테이너(BeanFactory)로부터 bean 설정파일(applicationContext.xml)에 정의되어 있는 bean(greetingService)을 가져와 bean에 정의되어 있는 인사말 메시지를 print한다.

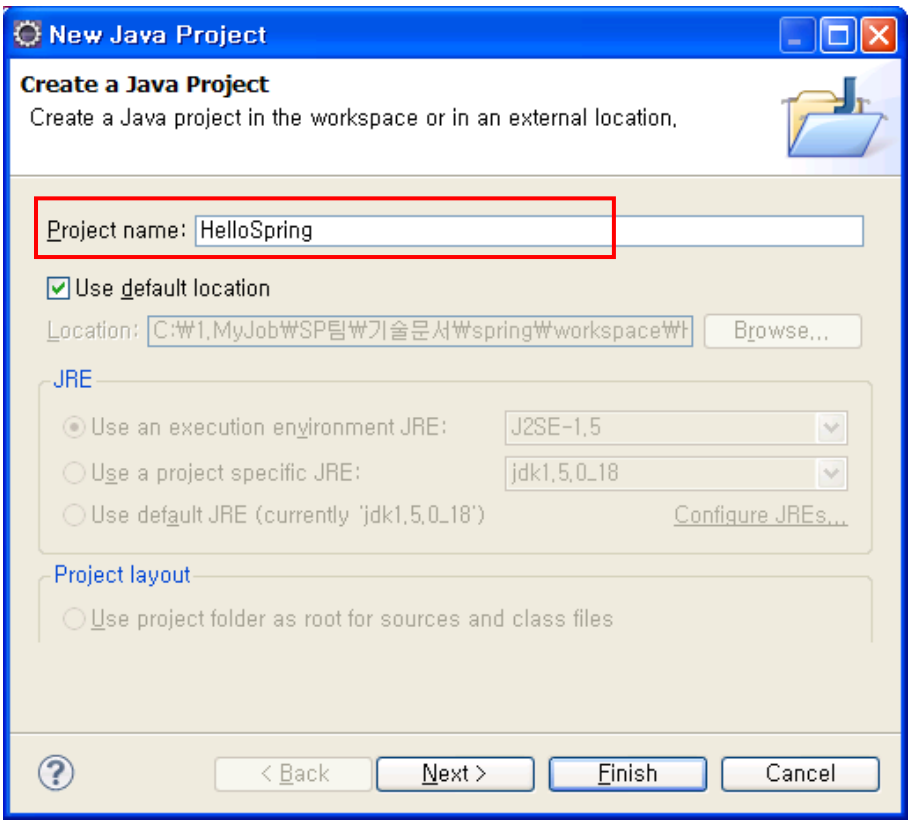
## HelloSpring 구현

1. HelloSpring 프로젝트 생성

   1-1. 메뉴 – File – Java Project 클릭

   1-2. Project name : 에 HelloSpring 입력

   1-3. Finish 버튼을 클릭




2. sample 자바 파일 구현

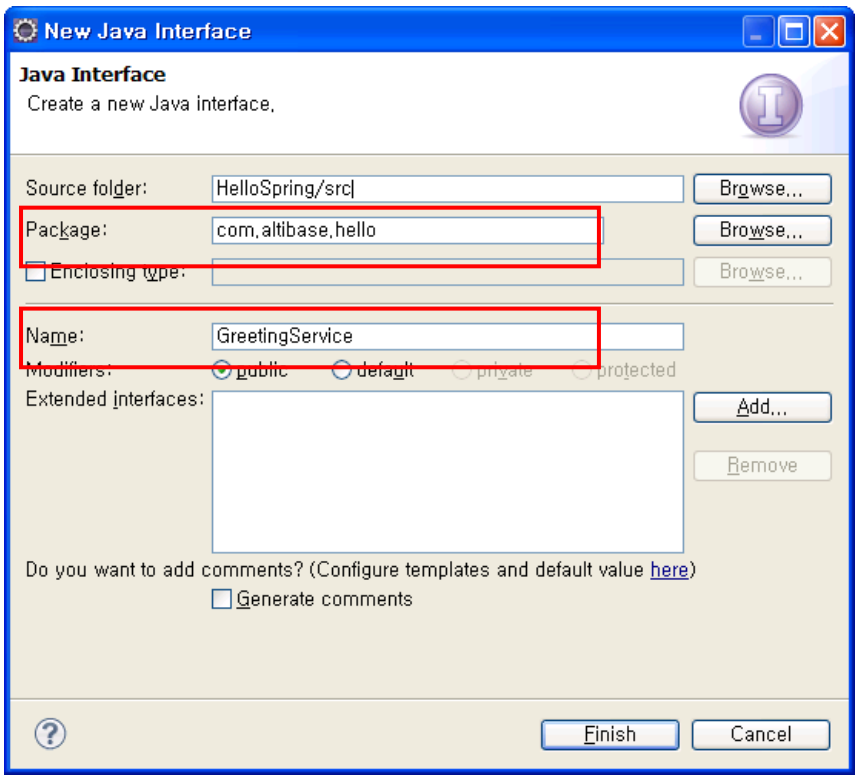
   다음의 클래스 및 인터페이스를 구현한다.

   2-1. GreetingService 인터페이스

서비스를 정의해 놓은 인터페이스이다.

HelloSpring 프로젝트의 src 디렉토리에서 마우스 오른쪽 버튼 클릭하여 New - Interface를 클릭한다.

Package: 에 com.altibase.hello 를 입력하고 Name: 에 GreetingService를 입력한다.




GreeingService.java 파일이 생성되면 다음과 같이 소스를 작성한다.

```
package com.altibase.hello;

public interface GreetingService {
        public void sayGreeting();
}
```

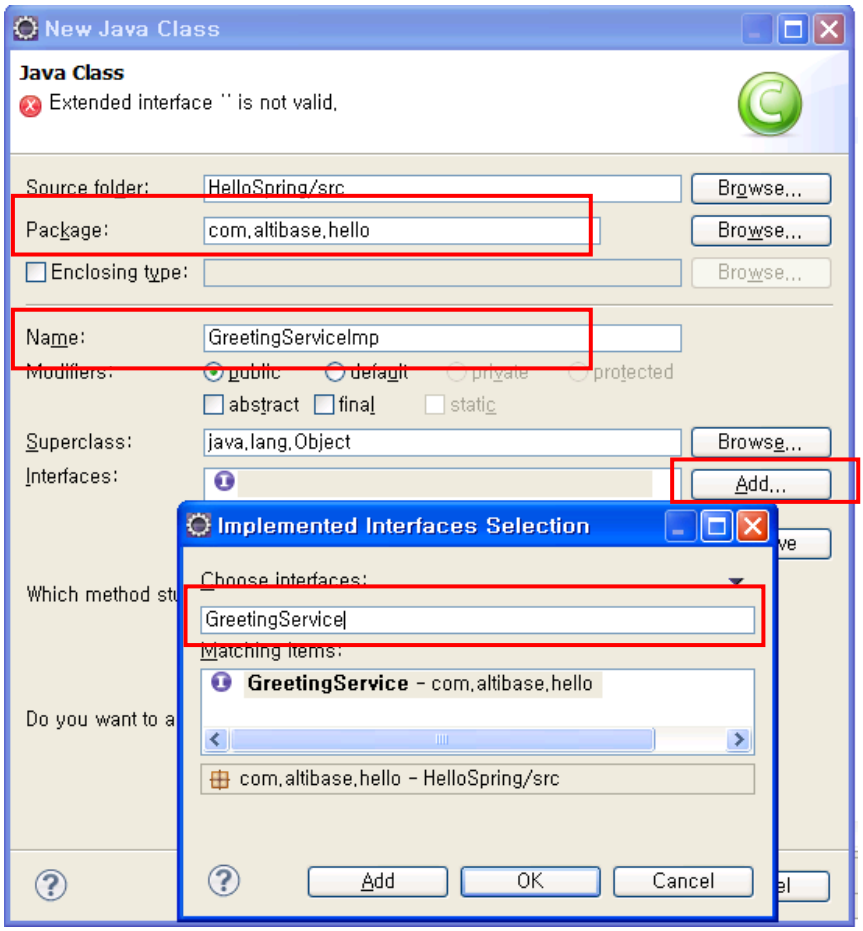
2-2. GreetingServiceImp 클래스

GreetingServie 인터페이스를 구현한 클래스이다.

HelloSpring 프로젝트의 src 디렉토리에서 마우스 오른쪽 버튼 클릭하여 New - Class를 클릭한다.

Package: 에 com.altibase.hello 를 입력하고 Name: 에 GreetingServiceImp를 입력한다.

Interfaces: 옆에 Add 버튼을 클릭하여 GreetingService를 입력한 후 OK 버튼을 클릭한다.




GreeingServiceImp.java 파일이 생성되면 다음과 같이 소스를 작성한다.

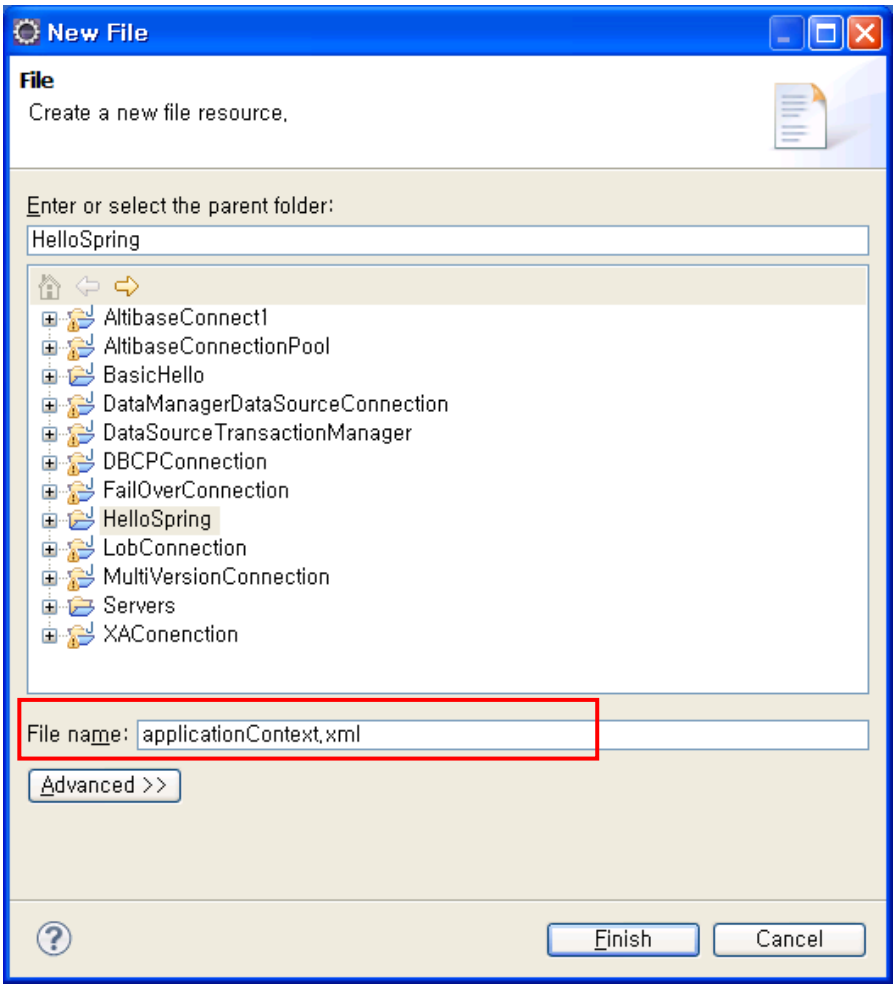
```
package com.altibase.hello;

public class GreetingServiceImp implements GreetingService{
        private String greeting;
        public GreetingServiceImp(){

        }
        public GreetingServiceImp(String greeting){
                this.greeting = greeting;
        }
        public void sayGreeting() {
                System.out.println(greeting);
        }
        public void setGreeting(String greeting){
                this.greeting = greeting;
        }
}
```

3. bean 설정 파일 정의

bean 설정 파일(applicationContext.xml)에 GreetingServiceImp에 대한 bean을 정의한다.

HelloSpring 프로젝트에서 마우스 오른쪽 버튼 클릭하여
New – File을 클릭한다.

File name: 에 applicationContext.xml을 작성한다.




applicationContext.xml 파일에 다음과 같이 작성한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd">

   <bean id="greetingService" class="com.altibase.hello.GreetingServiceImp">
                  <property name="greeting">
                        <value>Hello</value>
                  </property>
   </bean>

</beans>
```

greetingService는 bean의 이름으로 java 프로그램에서 해당 이름으로 bean 가져와 사용할 수 있다.

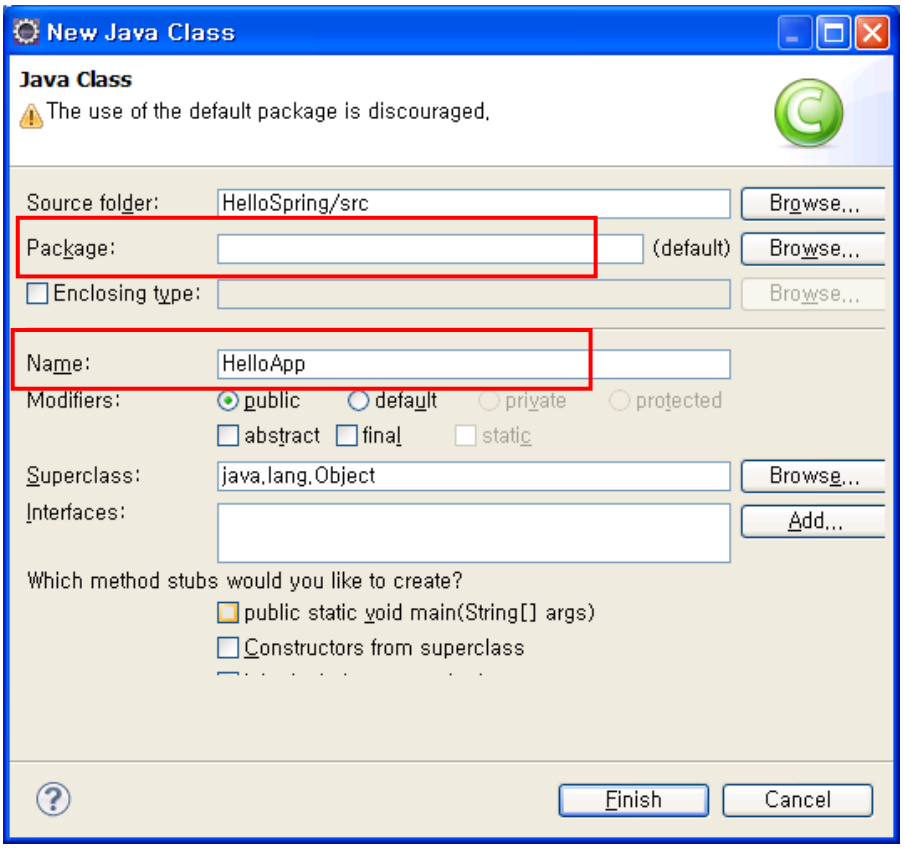
class에는 서비스를 담당하는 GreetingServiceImp 클래스의 full name으로 명시해 주고, property에 GreetingServiceImp에 정의되어 있는 멤버 변수 greeting에 setting될 메시지를 넣어주면 된다.

4. HelloApp 클래스 정의

main 클래스로 BeanFactory로부터 GreetingServiceImp가 정의되어있는 bean을 가져와 서비스 메소드(sayGreeting())를 호출한다.

HelloSpring 프로젝트의 src 디렉토리에서 마우스 오른쪽 버튼 클릭하여 New – Class를 클릭한다.

Package: 부분은 공락으로 나두고 Name: 에 HelloApp를 입력한다.




HelloApp.java 파일이 생성되면 다음과 같이 소스를 작성한다.

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.altibase.hello.GreetingService;
public class HelloApp {

        /**
         * @param args
         */
```

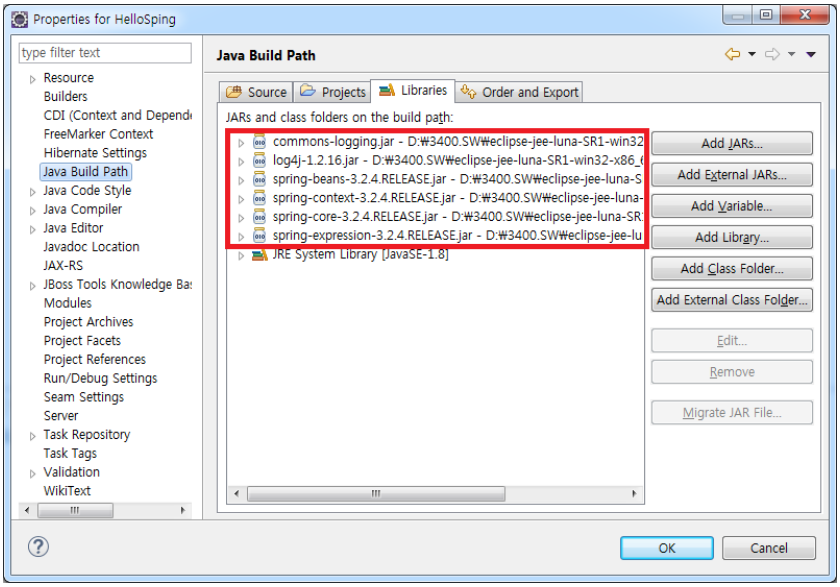
```
        public static void main(String[] args) {

        @SuppressWarnings("resource")
                ApplicationContext factory = new
ClassPathXmlApplicationContext("applicationContext.xml");

        GreetingService greetingService =
(GreetingService)factory.getBean("greetingService");
        greetingService.sayGreeting();
        }
}
```

위의 소스 applicationConext.xml은 위에 정의한 bean 설정 파일이고, greetingService는 applicationContext.xml에 정의된 GreetingServiceImp 클래스에 대한 bean 이름을 나타낸다.

5. 관련 jar 파일들 추가

Spring관련 jar 파일을 HelloSpring에 추가한다.

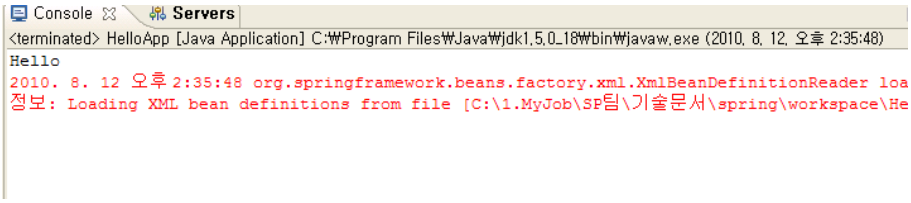
HelloSpring 프로젝트에서 마우스 오른쪽 버튼 클릭하여 Properties를 클릭 – Java Build Path – Libraries 에서 Add External JARS를 클릭하여 Spring Framework의 jar 파일 중 spring.jar 파일과 common-logging.jar 파일을 추가한다. 해당 파일의 위치는 DriverManagerDataSource 설명 부분을 참조한다.




6. HelloSpring 프로젝트를 실행한다.

HelloSpring 프로젝트를 클릭한 후 메뉴에서 Run을 실행하거나 Run 실행 단추를 클릭한다.

Console창에 Hello 문자열이 print된다.


Console
Servers
<terminated> HelloApp [Java Application] C:₩Program Files₩Java₩jdk1.5.0_18₩bin₩javaw.exe (2010. 8. 12. 오후 2:35:48)
Hello
2010. 8. 12 오후 2:35:48 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loa
정보: Loading XML bean definitions from file [C:\1.MyJob\SP팀\기술문서\spring\workspace\He

**알티베이스㈜**
서울특별시 구로구 구로 3 동 182-13
대륭포스트 2 차 1008 호
02-2082-1000
http://www.altibase.com

**대전사무소**
대전광역시 서구 둔산동 921
주은리더스텔 901 호
042-489-0330

**기술지원본부**
서울특별시 구로구 구로 3 동 182-13
대륭포스트 2 차 908 호
02-2082-1000

**기술지원센터**
02-2082-1114
http://support.altibase.com