제7장 스프링 데이터(1)

- Spring과 DB
- Spring의 트랜잭션 관리
- Spring과 ORM

Spring과 DB

- 순수 **JDBC** 코드
- Spring의 데이터 접근 방식
- Spring의 데이터 소스 설정
- Spring의 **JDBC** 사용

순수 JDBC 코드(1/2)

- JDBC
  - 자바의 데이터 엑세스 기술의 기본이 되는 로우 레벨의 API
  - 표준 인터페이스 제공
    - SQL 호환성만 유지된다면 DB 변경 시 JDBC 코드의 변경 없이 재사용 가능

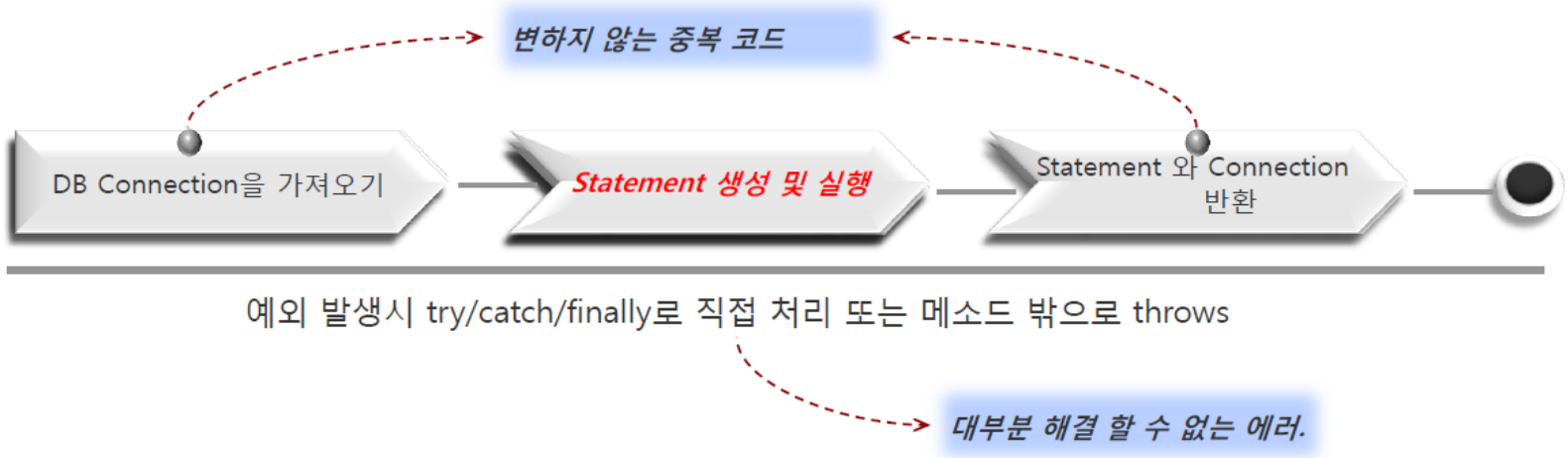
```
public void registerBoard(Board board) throws SQLException {
      Class.forName("com.mysql.jdbc.Driver");
      Connection c = DriverManager.getConnection(
                        "jdbc:mysql://localhost/testdb", "test", "test");
      PreparedStatement ps = c.prepareStatement(
                        "insert into board( title, content) values(?,?)");
      ps.setString(1, board.getTitle());
      ps.setString(2, board.getContent());
      ps.executeUpdate();
      ps.close();
       c.close();
}
```

```
public Notice getBoard (String title) throws SQLException {
      Class.forName("com.mysql.jdbc.Driver");
      Connection c = DriverManager.getConnection( "jdbc:mysql://localhost/testdb", "test", "test");
      PreparedStatement ps = c.prepareStatement( "select * from board where title = ?");
      ps.setString(1, title);

      Notice notice = null;
      ResultSet rs = ps.executeQuery();
      if(rs.next()){
                  Notice notice = new Notice();
                  notice.setTitle(rs.getString("title"));
                  notice.setContent(rs.getString("content"));
       }
      rs.close();       ps.close();       c.close();
      return notice;
}
```

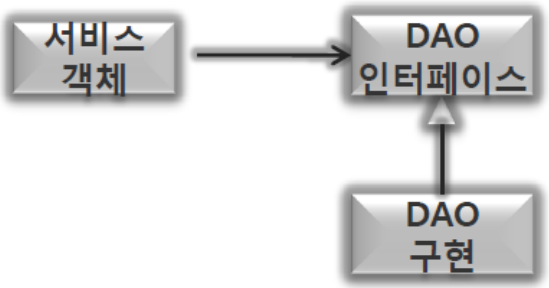
순수 JDBC 코드(2/2)

- JDBC 코드
  - 20%만 실제로 수행되는 코드! 나머지 80%는 반복되는 코드
  - 데이터 접근 도중 발생되는 에러의 실질적인 처리는 불가



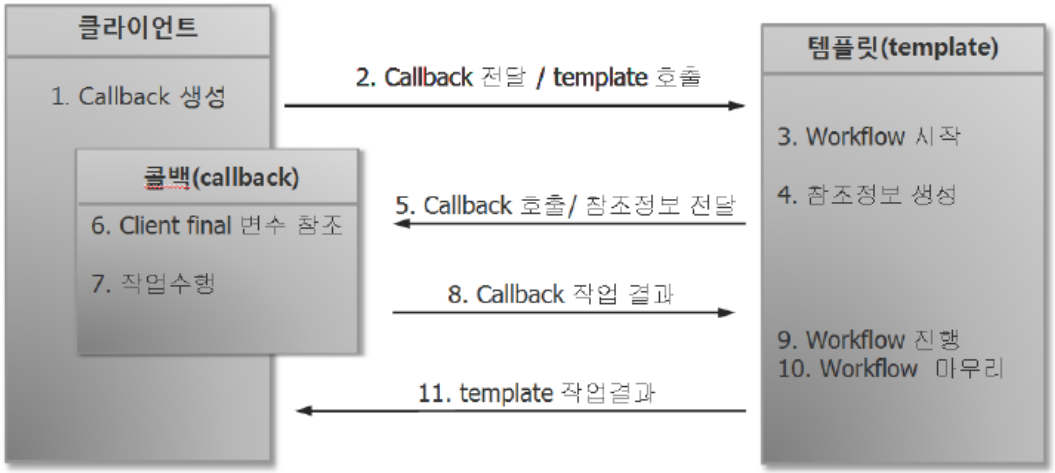
Spring의 데이터 접근 방식(1/10)

- DAO 인터페이스 적용
  - DAO 패턴으로 데이터 액세스 계층 분리
    - Data Access Object (DAO) 패턴
      - 데이터 액세스와 비즈니스 로직 분리
      - DAO 객체에게 DB 접근에 대한 책임성을 위임
      - 나머지 시스템은 Data Transfer Object(DTO)를 통해 상호작용
    - DAO의 책임
      - 외부 데이터로부터 데이터를 읽고 비즈니스 컴포넌트가 사용하기 위한 객체 캡슐화를 제공
      - 비즈니스 로직은 별도로 위치하고 독립적으로 테스트가 가능하며 저장 레이어 변경이 발생해도 재사용이 가능함
    - DTO 또는 도메인 오브젝트만을 사용하는 인터페이스를 통해 데이터 액세스 기술을 외부에 노출하지 않도록 하고 데이터 액세스 기능 처리
      - 장점
        - 서비스 객체를 손쉽게 테스트가 가능하도록 함
        - 저장 기술 (persistence technology)에 종속적이지 않게 함
        - DAO를 이용하는 서비스 계층의 코드를 기술이나 환경에 종속되지 않는 순수한 POJO로 개발 가능



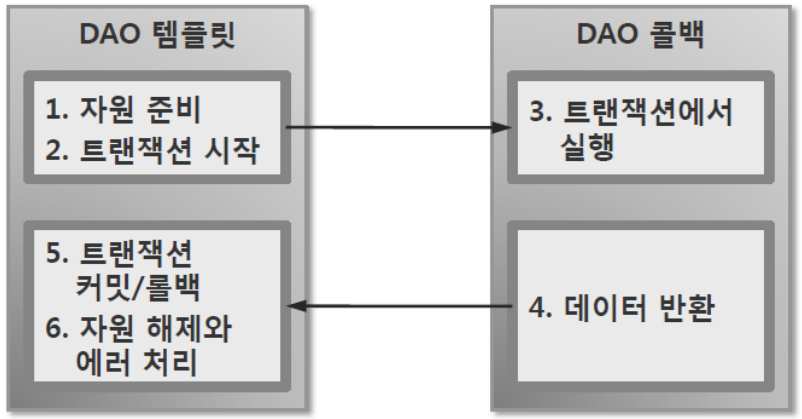
Spring의 데이터 접근 방식(2/10)

- 데이터 접근 템플릿화
  - 템플릿 메소드(template method) 패턴
    - 프로세스에 대한 틀(skeleton) 정의
    - 전반적인 프로세스는 고정되어 있고, 특정 시점의 프로세스는 특정 상황에 따라 세부적인 구현 내용이 달라짐
    - 템플릿/콜백
      - 템플릿(template) : 프로세스의 고정된 부분을 담당
      - 콜백(callback) : 템플릿 안에서 수행될 비즈니스 로직을 담당



Spring의 데이터 접근 방식(3/10)

- 데이터 접근 템플릿화
  - 템플릿 메소드 패턴을 적용
    - 데이터 접근 단계의 연결과 자원 해제하는 부분은 고정
      - 템플릿(template)
    - 데이터의 처리(CRUD)는 각각 다름
      - 콜백(callback)



* DAO 템플릿은 공통적인 데이터 접근 작업을 수행하며, 애플리케이션에 특화된 작업은 사용자가 정의한 DAO 콜백 객체로 호출함

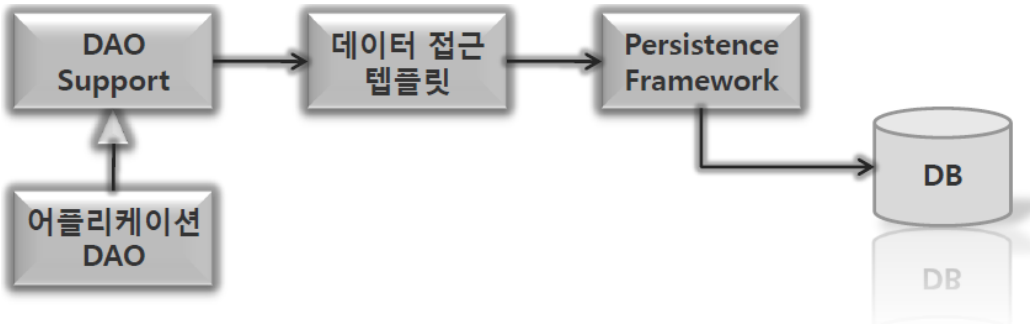
Spring의 데이터 접근 방식(4/10)

- 데이터 접근 템플릿화
  - Spring의 템플릿 클래스

템플릿 클래스 (org.springframework.*)	사용 목적
jca.cci.core.CciTemplate	JCA CCI 연결
jdbc.core.JdbcTemplate	JDBC 연결
jdbc.core.namedparam.NamedParameterJdbcTemplate	명명된(named) 파라미터에 대한 지원과 함께 사용하는 JDBC 연결
jdbc.core.simple.SimpleJdbcTemplate	Java 5 구조와 함께 단순화된 JDBC 연결
org.hibernate.HibernateTemplate	Hibernate 2.x 세션
org.hibernate3.HibernateTemplate	Hibernate 3.x 세션
org.ibatis.SqlMapClientTemplate	iBATIS SqlMap 클라이언트
orm.jdo.JdoTemplate	Java Data Object 구현
orm.jpa.JpaTemplate	Java Persistence API 엔티티 관리자
orm.toplink.TopLinkTemplate	오라클의 TopLink

Spring의 데이터 접근 방식(5/10)

- DAO 지원 클래스 사용
  - Spring의 DAO 지원 클래스



  - 애플리케이션에 대한 DAO 구현시 Spring의 DAO 지원 클래스를 상속 받음
    - 내부의 데이터 접근 템플릿에 바로 접근하려면 템플릿 조회 메소드를 호출
    - JdbcDaoSupport 클래스를 상속받고, getJdbcTemplate()을 호출해서 작업 수행
  - 저장 플랫폼을 접근하려면 DB와 통신하기 위해 사용되는 클래스를 접근함
    - JdbcDaoSupport의 getConnection() 메소드

Spring의 데이터 접근 방식(6/10)

- DAO 지원 클래스 사용
  - Spring의 DAO 지원 클래스

DAO 지원 클래스 (org.springframework.*)	사용 목적
jca.cci.support.CciDaoSupport	JCA CCI 연결
jdbc.core.support.JdbcDaoSupport	JDBC 연결
jdbc.core.namedparam.NamedParameterJdbcDaoSupport	명명된(named) 파라미터에 대한 지원과 함께 사용하는 JDBC 연결
jdbc.core.simple.SimpleJdbcDaoSupport	Java 5 구조와 함께 단순화된 JDBC 연결
org.hibernate.support.HibernateDaoSupport	Hibernate 2.x 세션
org.hibernate3.support.HibernateDaoSupport	Hibernate 3.x 세션
org.ibatis.support.SqlMapClientDaoSupport	iBATIS SqlMap 클라이언트
orm.jdo.support.JdoDaoSupport	Java Data Object 구현
orm.jpa.support.JpaDaoSupport	Java Persistence API 엔티티 관리자
orm.toplink.support.TopLinkDaoSupport	오라클의 TopLink

Spring의 데이터 접근 방식(7/10)

- Spring의 데이터 접근 예외처리
  - 예외의 종류
    - Error
      - 시스템의 비정상적인 상황에서 발생
      - 주로 자바 VM에서 발생시키는 것으로 애플리케이션 코드에서 처리 불가
      - Ex> OutOfMemoryError, ThreadDeath
    - Exception과 Checked 예외
      - 통상적 예외
      - 애플리케이션 코드의 작업 중 예외 상황이 발생했을 경우 사용
      - 반드시 예외를 처리하는 코드를 작성해야 함
        - 미 작성시 컴파일 오류 발생
    - RuntimeException과 Unchecked 예외
      - 명시적인 예외처리를 강제하지 않음
      - 주로 프로그램의 오류가 있을 때 발생하도록 의도함
        - 개발자의 부주의로 발생
      - Ex> NullPointerException , IllegalArgumentException

Spring의 데이터 접근 방식(8/10)

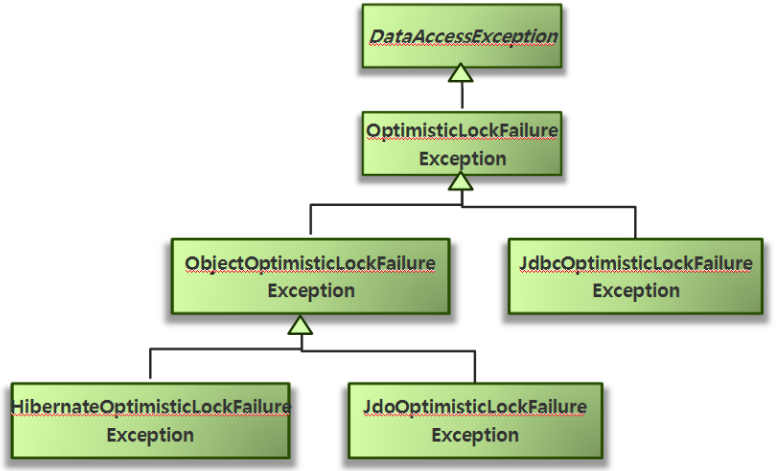
- Spring의 데이터 접근 예외처리
  - java.sql.SQLException
    - Checked 예외
    - 발생 유형
      - 애플리케이션이 DB 접근이 불가능한 경우
      - 수행되는 질의(query)가 문법에 오류가 발생되는 경우
    - 질의(query)문에 있는 테이블이나 컬럼이 존재하지 않는 경우
      - DB 제약조건으로 인해 값에 대한 저장이나 수정이 안되는 경우
    - SQLException은 복구 가능한 예외인가? NO~!
  - Spring의 데이터 접근 예외처리
    - JDBC 작업 중 발생하는 모든 예외(Checked 예외)는 Spring의 JDBC 예외 변환기가 Unchecked 예외로 처리
    - Spring의 데이터 접근 예외를 사용하면 Spring에서 제공하는 데이터 접근 템플릿을 사용해야 함.

JDBC 코드

```
public void registerNotice(Notice notice)
                    throws SQLException {
    ******
}
```

Spring의 데이터 접근 방식(9/10)

- Spring의 데이터 접근 예외처리
  - 예외 발생시 처리 문제
    - JDBC는 모든 데이터 접근 문제를 SQLException 예외로 처리함
      - 세부적인 처리를 위해서는 SQLException을 세부적으로 분석해야 함
    - Hibernate와 같은 저장(persistence) 프레임워크는 예외를 여러 계층으로 구조화 함
      - Hibernate에 특수한 예외를 발생시키면 애플리케이션은 Hibernate에 종속이 됨
    - 세부적으로 분석된 데이터 접근 예외에 대한 계층 구조는 필요하지만, 특정 저장 프레임워크와 직접적으로 연관이 되어서는 안됨
  - 저장 플랫폼에 독립적인 예외
    - DataAccessException
      - Unchecked 예외
        - 명시적으로 catch 블록을 사용할 필요가 없음
      - 자바 데이터 액세스시 발생하는 예외를 전환함
      - 데이터 액세스 기술의 종류와 상관없이 동일한 의미의 예외 발생시, 일관된 예외 발생



Spring의 데이터 접근 방식(10/10)

- Spring의 데이터 접근 예외처리
  - 저장 플랫폼에 독립적인 예외

JDBC 예외	Spring 데이터 접근 예외
BatchUpdateException DataTruncation SQLExcxeption SQLWarning	CannotAquireLockException CannotSerializeTransactionException CleanupFailureDataAccessException ConcurrencyFailureException **DataAccessException** DataAccessResourceFailureException DataIntegrityViolationException DataRetrievalFailureException DeadLockLoserDataAccessException EmptyResultDataAccessException IncorrectResultSizeDataAccessException IncorrectUpdateSemanticsDataAccessException InvalidDataAccessApiUsageException InvalidDataAccessResourceUsageException OptimiticLockingFailureException PermissionDeniedDataAccessException PessimisticLockingFailureException TypeMismatchDataAccessException UncategorizedDataAccessException

Spring의 데이터 소스 설정(1/4)

*JNDI(Java Naming and Directory Interface)는 디렉터리 서비스에서 제공하는 데이터 및 객체를 발견(discover)하고 참고(lookup) 하기 위한 자바 API

- JNDI 데이터 소스 사용
  - Spring 애플리케이션이 JEE 어플리케이션 서버 내에 배포될 때 사용
  - 장점
    - 애플리케이션 외부에 전적으로 관리를 맡김
    - 애플리케이션 서버는 우수한 성능으로 데이터 소스를 관리하고 시스템 관리자는 데이터 소스를 제어할 수 있음
  - JndiObjectFactoryBean

```
<bean id="dataSource"
        class="org.springframework.jndi.JndiObjectFactoryBean" scope="singleton">
        <property name="jndiName" value="/jdbc/RantzDatasource"/>
        <property name="resourceRef" value="true"/>
</bean>
```

    - jndiName 속성은 JNDI에서 자원의 이름을 지정
    - 애플리케이션이 자바 애플리케이션 서버 내에서 구동하면 resourceRef 값을 true 로 세팅

Spring의 데이터 소스 설정(2/4)

***JNDI**는 데이터베이스의 DB Pool을 미리 Naming시켜주는 방법으로 WAS의 데이터베이스 정보에 JNDI를 설정해 놓으면 웹 애플리케이션은 JNDI만 호출하면 간단
- 자원 낭비를 줄일 수 있음

- JNDI 데이터 소스 사용
  - Spring 에서의 JNDI 데이터 소스

```
<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"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.0.xsd">
    <jee:jndi-lookup id="dataSource"
                     jndi-name="/jdbc/RantzDatasource"
                     resource-ref="true"/>
</beans>
```

jee네임스페이스 사용

Spring의 데이터 소스 설정(3/4)

- 풀링된 데이터 소스 사용
  - Jakarta Commons Database Connection Pools (DBCP)
    - commons.apache.org/dbcp
    - BasicDataSource

```
<bean id="dataSource"  class="org.apache.commons.dbcp.BasicDataSource">

        <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
        <property name="url" value="jdbc:hsqldb:hsql://localhost/roadrantz/roadrantz"/>
        <property name="username" value="sa"/>
        <property name="password" value=""/>
        <property name="initialSize" value="5"/>
        <property name="maxActive" value="10"/>
</bean>
```

Spring의 데이터 소스 설정(4/4)

- JDBC 드라이버 기반한 데이터 소스
  - DriverManagerDataSource
    - 요청된 연결에 대해서 매번 새로운 연결을 제공
    - DBCP의 BasicDataSource와 달리 DriverManagerDataSource에 의해 제공되는 연결은 풀링되지 않음
  - SingConnectionDataSource
    - 요청된 연결에 대해서 매번 동일한 연결을 제공
    - 정확하게 풀링된 데이터 소스는 아니지만 단 하나의 연결을 풀링하는 데이터 소스라고 보면 됨

```
<bean id="dataSource"    class="org.springframework.jdbc.datasource.DriverManagerDataSource">

            <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
            <property name="url" value="jdbc:hsqldb:hsql://localhost/roadrantz/roadrantz"/>
            <property name="username" value="sa"/>
            <property name="password" value=""/>
</bean>
```

  - 멀티쓰레드 환경에서 사용이 제약됨

Spring의 JDBC 사용(1/11)

- JDBC 코드
  - 저장 – 소스 코드

```
public class SpringBoardDaoImpl {
    private DataSource dataSource;
    public void setDataSource(DataSource dataSource) {
            this.dataSource = dataSource;
    }       //Spring DI setter 방식 이용함

private static final String BOARD_INSERT =
        "INSERT INTO Board(id, title, content) VALUES(?,?,?)";
```

```
public void registerBoard(Board board) {
    Connection conn = null;
    PreparedStatement ps = null;
    try {
            conn = dataSource.getConnection();          //DB 연결
            ps = conn.prepareStatement(BOARD_INSERT);
            ps.setInt(1, board.getId());
            ps.setString(2, board.getTitle());
            ps.setString(3, board.getContent());
            ps.executeUpdate();

    } catch (SQLException e) {
            e.printStackTrace();
    } finally {
            try {
            if (ps != null)
            ps.close();                 //자원 해제
            if (conn != null)
            conn.close();
    } catch (SQLException e) {
    }
}}}
```

Spring의 JDBC 사용(2/11)

- JDBC 코드
  - 저장 – Spring 설정 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="dataSource“ class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
        <property name="url" value="jdbc:hsqldb:hsql://localhost/springTest" />
        <property name="username" value="SA" />
        <property name="password" value="" />
    </bean>
    <bean id="SpringBoardDaoImpl" class="kr.nextree.kosta.spring.ch03.jdbc.dao.impl.SpringBoardDaoImpl">
        <property name="dataSource" ref="dataSource" />
     </bean>
</beans>
```

Spring의 JDBC 사용(3/11)

- JDBC 코드
  - 저장 – 테스트 코드

```
@Before
public void setUp() {
        context = new ClassPathXmlApplicationContext("applicationContext.xml");
}
@Test
public void testRegisterBoard() {
        SpringBoardDao dao = (SpringBoardDao) context.getBean("SpringBoardDaoImpl");
        List<Board> boards = dao.findBoards();
        Board board = new Board();
        board.setId(boards.get(boards.size()-1).getId() + 1);
        board.setTitle("스프링 프레임워크 ");
        board.setContent("스프링 프레임워크");
        dao.registerBoard(board);
        List<Board> registedboards = dao.findBoards();
        assertTrue("게시물 등록 실패", registedboards.size() == (boards.size() + 1));
}
```

Spring의 JDBC 사용(4/11)

- JDBC 템플릿 사용
  - 세가지 템플릿 클래스
    - JdbcTemplate
      - Spring의 JDBC 템플릿 중에 가장 기본 클래스로 JDBC를 통한 데이터베이스에 대한 단순한 연결과 단순한 인덱싱 파라미터 쿼리를 제공
    - NamedParameterJdbcTemplate
      - 인덱싱된 파라미터가 아닌 SQL에 명명된(named) 파라미터로 값을 세팅하는 쿼리를 수행하게 함
    - SimpleJdbcTemplate
      - Autoboxing 이나, generics, 가변(variable) 파라미터 리스트와 같은 Java 기능을 사용하여 JDBC 템플릿 사용방법을 단순화시켜서 제공

Spring의 JDBC 사용(5/11)

- JDBC 템플릿 사용
  - JdbcTemplate를 사용한 데이터 접근

//SpringBoardDao.java

```
public void registerBoard(Board board);
```

```
public class JdbcTemplateBoardDaoImpl implements SpringBoardDao {
    private JdbcTemplate jdbcTemplate;
    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {          //Spring 설정 파일에서 주입
            this.jdbcTemplate = jdbcTemplate;
    }
    …
    public void registerBoard(Board board) {
            jdbcTemplate.update(BOARD_INSERT, new Object[] { board.getId(),     //JdbcTemplate을 사용한 실행 구문
            board.getTitle(), board.getContent() });
    }
}
```

Spring의 JDBC 사용(6/11)

- JDBC 템플릿 사용
  - JdbcTemplate를 사용한 데이터 접근

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
</bean>
<bean id="JdbcTemplateBoardDaoImpl"   class="kr.nextree.kosta.spring.ch03.jdbc.dao.impl.JdbcTemplateBoardDaoImpl">
        <property name="jdbcTemplate" ref="jdbcTemplate" />
</bean>
```

    - DB 연결, statement 생성, SQL 실행 등은 JdbcTemplate 이 대신 수행함
    - SQLException은 JdbcTemplate에서 catch 하여 throw 함
      - 일반적인 SQLException을 세부적인 데이터 접근 에러로 변경하여 throw 함
      - Spring의 데이터 접근 예외들은 모두 runtime 계열 예외이므로, saveMotorist() 메소드에 예외를 처리할 필요가 없음

Spring의 JDBC 사용(7/11)

- JDBC 템플릿 사용
  - 명명된(named) 파라미터 사용
    - NamedParameterJdbcTemplate 사용

```
private NamedParameterJdbcTemplate jdbcTemplate;
public void setJdbcTemplate(NamedParameterJdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
}
private static final String BOARD_INSERT =
        "INSERT INTO Board(id, title, content)
        VALUES (:id, :title, :content)";                    //SQL에 표시된 명명된(named) 파라미터

public void registerBoard(Board board) {
        Map<String, Object> params = new HashMap<String, Object>();    //java.util.Map 으로 파라미터의 값을 넘김
        params.put("id", board.getId());
        params.put("title", board.getTitle());
        params.put("content", board.getContent());

        jdbcTemplate.update(BOARD_INSERT, params);
}
```

Spring의 JDBC 사용(8/11)

- JDBC 템플릿 사용
  - 명명된(named) 파라미터 사용

```
<bean id="jdbcTemplate"   class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
            <constructor-arg ref="dataSource"/>
</bean>
<bean id="NamedParameterJdbcTemplateBoardDaoImpl"
            class="kr.nextree.kosta.spring.ch03.jdbc.dao.impl.NamedParameterJdbcTemplateBoardDaoImpl">
            <property name="jdbcTemplate" ref="jdbcTemplate"/>
</bean>
```

Spring의 JDBC 사용(9/11)

- JDBC 템플릿 사용
  - Java에서 JDBC 단순화

```
public void registerBoard(Board board) {
    jdbcTemplate.update(BOARD_INSERT, new Object[] { board.getId(),
                                        board.getTitle(), board.getContent() });
}
```

//변수의 길이가 결정되지 않은 경우,
//Object 배열을 감싸서 파라미터를 넘김

```
public void registerBoard(Board board) {
    jdbcTemplate.update(BOARD_INSERT, board.getId(),
                                        board.getTitle(), board.getContent());
}
```

//Java의 가변 파라미터(varargs) 리스트를 사용해서
//Object 배열을 감싸지 않고, 파라미터를 직접 넘길 수 있음

```
<bean id="jdbcTemplate"
            class="org.springframework.jdbc.core.simple.SimpleJdbcTemplate">   //SimpleJdbcTemplate 사용
            <constructor-arg ref="dataSource"/>                                //생성자 주입
</bean>
```

Spring의 JDBC 사용(10/11)

- JDBC를 위한 **Spring**의 **DAO** 지원 클래스 사용
  - JdbcDaoSupport 클래스
    - JdbcTemplate을 포함

//JdbcDaoSupport 가 dataSource를 가지고 있으므로,
jdbcTemplate을 거치지 않고 바로 매핑이 가능

```
public class JdbcDaoSupportBoardDaoImpl extends JdbcDaoSupport  implements SpringBoardDao {
   public void registerBoard(Board board) {
            getJdbcTemplate().update( BOARD_INSERT, new Object[] { board.getId()
                                                                , board.getTitle(), board.getContent() });
}
```

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
            <property name="dataSource" ref="dataSource"/>
</bean>
<bean id="JdbcTemplateBoardDaoImpl"  class="kr.jdbc.dao.impl.JdbcTemplateBoardDaoImpl">
            <property name="jdbcTemplate" ref="jdbcTemplate" />
</bean>
```

```
<bean id="JdbcDaoSupportBoardDaoImpl"   class="kr.jdbc.dao.impl.JdbcDaoSupportBoardDaoImpl">
            <property name="dataSource" ref="dataSource" />
</bean>
```

Spring의 JDBC 사용(11/11)

- JDBC를 위한 **Spring**의 **DAO** 지원 클래스 사용
  - 명명된(named) 파라미터를 위한 DAO 지원
    - NamedParameterJdbcTemplate 사용
      - getNamedParameterJdbTemplate() 메소드
  - 단순화시킨 Java DAO
    - SimpleJdbcTemplate 사용
      - getSimpleJdbcTemplate() 메소드