\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*Chapter 10 ORM 연동 지원\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

: SPRING은 Hibernate, JPA, JDO와 같은 완전한 ORM 기능을 제공하는 솔루션뿐만 아니라 iBatis와 같은 SQL 기반 매퍼 Framework까지 널리 사용되고 있는

ORM 솔루션과의 연동을 기존적으로 제공하고 있다. 본 절에서는 이들 ORM Framework 중 Hibernate, JPA, 그리고 iBatis와 SPRING을 연동하는 방법에 대해서

살펴보도록 하겠다.

===================================================================================================

1. @Repository를 이용한 예외 변환 처리

: 앞서 8장에서 배웠던 JdbcTemplate Class를 사용하는 Code를 보면 SQLException이 발생하지 않음을 알 수 있다.

public int count(){

return jdbcTemplate.queryForInt("select count(\*) from GUESTBOOK\_MESSAGE");

}

JdbcTemplate Class는 내부적으로 JDBC API를 이용해서 Database 연동을 처리하는데, 이 과정에서 SQLException이 발생할 경우 SQLException을

SPRING이 DataAccessException으로 변환해서 발생시킨다.

이와 비슷하게 @Repository Annotation을 이용하면 Hibernate와 JPA API가 발생시키는 예외를 SPRING의 DataAccessException으로 변환할 수 있다.

예를 들어, Hibernate API를 사용하는 과정에서 Hibernate의 ObjectNotFoundException 예외가 발생하면 동일한 의미를 갖는 SPRING의

ObjectRetrievalFailureException을 살생시킨다.

다음 Code는 @Repository Annotation의 적용 예를 보여주고 있다.

import org.springframework.stereotype.Repository;

@Repository

public class HibernateItemDao implements ItemDao {

private SessionFactory sessionFactory;

public void setSessionFactory(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

}

@Override

public Item findById(Integer itemId) {

return (Item) sessionFactory.getCurrentSession()

.get(Item.class, itemId);

}

}

@Repository Annotation이 적용된 DAO Class의 메서드가 발생시킨 예외를 SPRING의 DAO 예외로 변환하려면, 다음과 같이 PersistenceExceptionTranslationPostProcessor를

빈 객체로 등록해 주어야 한다.

<bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor"/>

<bean id="itemDao" class="madvirus.spring.chap10.store.dao.hibernate.HibernateItemDao"

p:sessionFactory-ref="sessionFactory"/>

===================================================================================================

2. Hibernate 연동 지원

: SPRING은 간단한 설정만으로 Hibernate의 SessionFactory와 SPRING의 Transaction을 연동할 수 있도록 지원하고 있다. SPRING이 지원하는 Hibernate 연동 기능을

제공하려면 다음과 같이 설정해 주면 된다.

- LocalSessionFactoryBean을 이용한 SessionFactory 설정

- AnnotationSessionFactoryBean을 이용한 SessionFactory 설정

- HibernateTransactionManager를 이용한 Transaction 관리자 설정

- DAO에서 SessionFactory 사용하기

본 절에서는 Hibernate 자체의 사용방법은 살펴보지 않을 것이며, SPRING에서 Hibernate를 연동하는 방법만 살펴볼 것이다. Hibernate의 사용방법이

궁금한 학생들은 http://www.hibernate.org Site나 관련서적을 참고하기 바란다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2.1 SessionFactory 설정

: SPRING은 SessionFactory를 생성해 주는 두 개의 팩토리 빈 Class를 제공하고 있다.

- LocalSessionFactoryBean : XML Mapping 설정을 이용해 SessionFactory를 생성해준다.

- AnnotationSessionFactoryBean : XML Mapping 설정과 Annotation Mapping 설정을 이용해서 SessionFactory를 생성해 준다.

LocalSessionFactoryBean Class는 XML Mapping 설정 파일을 이용하는 SessionFactory를 생성할 때 사용된다. Hibernate는 hibernate.cfg.xml 파일을 이용해서

DB 연결방법, Mapping 파일 목록, 프로퍼티 등을 설정하는데, LocalSessionFactoryBean Class를 이용해서 이들 정보를 설정할 수 있도록 하고 있다. 차이점이 있다면,

Hibernate의 설정 파일은 Transaction 관리에 대한 것도 설정하는 반면에, SPRING과 연동할 때에는 SPRING이 제공하는 Transaction 련 설정을 사용한다는 점이다.

아래 Code는 LocalSessionFactoryBean Class를 이용해서 SessionFactory를 설정하는 예를 보여주고 있다.

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"

destroy-method="close" p:driverClass="com.mysql.jdbc.Driver"

p:jdbcUrl="jdbc:mysql://localhost/springbook?useUnicode=true&amp;characterEncoding=euckr"

p:user="root" p:password="" />

<bean id="sessionFactory"

class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="mappingResources">

<list>

<value>

madvirus/spring/chap10/store/dao/hibernate/Item.hbm.xml

</value>

<value>

madvirus/spring/chap10/store/dao/hibernate/PaymentInfo.hbm.xml

</value>

<value>

madvirus/spring/chap10/store/dao/hibernate/PurchaseOrder.hbm.xml

</value>

</list>

</property>

<property name="hibernateProperties">

<value>

hibernate.dialect=org.hibernate.dialect.MySQLInnoDBDialect

</value>

</property>

</bean>

LocalSessionFactoryBean Class의 주요 프로퍼티는 다음과 같다.

- dataSource : DataSource를 설정한다.

- mappingResource : Mapping 정보를 담고 있는 설정 파일 목록을 지정한다.

- hibernateProperties : Dialect, 캐시 등 Hibernate의 SessionFactory와 Session을 생성하는데 사용되는 추가적인 프로퍼티를 설정한다.

Annotation을 이용해서 Mapping 정보를 설정할 경우 AnnotationSessionFactoryBean Class를 이용하면 된다.

<bean id="sessionFactory"

class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="annotatedClasses">

<list>

<value>

madvirus.spring.chap10.domain.Item

</value>

</list>

</property>

<property name="hibernateProperties">

hibernate.dialect=org.hibernate.dialect.MySQLInnoDBDialect

</property>

</bean>

AnnotationSessionFactoryBean Class는 LocalSessionFactoryBean Class를 상속받고 있기 때문에, dataSource 프로퍼티, hibernateProperties 프로퍼티의

의미는 동일하다. 추가적으로 annotatedClasses 프로퍼티는 Annotation을 이용해서 Mapping 정보를 설정하고 있는 Class의 목록을 명시한다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2.2 Hibernate를 위한 Transaction 관리자 설정

: JtaTransactionManager를 사용하지 않는 경우 HibernateTransactionManager를 Transaction 관리자로 설정해 주어야 한다. 아래 Code는

HibernateTransactionManager를 이용한 Transaction 관리자의 설정 예를 보여주고 있다.

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="mappingResources">

<list>

<value>

madvirus/spring/chap10/store/dao/hibernate/Item.hbm.xml

</value>

...

</list>

</property>

<property name="hibernateProperties">

<value>

hibernate.dialect=org.hibernate.dialect.MySQLInnoDBDialect

</value>

</property>

</bean>

<bean id="transactionManager"

class="org.springframework.orm.hibernate3.HibernateTransactionManager"

p:sessionFactory-ref="sessionFactory" />

HibernateTransactionManager의 sessionFactory 프로퍼티는 SPRING의 Transaction 관리 기능을 연동할 sessionFactory 빈을 설정한다.

JtaTransactionManager를 사용하더라도 LocalSessionFactoryBean이 Transaction과 Hibernate Session을 알맞게 연동해주기 때문에 추가적인

설정을 하지 않아도 된다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2.3 SessionFactory를 DI로 전달 받기

: SessionFactory 빈을 설정하고 Hibernate를 위한 Transaction 관리자를 설정했다면, 이제 남은 건 DAO Code에서 SessionFactory를 사용하는 것 뿐이다.

특별한 Code를 사용할 필요 없이 DAO Code에서 SessionFactory 빈을 전달받아 사용하면 된다.

<bean id="sessionFactory"

class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

...

</bean>

<bean id="transactionManager"

class="org.springframework.orm.hibernate3.HibernateTransactionManager"

p:sessionFactory-ref="sessionFactory" />

<bean id="itemDao"

class="madvirus.spring.chap10.store.dao.hibernate.HibernateItemDao"

p:sessionFactory-ref="sessionFactory" />

LocalSessionFactoryBean Class는 Hibernate의 SessionFactory.getCurrentSession() 기능을 지원하기 때문에, Hibernate API를 이용하는 DAO Code는

다음과 같이 getCurrentSession() 메서드를 이용해서 Session을 구한 뒤 필요한 기능을 실행하면 된다.

@Repository

public class HibernateItemDao implements ItemDao {

private SessionFactory sessionFactory;

public void setSessionFactory(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

}

@Override

public Item findById(Integer itemId) {

return (Item) sessionFactory.getCurrentSession()

.get(Item.class, itemId);

}

}

===================================================================================================

3. JPA 연동 지원

: JPA(Java Persistence API)는 EJB3 버전에 포함되어 있는 API로서, Hibernate와 마찬가지로 ORM을 지원하는 API이다. EJB3 버전에 포함되어 있기는 하지만

EJB Container가 없어도 사용할 수 있기 때문에 Hibernate와 함께 점차 사용이 늘어나고 있다. JPA는 Hibernate와 달리 Sun에서 제시한 표준 규약이기 때문에

JEE5 Container가 지원한다는 장점도 있다.

EJB Container가 없어도 JPA API를 사용할 수 있도록 해주는 JPA 구현체가 다양하게 존재하는데, 본 절에서는 EclipseLink와 Hibernate의 EntityManager를

중심으로 SPRING과 JPA의 연동 방식을 설명하겠다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.1 EntityManagerFactory 설정

: SPRING은 JPA의 EntityManagerFactory를 생성할 수 있는 두 가지 팩토리 Class를 제공하고 있으며, 이 두 Class는 다음과 같다.

- LocalEntityManagerFactoryBean : Application에서 직접 관리하는 EntityManagerFactory를 생성한다. 설정 파일에서 커넥션등의 정보를 직접 설정한다.

- LocalContainerEntityManagerFactoryBean : Container과 관리하는 EntityManagerFactory를 생성한다.

(1) LocalEntityManagerFactoryBean 설정

: LocalEntityManagerFactoryBean Class를 사용할 경우, Class 패스에 위치한 META-INF/persistence.xml 파일에 Database 연결 및 Mapping 관련 정보가 포함된다.

아래 Code는 JAP 구현체로 EclipseLink를 사용하는 경우의 persistence.xml 파일 예를 보여 주고 있다.

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

<persistence xmlns="http://java.sun.com/xml/ns/persistence"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd"

version="1.0">

<persistence-unit name="store" transaction-type="RESOURCE\_LOCAL">

<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

<class>madvirus.spring.chap10.store.domain.Item</class>

<class>madvirus.spring.chap10.store.domain.PaymentInfo</class>

<class>madvirus.spring.chap10.store.domain.PurchaseOrder</class>

<properties>

<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />

<property name="javax.persistence.jdbc.url"

value="jdbc:mysql://localhost/springbook?characterEncoding=euckr" />

<property name="javax.persistence.jdbc.user" value="root" />

<property name="javax.persistence.jdbc.password" value="" />

</properties>

</persistence-unit>

</persistence>

persistence.xml 파일에 Database 연결등의 정보가 모두 포함되어 있기 때문에, SPRING 설정 파일에서는 EntityManagerFactory와 관련된 JPA 유닛 이름만

알맞게 명시하면 된다. 예를 들어, 위 persistence.xml 파일에서 유닛 이름을 "store"로 설정했는데, 이 경우 SPRING에서는 아래 Code와 같이 persistenceUnitName

프로퍼티를 이용해서 유닛 이름만 알맞게 설정해 주면 LocalEntityManagerFactoryBean 설정이 완료된다.

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">

<property name="persistenceUnitName" value="store" />

</bean>

LocalEntityManagerFactoryBean은 주로 Test 용도나 JPA만을 이용해서 Database에 접근할 때 사용된다.

(2) LocalContainerEntityManagerFactoryBean 설정

: LocalContainerEntityManagerFactoryBean은 JPA의 표준 Container 로딩 규칙에 따라 EntityManagerFactory를 생성해 주는 팩토리 Class이다. 이 Class를 이용해서

EntityManagerFactory를 생성하고 싶다면 아래 Code와 같이 설정하면 된다.

<bean id="jpaAdapter"

class="org.springframework.orm.jpa.vendor.EclipseLinkJpaVendorAdapter">

<property name="databasePlatform"

value="org.eclipse.persistence.platform.database.MySQLPlatform" />

</bean>

<context:load-time-weaver />

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="store" />

<property name="dataSource" ref="dataSource" />

<property name="jpaVendorAdapter" ref="jpaAdapter" />

</bean>

LocalContainerEntityManagerFactoryBean의 주요 프로퍼티는 다음과 같다.

- persistenceUnitName : persistence.xml 파일에 정의된 유닛 이름.

- dataSource : Database 연동을 위한 Adapter.

- jpaVendorAdapter : JPA API 구현체 설정을 위한 Adapter.

- loadTimeWeaver : JPA의 ClassTransformer 규약에 맞게 Class를 조작할 때 사용될 LoadTimeWeaver를 설정한다.

jpaVendorAdapter 프로퍼티는 JPA API 구현체를 위한 설정을 제공하는 JpaVendorAdapter 객체를 설정한다. SPRING은 다음과 같이 네 개의 구현 Class를 제공하고 있다.

이들 Class는 모두 org.springframework.orm.jpa.vendor 패키지에 속해 있다.

- HibernateJpaVendorAdapter : Hibernate EntityManager를 JPA 구현체로 사용할 경우, 관련 설정을 제공한다.

- EclipseLinkJpaVendorAdapter : EclipseLink를 JPA 구현체로 사용할 경우, 관련 설정을 제공한다.

- TopLinkJpaVendorAdapter : TopLink를 JPA 구현체로 사용할 경우, 관련 설정을 제공한다.

- OpenJpaVendorAdapter : OpenJPA를 JPA 구현체로 사용할 경우, 관련 설정을 제공한다.

각각의 JpaVendorAdapter 구현 Class를 설정할 때에는 아래의 프로퍼티를 설정할 수 있다.

프로퍼티 설 명

=====================================================================================

database 연동될 DBMS 종류. 입력 가능한 값은 DEFAULT, DB2, HSQL, INFOMIX, ORACLE, SQL\_SERVER, SYBASE이다.

이들 값은 org.springframework.or,.jpa.vendor.Database 열거형 타입에 명시되어 있다.

=====================================================================================

databasePlatform JPA 벤더에 특정한 Database 관련 설정을 추가한다. 예를 들어, HibernateJpaVendorAdapter의 경우

hibernate.dialect 프로퍼티를 위한 값을 설정하고, TopLinkJpaVendorAdapter의 경우 toplink.target-database

프로퍼티를 위한 값을 설정한다.

Database 열거형 타입에 포함되어 있지 않은 DBMS를 설정하거나 JPA벤더에 좀 더 알맞은 설정을 해야 할 때

사용한다.

=====================================================================================

generateDdl boolean 타입으로, 테이블 생성 여부를 설정한다. 실제 동작 방식은 JPA 벤더에 따라 다르다.

=====================================================================================

showSql boolean 타입으로, 쿼리 출력 관련 여부를 설정한다. 실제 동작 방식은 JPA 벤더에 따라 다를 수 있다.

=====================================================================================

LocalContainerEntityManagerFactoryBean의 loadTimeWeaver 프로퍼티는 JPA의 ClassTranformer 규약에 맞춰 Class를 조작할 때 사용 될 LoadTimeWeaver

객체를 설정한다. Hibernate EntityManager의 경우는 LoadTimeWeaver를 필요로 하지 않으나, EclipseLink나 Oracle TopLink를 이용하는 경우에는

상황에 따라 알맞은 LoadTimeWeaver 객체를 설정해 주어야 한다.

LoadTimeWeaver를 설정하는 가장 간단한 방법은 <context:load-time-weaver /> 태그를 사용하는 것이다.

<context:load-time-weaver />

...

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="store" />

<property name="dataSource" ref="dataSource" />

<property name="jpaVendorAdapter" ref="jpaAdapter" />

</bean>

아래 Code와 같이 loadTimeWeaver 프로퍼티에 알맞은 LoadTimeWeaver 구현체를 등록해 주어도 된다.

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="store" />

...

<property name="loadTimeWeaver">

<bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"/>

</property>

</bean>

JPA 구현체로서 Hibernate EntityManager를 사용하는 경우 LoadTimeWeaver를 필요로 하지 않으므로, <context:load-time-weaver> 태그나

LoadTimeWeaver 프로퍼티를 설정할 필요가 없다.

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="store" />

<property name="dataSource" ref="dataSource" />

<property name="jpaVendorAdapter" ref="jpaAdapter">

<bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">

</bean>

</property>

</bean>

LoadTimeWeaver를 사용하는 경우, 실행 환경에 따라 추가적인 설정이 필요할 수 있다. 예를 들어, Java Application으로 실행한다면 java 실행시

-javaagent 옵션을 이용해서 Java Agent로 org.springframework.instrument-3.0.1.RELEASE.jar 파일을 지정해 주어야 하고, 톰캣을 사용하는 경우

TomcatInstrumentableClassLoader를 Class 로더로 설정해 주어야 한다.

LocalContainerEntityManagerFactoryBean을 사용하는 경우 Database 연동 및 JPA 벤더 설정을 SPRING에서 처리하기 때문에, persistence.xml 파일은

다음과 같이 Mapping 관련 정보만 저장하면 된다.

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

<persistence xmlns="http://java.sun.com/xml/ns/persistence"

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

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd"

version="1.0">

<persistence-unit name="store">

<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

<class>madvirus.spring.chap10.store.domain.Item</class>

<class>madvirus.spring.chap10.store.domain.PaymentInfo</class>

<class>madvirus.spring.chap10.store.domain.PurchaseOrder</class>

</persistence-unit>

</persistence>

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.2 JPA를 위한 Transaction 관리자 설정

: JtaTransactionManager를 사용하지 않는 경우, JpaTransactionManager를 Transaction 관리자로 설정해 주어야 한다. 아래 Code는 JpaTransactionManager를

이용한 Transaction 관리자의 설정 예를 보여주고 있다.

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="store" />

<property name="dataSource" ref="dataSource" />

<property name="jpaVendorAdapter" ref="jpaAdapter" />

</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory" />

</bean>

JpaTransactionManager의 entityManagerFactory 프로퍼티는 SPRING의 Transaction 관리 기능을 연동할 EntityManagerFactory 빈을 설정한다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3.3 EntityManagerFactory와 EntityManager를 DI로 전달받기

: 아래 Code와 같이 일반적인 DI를 이용해서 EntityManagerFactory 객체를 전달 할 수 있다.

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

...

</bean>

<bean id="service" class="..">

<property name="entityManagerFactory" ref="entityManagerFactory"/>

</bean>

EntityManagerFactory를 전달하는 또 다른 방법은 @PersistenceUnit Annotation을 사용하는 것이다.

import javax.persistence.EntityManagerFactory;

import javax.persistence.PersistenceUnit;

public class SomeService {

@PersistenceUnit

private EntityManagerFactory emf;

public void some() {

EntityManager entityManager = emf.createEntityManager();

System.out.println(emf);

entityManager.close();

}

}

@PersistenceUnit Annotation이 적용되려면 PersistenceAnnotationBeanPostProcessor Class를 빈 객체로 등록해 주어야 한다.

<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>

<!--

또는 아래 태그 사용

<context:annotation-config/>

-->

또는, PersistenceAnnotationBeanPostProcessor 대신에 <context:annotation-config/> 태그를 사용해도 된다.

DAO를 구현할 때에는 EntityManagerFactory를 사용하기 보다는 EntityManagerFactory로 부터 구한 EntityManager를 사용하게 된다.

EntityManagerFactory로부터 매번 EntityManager를 가져올 수 있지만, Transaction은 EntityManager 단위로 처리되기 때문에 다음 Code와 같이 DAO의 메서드에서

매번 EntityManager를 생성할 경우 Transaction이 올바르게 처리되지 않게 된다.

@Repository

public class JpaPaymentInfoDao implements PaymentInfoDao {

private EntityManagerFactory emf;

@Override

public void insert(PaymentInfo paymentInfo) {

EntityManager entityManager = emf.createEntityManager();

entityManager.persist(paymentInfo);

...

}

}

EntityManagerTransaction() 메서드를 호출하면 현재 진행중인 Transaction에 EntityManager를 연동할 수 있지만, 그것보다는 아래 Code와 같이 @PersistenceContext

Annotation을 이용해서 EntityManager를 사용하는 것이 Transaction 연동을 더 쉽게 처리할 수 있다.

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

@Repository

public class JpaPaymentInfoDao implements PaymentInfoDao {

@PersistenceContext

private EntityManager entityManager;

@Override

public void insert(PaymentInfo paymentInfo) {

entityManager.persist(paymentInfo);

...

}

}

@PersistenceContext Annotation을 적용하면 EntityManager는 실제로 현재 진핸중인 Transaction과 연동된 EntityManager를 사용하게 된다. (실제 내부적으로는

프록시 객체가 @PersistenceContext Annotation이 적용된 프로퍼티에 할당되고, 해당 프록시를 통해서 실제 Transaction과 연결된 EntityManager에 접근하게 된다.)

@PersistenceContext Annotation이 올바르게 동작하려면 @PersistenceUnits Annotation과 마찬가지로 PersistenceAnnotationBeanPostProcessor Class나

<context:annotation-config/> 태그를 설정 파일에 등록해 주어야 한다.

===================================================================================================

4. iBATIS 연동 지원

: iBATIS는 Database 테이블과 Java 객체 사이의 단순한 Mapping을 비교적 간단한 설정을 통해 처리할 수 있기 때문에, 널리 사용되는 Framework 중의 하나로

자리잡고 있다. 하지만, iBATIS에서 Database 연동을 처리할 때 사용되는 SqlMapClient 역시 JDBC 프로그램을 할때와 마찬가지로 try~catch 블록을 사용해서

예외를 처리해 주어야 한다. 또한, 에러 원인에 따라 알맞은 예외를 발생시키기 보다는 SQLException을 발생하고 있다.

SPRING은 SqlMapClient를 사용할 때 발생하는 Code 중복을 없애고 SQLException을 SPRING이 제공하는 예외 Class로 변환해 주는 SqlMapClientTemplate Class를

제공하고 있다. 본 절에서는 SPRING에서 iBATIS를 연동하기 위해 SqlMapClientTemplate Class를 사용하는 방법을 살펴보도록 하겠다.

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4.1 SqlMapClientFactoryBean을 이용한 SqlMapClient 설정

: SPRING이 제공하는 SqlMapClientTemplate Class는 내부적으로 iBATIS의 SqlMapClient를 사용한다. SPRING은 SqlMapClient를 SPRING 설정 파일에서 쉽게 설정할 수

있도록 돕는 SqlMapClientFactoryBean Class를 제공하고 있다. 이 Class를 사용하면 SqlMapClient를 SPRING 빈으로 설정할 수 있다. 아래 Code는 SqlMapClientFactoryBean

Class의 설정 예이다.

<bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean"

p:dataSource-ref="dataSource"

p:configLocation="classpath:sqlMapConfig.xml" />

dataSource 프로퍼티에는 DataSource를 전달하며, configLocation 프로퍼티는 iBATIS 설정 파일을 명시한다. 아래 Code는 iBATIS 설정 파일의 예이다.

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

<!DOCTYPE sqlMapConfig

PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"

"http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

<settings useStatementNamespaces="true" />

<sqlMap resource="madvirus/spring/chap10/store/dao/ibatis/Item.xml" />

<sqlMap resource="madvirus/spring/chap10/store/dao/ibatis/PaymentInfo.xml" />

<sqlMap resource="madvirus/spring/chap10/store/dao/ibatis/PurchaseOrder.xml" />

</sqlMapConfig>

iBATIS 설정 파일의 <sqlMap> 태그는 Mapping 설정 파일을 명시한다. 아래 Code는 iBATIS Mapping 설정 파일의 일부를 보여주고 있다.

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

<!DOCTYPE sqlMap

PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"

"http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Item">

<typeAlias alias="Item" type="madvirus.spring.chap10.store.domain.Item" />

<resultMap id="itemResultMap" class="Item">

<result property="id" column="ITEM\_ID" />

<result property="price" column="PRICE" />

</resultMap>

<select id="findById" resultMap="itemResultMap" resultClass="Item"

parameterClass="java.lang.Integer">

select \* from ITEM where ITEM\_ID = #value#

</select>

</sqlMap>

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4.2 SqlMapClientTemplate을 이용한 iBATIS 연동

: SqlMapClient를 위한 빈을 설정했다면 SqlMapClientTemplate을 이용해서 iBATIS를 연동할 수 있다. SqlMapClientTemplate은 다음과 같이 sqlMapClient

프로퍼티를 통해서 SqlMapClient를 설정한다.

<bean id="sqlMapClient"

class="org.springframework.orm.ibatis.SqlMapClientFactoryBean"

p:dataSource-ref="dataSource"

p:configLocation="classpath:sqlMapConfig.xml" />

<bean id="sqlMapClientTemplate"

class="org.springframework.orm.ibatis.SqlMapClientTemplate"

p:sqlMapClient-ref="sqlMapClient" />

<bean id="itemDao" class="madvirus.spring.chap10.store.dao.ibatis.IBatisItemDao"

p:sqlMapClientTemplate-ref="sqlMapClientTemplate" />

SqlMapClientTemplate Class를 설정했다면 DAO Class에서 SqlMapClientTemplate Class를 사용해서 iBATIS를 사용할 수 있다. 아래 Code는 SqlMapClientTemplate을

이용하는 DAO Class의 구현 예이다.

import org.springframework.orm.ibatis.SqlMapClientTemplate;

public class IBatisItemDao implements ItemDao {

private SqlMapClientTemplate sqlMapClientTemplate;

public void setSqlMapClientTemplate(

SqlMapClientTemplate sqlMapClientTemplate) {

this.sqlMapClientTemplate = sqlMapClientTemplate;

}

@Override

public Item findById(Integer itemId) {

return (Item) sqlMapClientTemplate.queryForObject("Item.findById",

itemId);

}

}

SqlMapClientTemplate Class는 iBATIS의 SqlMapClient Class가 제공하는 대부분의 메서드와 동일한 이름과 파라미터 타입, 리턴 타입을 정의하고 있다.

차이점이 있다면 SQLException을 발생하는 대신 SPRING이 제공하는 예외를 발생한다는 점이다.

iBATIS의 SqlMapExecutor를 직접 사용하고 싶다면, 다음 Code와 같이 SqlMapClientCallback Interface의 구현 객체를 전달 받는 execute() 메서드를 사용하면 된다.

sqlMapClientTemplate.execute(new SqlMapClientCallback<Item>(){

@Override

public Item doInSqlMapClient(SqlMapExecutor executor) throws SQLException{

//..

return someValue;

}

});

SqlMapClientCallback 구현 객체의 doInSqlMapClient() 메서드에 SqlMapExecutor 파라미터가 전달되므로, 직접 SqlMapExecutor를 사용해서 Database

연동을 구현할 수 있다.

4.3 SqlMapClientDaoSupport Class를 이용한 DAO Class 구현

: SPRING은 SqlMapClientTemplate Class를 DAO Class에서 좀 더 쉽게 사용할 수 있도록 하기 위해 SqlMapClientDaoSupport Class를 제공한다. 이 Class를

상속받은 Class는 다음과 같이 sqlMapClientDaoSupport 프로퍼티를 통해서 SqlMapClientDaoSupport를 전달 받는다.

<bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean"... />

<bean id="sqlMapClientTemplate"

class="org.springframework.orm.ibatis.SqlMapClientTemplate"

p:sqlMapClient-ref="sqlMapClient" />

<bean id="paymentInfoDao"

class="madvirus.spring.chap10.store.dao.ibatis.IBatisPaymentInfoDao"

p:sqlMapClientTemplate-ref="sqlMapClientTemplate" />

설정 파일을 좀 더 간결하게 하려면 다음과 같이 sqlMapClient 프로퍼티에 직접 SqlMapClient 빈을 전달함으로써 sqlMapClientTemplate을 위한 설정을 줄일 수 있다.

<bean id="sqlMapClient"

class="org.springframework.orm.ibatis.SqlMapClientFactoryBean"

p:dataSource-ref="dataSource"

p:configLocation="classpath:sqlMapConfig.xml" />

<bean id="paymentInfoDao"

class="madvirus.spring.chap10.store.dao.ibatis.IBatisPaymentInfoDao"

p:sqlMapClient-ref="sqlMapClient" />

SqlMapClientDaoSupport Class를 상속받은 Class는 getSqlMapClientTemplate() 메서드를 사용해서 SqlMapClientTemplate을 구할 수 있다. 아래 Code는 구현 예이다.

import org.springframework.orm.ibatis.support.SqlMapClientDaoSupport;

public class IBatisPaymentInfoDao extends SqlMapClientDaoSupport implements PaymentInfoDao {

@Override

public void insert(PaymentInfo paymentInfo) {

getSqlMapClientTemplate().insert("PaymentInfo.insert", paymentInfo);

}

}