하이버네이트, JPA프로젝트

강사 : 강병준

Hibernate

1. ORM 개념은 오래 전부터 나왔는데 처음의 시초는 EJB Entity Beans라고 할 수 있는데 EJB는 여러 가지 강력한 기능을 가졌음에도 불구하고 사용법이 너무 어려워서 근래에는 거의 사용되지 않습니다.
2. 스프링 JDBC는 JDBC를 매핑해서 SQL문을 이용한 소스 코드를 단순하게 유지해 주지만 때로는 SQL문을 사용하지 않고 싶을 때도 있다
3. 사용법은 편하게 하면서도 더 많은 기능을 지원하게 한 것이 ORM인데 SQL문을 쓰지 않고 ORM을 사용하는 중에서 대표적인 것이 Hibernate
4. 자바 객체를 RDBMS의 하나의 ROW로 맵핑
5. Hibernate가 개발된 후에 Java 스펙에도 ORM의 표준 스펙을 정의했는데 이것이 JPA (Java Persistence API)
6. JPA는 하나의 스펙으로 JPA에 대한 구현체가 필요합니다.
7. 실제 구현체는 TopLink, OpenJPA등 여러가지가 있는데, 이렇게 ORM API를 JPA를 통해서 추상화 시켜놓음으로써, 다른 ORM과 쉽게 교체가 가능하게 만든 개념이지만, 실제로 ORM은 거의 Hibernate만 사용됩니다.

JPA(Java Persistence API)와 O/R 매핑

- **JPA(Java Persistence API)와 O/R 매핑**
  - JPA는 자바 표준 O/R매핑 사양이다. 처음에는 EJB사양으로 시작했으나 EJB와 분리하여 독자적으로 사용되었다. JPA는 사양이며 구현된 것이 Hibernate와 탑링크라는 제품이다.
  - ORM(Object Relational Mapping)은 객체지향언어에서 취급하는 클래스와 관계형데이터베이스인 테이블과 단순히 1:1로 대응하지 않는다
  - OR매핑은 다른 것을 대응시켜 주는 것
  - JPA는 기본적으로 JBoss나 Weblogic과 같은 JEE 컨테이너를 가정으로 개발되어서 JTS/JTA, EJB3등과 사용하기에는 좋지만 컨테이너 없이 Tomcat과 같은 Servlet 컨테이너만 사용할 경우 큰 이득을 보기 어렵습니다.
- **하이버 네이트**
  - 무료로 제공되는 O/R매핑툴로서 http://www.hibernate.org/ 에서 다운받을 수 있다

Hibernate

1. Hibernate는 자체적으로 쿼리를 생성하고 OR Mapper로써 객체들을 DB에서 로딩할 때 레퍼런스 된 객체 등을 로딩하는 등,제대로 특성을 이해하고 사용하지 않으면 장애를 일으키는 경우가 많기 때문에 데이터베이스에 대한 전문적인 지식을 갖추고 사용해야 하므로 SI에서는 기피하고 솔루션 개발업체에서는 선호합니다.

2. Hibernate의 장점
   1) 복잡한 JDBC 코드의 제거
   2) 성숙된 ORM 기능 제공
   3) 기존 데이터베이스와 연동
   4) Spring Framework와 통합기능 제공
   5) 성능이 우수

Hibernate

1. Hibernate
   1) 객체 <--도메인설계,일부HQL--> DB테이블/뷰
   2) 장점: 자동화. ER/클래스 다이어그램에 의한 자동 JOIN/Cascade, Middlegen, MyEclipse 등 자동화 툴 존재
   3) 단점: 기술적 난이도 높음(아키텍쳐 이해, 설계, 트랜젝션, 캐쉬구성, HQL 등등)
2. iBatis
   1) 객체 <--sql구현--> DB테이블/뷰
   2) 장점: 기술적 난이도 낮음.
   3) 단점: 효율이 좋다고 말할 수 없음.
3. SI업체,이미 DB가 존재, DB설계가 엉망 인 경우 --> iBatis(Mybatis)
4. 제품계발, 신규개발, 다양한DB에서 동작, 제대로 된 DB설계(ER다이어그램, 클래스/도메인 다이어그램) 인 경우 --> Hibernate

Hibernate

Hibernate 프레임워크 구성

1. Configuration 객체: 데이터베이스 연결과 클래스 매핑 설정
2. SessionFactory 객체: Configuration 객체를 이용해서 생성
   1) 데이터베이스 사용을 위한 Session 객체를 생성하기 위한 객체
   2) 무거운 객체이므로 애플리케이션 시작 시에 생성하고 유지
   3) 데이터베이스 당 하나의 객체를 구성 파일을 사용하여 생성
3. Session: 물리적인 데이터베이스 연결 객체
   가벼운 객체이므로 필요할 때 마다 생성하여 사용하고 소멸 시킬 수 있음
4. Transaction: 트랜잭션 처리를 위한 객체
5. Query: SQL 또는 HQL을 사용하여 데이터베이스에서 데이터를 가져오거나 생성함
6. Criteria: Criteria 질의를 사용하여 데이터베이스에서 데이터를 가져오거나 생성

Hibernate

매핑 타입

1. Integer: int, java.lang.Integer
2. Long: long, java.lang.Long
3. Double: double, java.lang.Double
4. Boolean, yes/no. true/false: boolean, java.lang.Boolean
5. Date: java.util.Date, java.sql.Date
6. Time: java.util.Date, java.sql.Time
7. Timestamp: java.util.Date, java.sql.Timestamp
8. Image: java.lang.byte[]
9. String, Char, Varchar, Text: java.lang.String
10. Blob: java.sql.Blob

Hibernate

매핑 방식

Hibernate 매핑 XML 이용(id로 PK 매핑, property로 컬럼 매핑)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="ch04.Person" table="person">
<id name="id" column="id" type="java.lang.String" />
<property name="name" column="name" type="java.lang.String" />
</class>
</hibernate-mapping>
```

질의 방식

1. SQL 질의

```
select * from product where product_id=:id
```

2. HQL 질의

(select 절을 사용하지 않고 테이블 이름 대신 매핑된 클래스 이름을 대입)

```
from ProductEntity where id=:id
```

3. Criteria 질의

```
Criterion eqCriterion = Restriction.eq("id", id);
```

Hibernate

Session 객체의 주요 메서드

1. Transaction beginTransaction()
2. SQLQuery createSQLQuery(String queryString)
   =>list(), executeUpdate()를 호출
3. Transaction getTransaction()
4. Object get(clazz, 조건)
5. save(Object): 데이터 삽입 – 삽입 후 아이디로 설정된 값 리턴
6. update(Object)
7. delete(Object)

이벤트 검색하기

O/R 맵핑 툴의 하나인 Hibernate는 HQL에 의한 질의한다.
HQL이란
- Hibernate Query Language(Hibernate 쿼리 언어)의 약어
- 하이버네이트를 통해 데이터베이스를 검색할 경우에 이용하는 쿼리
- HQL을 하이버네이트가 어떤 데이터베이스를 이용하고 있는지 살펴보고 알아서 그에 맞는 SQL로 변환하여 준다는 장점이 있다.
- HQL은 SQL에 닮아 있지만 클래스의 프로퍼티를 쓴다는 특징이 있다.

▼ HibernateTemplate 클래스에 정의되어 있는 주요 메소드

메소드	설명
Object get (Class entityClass, Serializable id)	• 지정한 클래스에 해당하는 ID를 갖는 클래스를 취득한다. • 그 ID를 갖는 데이터가 존재하지 않는 경우는 null이 반환된다.
Object load (Class theClass, Serializable id)	Get ( ) 메소드와 동일하지만 존재하지 않는 경우에 예외를 스로한다.
List loadAll(Class entityClass)	지정한 클래스의 오브젝트를 모두 취득한다.
Serializable save(Objectentity)	오브젝트를 영속화하기 위해 하이버네이트가 생성한 식별자를 반환한다.
void update(Object entity)	오브젝트를 갱신한다.
void saveOrUpdate(Objectentity)	오브젝트를 영속화 또는 갱신한다.
void saveOrUpdateAll (Collection entities)	커렉션에 저장된 오브젝트 모두에 saveOrUpdate( )을 실행한다.
void delete(Object entity)	오브젝트를 삭제한다.
List find(String queryString, Object[ ] values)	HQL에서 쓰인 쿼리 내의 '?' 에 값을 세팅하여 쿼리를 실행한다. 해당하는 클래스가 존재하지 않는 경우는 비어 있는 List(요소수가 0)가 반환된다.

JPA 설정파일

- 애플리케이션이 JPA를 이용하는 경우 클래스가 설정되어 있는 폴더의 바로 밑에 META-INF를 폴더로 만들고 JPA설정 파일인 persistence.xml을 만든다
  1) 영속성 유닛단위로 이름을 붙임
  2) <provider>요소에는 JPA를 구현한 각 제품이 제공하는 프로바이더 클래스 지정
  3) <class> 요소는 영속화할(데이터베이스에서 관리하는 대상이 되는) 클래스를 패키지 경로 포함 명칭으로 지정
  4) <properties>요소에 데이터베이스 접소 설정과 같은 정보 기록

persistence.xml에서 설정하는 프로퍼티

프로퍼티명	값의 예	설명
hibernate.dialect	Org.hibernate.dialect.MySQLDialect	하이버네이트가 제공하는 접속 대상인 데이터베이스네 맞는 Dialect클래스 지정
Hibernate.connection.driver_class	com.mysqljdbc.Driver	JDBC에서 드라이버 클래스의 패키지 경로 포함 명칭
Hibernate.connection.username	springuser	데이터베이스 접근 유저명
Hibernate.connection.password	springpassword	데이터베이스 패스워드
Hibernate.connection.url	Jdbc;mysql://localhost/spring	JDBC를 이용하기 위한 URL
Hibernate.show_sql	True	하이버네이트가 실행하는 SQL을 표시할지 지정. True면 발행한 SQL을 로그 출력

테이블 생성

```
create table person (
    id  varchar2(10) primary key,
    name varchar2(12)
)
```

log4j.properties

```
log4j.rootLogger=INFO, CA
log4j.appender.CA=org.apache.log4j.ConsoleAppender
log4j.appender.CA.layout=org.apache.log4j.PatternLayout
log4j.appender.CA.layout.ConversionPattern=[%p:%c{1}] %m%n
```

sample.biz.domain/Person

```
package sample.biz.domain;
import javax.persistence.Column;      import javax.persistence.Entity;
import javax.persistence.Id;
@Entity    //  JPA에 영속화를 하는 도메인 클래스
public class Person {
    private String id;       private String name;
    public Person() {    }
    public Person(String id, String name) {
        this.id = id;         this.name = name;
    }
    @Id  // primary key를 나타내는 필드 또는 그 필드의 getter메소드에 표시
    @Column(name = "id")   // 테이블의 컬럼에 대응하는 필드 또는
      // getter 메소드에 부여하고 name요소에 컬럼값 부여, 이름이 같으면 생략 가능
    public String getId() {       return id;    }
    public void setId(String id) {       this.id = id;    }
    public String getName() {        return name;    }
    public void setName(String name) {    this. name = name;    }
}
```

도메인 클래스

- 제약 조건
  - 인수 없는 생성자가 있을 것
  - 프로퍼티에 대한 엑세스 메소드 있을 것(필수는 아님)
  - Pk의 프로퍼티와 엑세스 메소드가 있을 것(필수는 아니지만 권장)

- 하이버네이트를 이용해 도메인 생성하고 갱신하는 절차
  - 비즈니스 클래스는 데이터 엑서스 오브젝트(DAO)에 엑세스
  - DAO는 하이버네이트가 제공하는 SessionFactory오브젝트로부터 세션 오브젝트를 가져오고 세션오브젝트를 이용해 도메인을 생성하고 갱신

HQL

하이버네이트는 sql과 비슷한 hql을 사용한다

sql	**hql**
Select * from emp	From emp (emp는 클래스 이름)
Select * from emp where id='1'	From emp where id='1 (id는 컬럼이 아닌 프로퍼티 이름)

Hibernate 제3라이브러리

라이브러리	설명
antlr (필수)	Hibernate는 질의 파서들을 산출하는데 ANTLR을 사용하고, 이 라이브러리는 또한 실행 시에 필요하다.
dom4j (필수)	Hibernate는 XML 구성과 XML 매핑 메타데이터 파일들을 파싱하는데 dom4j를 사용 한다.
CGLIB, asm (필수)	Hibernate는 (Java reflection과 결합하여) 런타임 시에 클래스들을 고양시키는데 코드 생성 라이브러리를 사용한다.
Commons Collections, Commons Logging (필수)	Hibernate는 Apache Jakarta Commons 프로젝트로부터 다양한 유틸리티 라이브러리 들을 사용한다.
EHCache (필수)	Hibernate는 second-level 캐시를 위한 다양한 캐시 프로바이더들을 사용할 수 있다. 만일 구성에서 변하지 않을 경우 EHCache가 디폴트 캐시 프로바이더이다.
Log4j (옵션)	Hibernate는 기본 로깅 메커니즘으로서 Log4j를 사용할 수 있는, Commons Logging API를 사용한다. 만일 Log4j 라이브러리가 컨텍스트 라이브러리 디렉토리 속에서 이용 가능하다면, Commons Logging은 Log4j와 컨텍스트 classpath 내에 있는 `log4j.properties` 구성을 사용할 것이다. Log4j에 대한 예제 properties 파일은 Hibernate 배포본에 번들화 되어 있다. 따라서 당신이 이면에서 무엇이 진행되는 지을 보고자 원할 경우에 log4j.jar와 (`src/`에 있는) 구성 파일을 당신의 컨텍스트 classpath 속으로 복사하라.
필수 여부?	Hibernate 배포본 내에 있는 `lib/README.txt` 파일을 살펴보라. 이것은 Hibernate에 배포된 제 3의 라이브러리들의 최신 목록이다. 당신은 그곳에 열거된 모든 필수 라이브러리들과 옵션 라이브러리들을 찾게 될 것이다(여기서 "빌드 시 필요함"은 당신의 어플리케이션이 아니라 Hibernate에 대한 의미임을 노트하라).

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion><groupId>ch03</groupId>
  <artifactId>ch03</artifactId><version>0.0.1-SNAPSHOT</version>
  <properties>
     <org.springframework.version>4.1.1.RELEASE</org.springframework.version>
     <hibernate.version>3.6.9.Final</hibernate.version>
  </properties>
  <repositories>
     <repository>
        <id>codelds</id>
       <url>https://code.lds.org/nexus/content/groups/main-repo</url>
    </repository>
   </repositories>
  <dependencies>
  <dependency>
     <groupId>com.oracle</groupId><artifactId>ojdbc6</artifactId>
    <version>11.2.0.3</version><scope>compile</scope>
</dependency>
<dependency>
   <groupId>org.springframework</groupId><artifactId>spring-jdbc</artifactId>
   <version>${org.springframework.version}</version>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${org.springframework.version}</version>
</dependency>
<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjweaver</artifactId>
   <version>1.6.8</version>
</dependency>
<!-- hibernate -->
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-entitymanager</artifactId>
   <version>${hibernate.version}</version>
</dependency>
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>${hibernate.version}</version>
</dependency>
<dependency>
   <groupId>javassist</groupId>
   <artifactId>javassist</artifactId>
   <version>3.9.0.GA</version>
</dependency>
```

```
<dependency>
   <groupId>cglib</groupId>
   <artifactId>cglib</artifactId>
   <version>2.2</version>
</dependency>
<dependency>
   <groupId>antlr</groupId>
   <artifactId>antlr</artifactId>
   <version>2.7.6</version>
</dependency>
<dependency>
   <groupId>dom4j</groupId>
   <artifactId>dom4j</artifactId>
   <version>1.6.1</version>
</dependency>
<dependency>
   <groupId>javax.transaction</groupId>
   <artifactId>jta</artifactId>
   <version>1.1</version>
   <type>jar</type>
   <scope>compile</scope>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-orm</artifactId>
   <version>${org.springframework.version}</version>
</dependency>
```

```
<dependency>
    <groupId>commons-collections</groupId>
    <artifactId>commons-collections</artifactId>
    <version>3.1</version><scope>provided</scope>
</dependency>
<!-- Logging -->
<dependency>
   <groupId>org.slf4j</groupId><artifactId>slf4j-api</artifactId>
   <version>1.5.10</version><type>jar</type><scope>compile</scope>
</dependency>
<dependency>
   <groupId>org.slf4j</groupId><artifactId>jcl-over-slf4j</artifactId>
   <version>1.5.10</version><type>jar</type><scope>runtime</scope>
</dependency>
<dependency>
   <groupId>org.slf4j</groupId><artifactId>slf4j-log4j12</artifactId>
   <version>1.5.10</version><scope>runtime</scope>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>
</dependencies>
</project>
```

applicationContext.hibernate.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:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
<context:property-placeholder location="jdbc.properties"/>
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
        destroy-method="close">
        <property name="driverClass" value="${jdbc.driverClassName}" />
        <property name="jdbcUrl" value="${jdbc.url}" />
        <property name="user" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
        <property name="maxPoolSize" value="${jdbc.maxPoolSize}" />
    </bean>
```

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mappingResources" value="sample/dao/Person.hbm.xml"/>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</prop>
            <prop key="hibernate.show_sql">false</prop>
        </props>
    </property>
</bean>
<bean class="org.springframework.orm.hibernate3.HibernateTemplate">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
<aop:config>
    <aop:advisor advice-ref="txAdvice"
        pointcut="execution(* *..*Service*.*(..))" />
</aop:config>
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />
        <tx:method name="add*" propagation="REQUIRED"/>
        <tx:method name="remove*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>
<context:component-scan base-package="sample"/>
</beans>
```

sample.dao/Person.hbm.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="sample.biz.domain.Person" table="PERSON" >
        <id name="id" column="ID" type="java.lang.String" />
        <property name="name" column="NAME" type="java.lang.String" />
    </class>
</hibernate-mapping>
```

sample.biz.dao/PersonDao

```
package sample.biz.dao;

import java.util.List;

import sample.biz.domain.Person;

public interface PersonDao {
    List<Person> findPerson();
    void addPerson(Person person);
    void removePerson(Person person);
    void updatePerson(Person person);
}
```

하이버 네이트 기능

- 지연로드
  - 필요할 때까지 오브젝트를 로드하지 않아 리소스 절약과 성능 향상을 지원

- 캐시
  - 일단 로드된 오브젝트를 캐시하여 테이블 엑세스 회수를 줄이는 등 리소스 절약과 성능 향상

- 스프링과 연계
  - 스프링과 연계하여 세션 열기/닫기 예외처리를 스프링에 맡겨서 처리

sample.dao/ HibernatePersonDao

```
package sample.dao;          import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.stereotype.Repository;
import sample.biz.dao.PersonDao;import sample.biz.domain.Person;
@Repository // 데이터 엑세스층의 DAO용으로 Autowired가 붙은 인스턴스변수나
            // 메소드에 인젝션 함
public class HibernatePersonDao implements PersonDao {
    @Autowired  // 자동으로 의존관계 설정하기 때문에 설정파일 제거
                // Autowired를 사용하려면 ApplicationContext사용
    private HibernateTemplate hibernateTemplate;
    public List<Person> findPerson() {  return hibernateTemplate.find("from Person");    }
    public void addPerson(Person person) {
        hibernateTemplate.save(person);   hibernateTemplate.flush();
    }
    public void removePerson (Person person) {
        hibernateTemplate.delete(person);        hibernateTemplate.flush();
    }
    public void updatePerson(Person person) {
        hibernateTemplate.update(person);        hibernateTemplate.flush();
    }
}
```

sample.biz.service/PersonService

```
package sample.biz.service;

import java.util.List;

import sample.biz.domain.Person;

public interface PersonService {
    List<Person> findPerson();
    void addPerson(Person person);
    void updatePerson(Person person);
    void removePerson(Person person);
}
```

sample.biz.service.impl/PersonServiceImpl

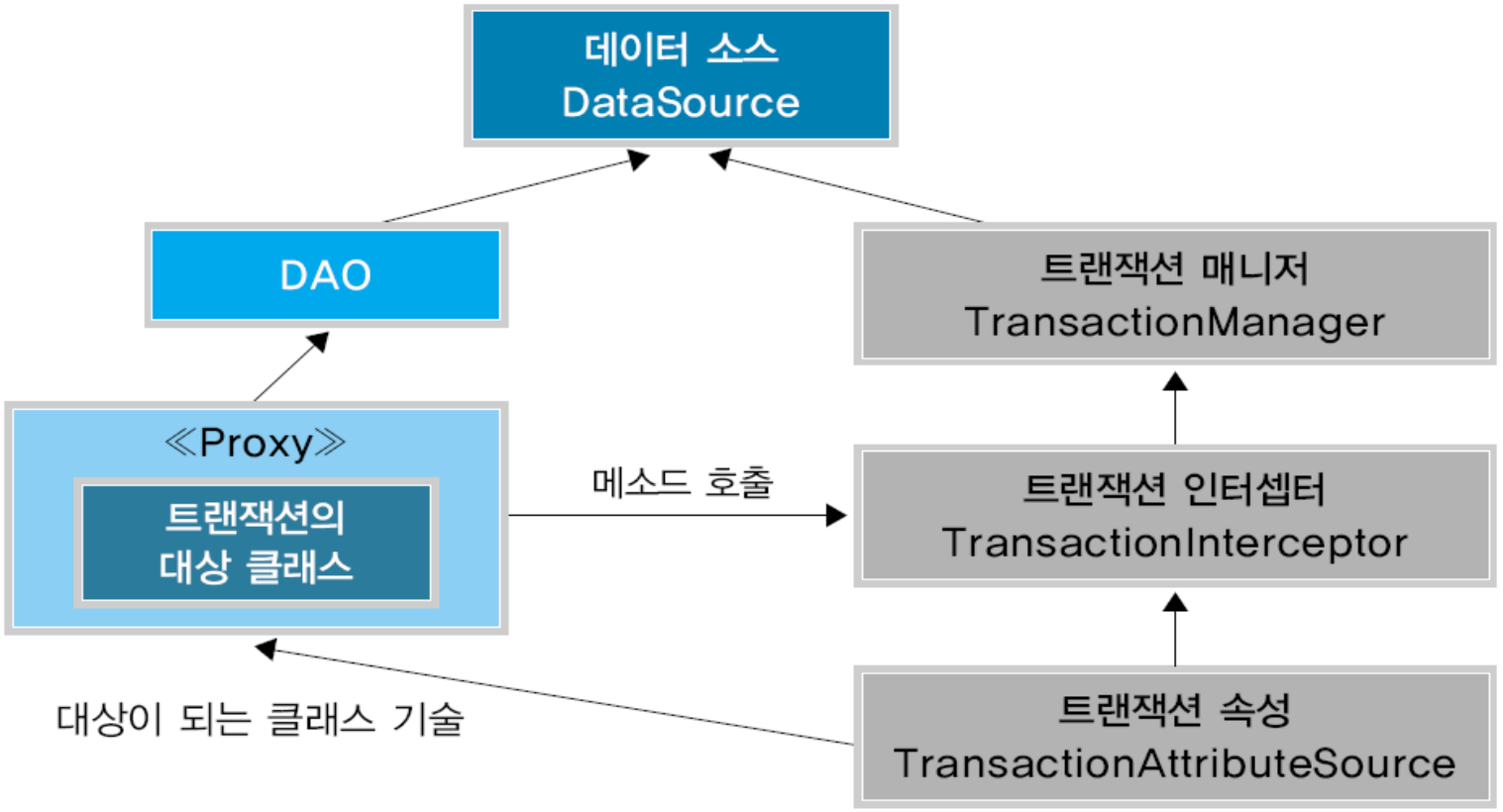
```
package sample.biz.service.impl;
import java.util.List;                    import org.hibernate.HibernateException;
import org.hibernate.Session;             import org.hibernate.SessionFactory;
import org.hibernate.Transaction;         import org.hibernate.cfg.Configuration;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import sample.biz.dao.PersonDao;      import sample.biz.domain.Person;
import sample.biz.service.PersonService;          import sample.dao.HibernatePersonDao;
@Service
public class PersonServiceImpl implements PersonService {
    @Autowired
    private PersonDao personDao;
    public List<Person> findPerson() {          return personDao.findPerson();     }
    public void addPerson(Person person) {          personDao.addPerson(person);     }
    public void removePerson(Person person) {     personDao.removePerson(person);     }
    public void updatePerson(Person person) {      personDao.updatePerson(person);     }
}
```

sample/Main

```
package sample;                import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import sample.biz.domain.Person;
import sample.biz.service.PersonService;
public class Main {
    public static void main(String[] args) {
        ApplicationContext ctx =       new
           ClassPathXmlApplicationContext("applicationContext.hibernate.xml");
        PersonService personService = ctx.getBean(PersonService.class);
        Person person = new Person();
        person.setId("person001");       person.setName("홍길동");
        personService.addPerson(person);
        List<Person> list = personService.findPerson();
        System.out.println(list.get(0).getName());
        person.setName("foo");
        personService.updatePerson(person);
        personService.removePerson(person);
    }
}
```

트랜잭션의 설정 관계

- 적용된 오브젝트의 메소드가 호출될 때, 다음 그림과 같이 각 메소드는 설정된 트랜잭션속성으로 실행된다.



▲ 트랜잭션의 설정 관계

Transactional 어노테이션

요소명	타입	기본값	설명
value	String	-	사용할 트랜잭션 매니저를 명시저으로 지정할 때 사용
propagation	Propagation	Propagation.REQUIRED	트랜잭션 속성
isolation	Isolation	Isolation.DEFAULT	트랜잭션 분리 레벨
readOnly	Boolean	false	읽기 전용은 true
timeOut	int	이용할 트랜잭션 시스템의 기본값	타입아웃시킬 초 수를 지정
rollbackFor	Class배열	-	롤백 대상이 되는 클래스
rollbackForClassname	클래스명 문자열배열	-	롤백 대상이 되는 클래스
noRollbackFor	Class배열	-	롤백 대상에서 제외되는 클래스
noRollbackForClassname	클래스명 문자열 배열	-	롤백 대상에서 제외되는 클래스

트랜잭션 속성

속성명	의미
PROPAGATION_REQUIRED	• 현재 트랜잭션이 존재한다면 계속 유지하며, 존재하지 않으면 새로운 트랜잭션을 생성한다. • EJB의 Required와 동일하며 통상 트랜잭션의 정의는 default로 설정한다.
PROPAGATION_SUPPORTS	• 현재 트랜잭션이 존재한다면 계속 유지하며, 존재하지 않으면 트랜잭션을 이용하지 않고 실행한다. • EJB의 Supports와 동일하다.
PROPAGATION_MANDATORY	• 현재 트랜잭션이 존재한다면 실행한다. 존재하지 않으면 예외를 던진(throw)다. • EJB의 Mandatory와 동일하다.
PROPAGATION_REQUIRES_NEW	• 항상 새로운 트랜잭션을 생성하며 이미 트랜잭션이 있는 경우는 그 트랜잭션은 중단된다. • EJB의 Requires New와 동일하다.

트랜잭션 속성

속성명	의미
PROPAGATION_NOT_SUPPORTED	• 항상 트랜잭션을 이용하지 않고 실행한다. • 이미 트랜잭션이 있는 경우는 그 트랜잭션은 중단된다. • EJB의 NotSupported와 동일하다.
PROPAGATION_NEVER	• 항상 트랜잭션을 이용하지 않고 실행하며, 만약 트랜잭션이 이미 있다면 예외를 던진(throw)다. • EJB의 Nerver와 동일하다.
PROPAGATION_NESTED	• 현재 트랜잭션이 존재한다면 nested한 트랜잭션을 생성하고, 존재하지 않는다면 PROPAGATION_REQUIRED와 동일하게 움직인다. • EJB에는 없는 속성이다.

트랜잭션 매니저

1. 트랜잭션 매니저는 스프링이 제공하는 트랜잭션 처리를 위한 부품. 트랜잭션의 시작과 종료로 콜백처리를 비롯해 트랜잭션의 정의정보(롤백의 조건이나 독립성 레벨 등)를 세밀하게 설정.
2. 데이터 액세스 기술(JDBC, 하이버네이트, 독립성 레벨)을 은폐하여 데이터 액세스 기술과 관계없이 트랙잭션 매니저 할용

트랜잭션 정의 정보

설정값	기본값	기본값 비교
1. 전파속성	PROPAGATION_REQUIRED	
2. 독립성 수준	ISOLATION_DEFAULT	데이터베이스의 기본 독립성 수준을 적용
3. 시간만료	-1	시간 만료없음
4. 읽기전용 상태	false	
5. 록백 대상 예외	설정하지 않음	실행시 예외가 롤백 대상
6. 커밋대상예외	설정하지 않음	검사 예이가 롤벡 대상

1. 전파속성 : 트랙잰션의 전파방법을 설정하는 속성
2. 독립성 수준 : 트랙적션 처리가 병행해서 실행될 때의 각 독립성을 결정하는 것
3. 시간 만료 : 트랙잰션이 취소되는 시간만료 시간을 초단위로 설정
4. 읽기전용 상태 : 트랜잭션 내의 처리가 읽기전용인지 설정,
   설정에 의해 DB나 ORM프레임워크쪽에서 최적화가 이루어진다
5. 롤백대상 예외 : 예외가 던져졌을 때 롤백할지 설정
6. 커밋대상 예외 : 예외가 던져졌을 때 커밋할지 설정

트랜잭션 정보 설정

1. 트랜잭션 정의정보설정(기본값)

```
<tx:advice id="transactionAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="*" />
        </tx:attributes>
    </tx:advice>
```

2. 트랜잭션 정의 정보 설정

```
<tx:advice id="transactionAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="get*" read-only="true" />
            <tx:method name="update*"
                propagation="REQUIRED"
                isolation="READ_COMMITTED"
                timeout="10"
                read-only="false"
                rollback-for="sample.biz.exception.BusinessException" />
        </tx:attributes>
    </tx:advice>
```

트랜잭션 처리하는 클래스 인터페이스 지정

```
<aop:config>
        <aop:advisor advice-ref="transactionAdvice" pointcut="execution(* *..*Service.*(..))" />
    </aop:config>
```