

# Spring 과 DB

# Spring의 데이터 접근 방식

- Spring에서의 JDBC 프로그래밍
  - 데이터 접근 템플릿 제공 – JDBC 프로그래밍을 쉽게 할 수 있는 템플릿
  - DAO 지원 클래스 제공 – Template을 쉽게 사용할 수 있도록 지원하는 class
- 데이터 접근 템플릿
  - 템플릿 메소드(template method) 패턴
    - 템플릿 메소드는 프로세스에 대한 틀(skeleton)을 정의함.
    - 전반적인 프로세스(의 흐름)는 고정되어 있고, 변하지 않음.
    - 특정 시점의 프로세스는 특정 상황에 따라 세부적인 구현 내용이 달라짐.
  - Spring의 데이터 접근은 템플릿 메소드 패턴을 적용
    - 데이터 접근 단계에서 연결을 얻고, 자원을 해제하는 부분은 고정
    - 데이터 접근 방법은 각각 다름.
  - 두가지 클래스
    - 템플릿(template)
      - 프로세스의 고정된 부분을 담당

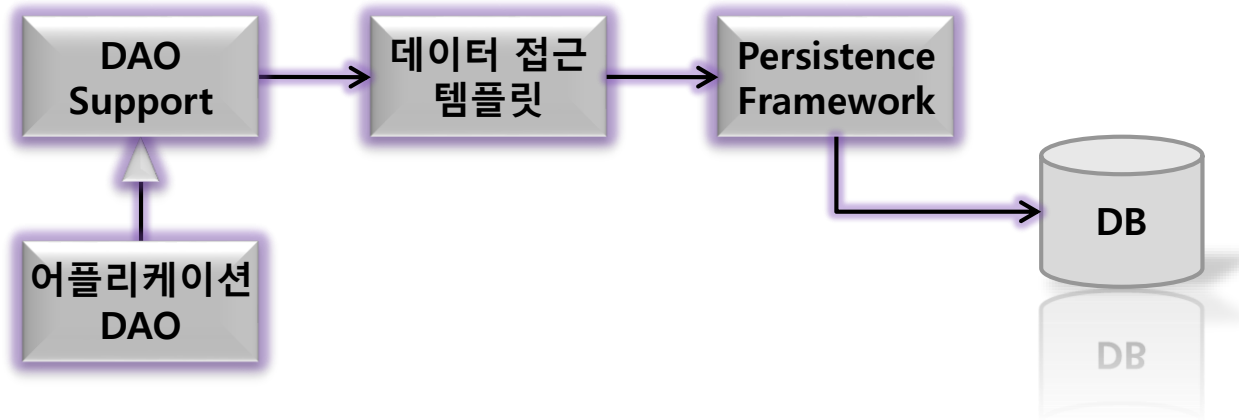
# Spring의 데이터 접근 방식

- 데이터 접근 템플릿화
  - Spring의 템플릿 클래스

템플릿 클래스 (org.springframework.*)	사용 목적
<code>jdbc.core.JdbcTemplate</code>	JDBC 연결
<code>jdbc.core.namedparam.NamedParameterJdbcTemplate</code>	명명된(named) 파라미터에 대한 지원과 함께 사용하는 JDBC 연결
<code>jdbc.core.simple.SimpleJdbcTemplate</code>	Java 5 구조와 함께 단순화된 JDBC 연결
<code>org.ibatis.SqlMapClientTemplate</code>	iBATIS SqlMap 클라이언트
<code>orm.jdo.JdoTemplate</code>	Java Data Object 구현
<code>orm.jpa.JpaTemplate</code>	Java Persistence API 엔티티 관리자

# Spring의 데이터 접근 방식

- DAO 지원 클래스 사용
  - Spring의 DAO 지원 클래스



- 어플리케이션에 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

- DAO 지원 클래스 사용
  - Spring의 DAO 지원 클래스

DAO 지원 클래스 (org.springframework.*)	사용 목적
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.ibatis.support.SqlMapClientDaoSupport	iBATIS SqlMap 클라이언트
orm.jdo.support.JdoDaoSupport	Java Data Object 구현
orm.jpa.support.JpaDaoSupport	Java Persistence API 엔티티 관리자

# 데이터 소스 설정-JNDI 데이터 소스 사용

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

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

- jndiName 속성은 JNDI에서 자원의 이름을 지정
- 어플리케이션이 자바 어플리케이션 서버 내에서 구동하면 resourceRef 값을 true로 세팅
  - 원래의 jndiName 값에 'java:comp/env' 가 붙음.
  - 'java:/comp/env/jdbc/oracle'

# 데이터 소스 설정-JNDI 데이터 소스 사용

- JNDI 데이터 소스 사용 - 2
  - Spring 2.0 이상에서의 JNDI 데이터 소스
  - Spring 설정파일에 jee 네임스페이스 등록하여 처리

```
<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-2.0.xsd" >
  <jee:jndi-lookup id="dataSource"
    jndi-name="/jdbc/oracle"
    resource-ref="true"/>
</beans>
```

# 데이터 소스 설정-DBCP 사용

- DBCP 사용
  - Spring이 기본으로 제공하는 DBCP 사용 할 수 있다.
  - Jakarta Commons Database Connection Pools (DBCP)
    - <http://commons.apache.org/dbcp/>
    - BasicDataSource 사용
  - DataSource를 bean으로 등록한다.

```
<bean id="dataSource"
  class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
  <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521/XE"/>
  <property name="username" value="hr"/>
  <property name="password" value="hr"/>
  <property name="initialSize" value="5"/>
  <property name="maxActive" value="10"/>
</bean>
```

- 설정항목
  - <http://commons.apache.org/dbcp/configuration.html> 참고



# 데이터 소스 설정-JDBC DataSource 사용

- JDBC 드라이버 기반한 데이터 소스
  - DriverManagerDataSource
    - 요청된 연결에 대해서 매번 새로운 연결을 제공.
    - DBCP의 BasicDataSource와 달리 DriverManagerDataSource에 의해 제공되는 연결은 풀링되지 않음.
  - SingleConnectionDataSource
    - 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:oracle:thin:@127.0.0.1:1521/XE "/>
  <property name="username" value="sa"/>
  <property name="password" value=""/>
</bean>
```

- 멀티쓰레드 환경에서 사용이 제약됨.

# Spring에서 JDBC 사용 예

- 기본 JDBC 코드 – DataSource를 주입 받아 JDBC 코딩

```
public class MemberDAO{
    private DataSource dataSource;
    public void insertMember(MemberVO mvo) {
        Connection conn = null;
        PreparedStatement pstmt = null;
        String sql = "insert ...";
        try {
            conn = dataSource.getConnection();
            pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, mvo.getId());
            pstmt.setString(2, mvo.getPassword());
            pstmt.setString(3, mvo.getName());
            pstmt.setString(4, mvo.getAddress());
            pstmt.executeUpdate();
        }
```

```
        } catch (SQLException e) {
            //오류처리
        } finally {
            try {
                if (stmt != null) pstmt.close();
                if (conn != null) conn.close();
            } catch (SQLException e) {}
        }
    }

    //주입받는다.
    public void setDataSource(DataSource ds) {
        this.dataSource = ds;
    }
}
```

# Spring에서 JDBC 사용

- JDBC 코드
  - 저장 – Spring 설정 파일

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>  
  <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:XE"/>  
  <property name="username" value="hr"/>  
  <property name="password" value="hr"/>  
</bean>  
  
<bean id="memberDAO" class="dao.MemberDAO">  
  <property name="dataSource" ref="dataSource"/>  
</bean>
```

# JDBC Template을 이용

- 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 5 기능을 사용하여 JDBC 템플릿 사용방법을 단순화 시켜서 제공

# JDBC Template을 이용

- JDBC 템플릿 사용 – update 계열
  - JdbcTemplate를 사용한 데이터 접근

```
public interface DAO {  
    void insert(MemberVO mvo);  
}
```

```
public class MemberDAO implements DAO {  
    private JdbcTemplate jdbcTemplate;  
    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {  
        this.jdbcTemplate = jdbcTemplate;  
    }  
    ...  
    @Override  
    public void insert(MemberVO mvo) {  
        jdbcTemplate.update(MEMBER_INSERT,  
            new Object[]{mvo.getId(), mvo.getPassword(),  
                mvo.getName(), mvo.getAddress()});  
    }  
}
```

*Spring 설정  
파일에서 주입*

*JdbcTemplate을  
사용한 실행 구문*

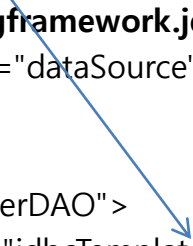
*쿼리의 인덱싱된 파라미터 값*

*실행할 SQL 구문*

# Spring에서 JDBC 사용

- JDBC 템플릿 사용
  - JdbcTemplate를 사용한 데이터 접근

```
<bean id="jdbcTemplate"
      class="org.springframework.jdbc.core.JdbcTemplate">
  <property name="dataSource" ref="dataSource"/>
</bean>
<bean id="dao"
      class="dao.MemberDAO">
  <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 예외로 변경하여 throw 함.
  - Spring의 템플릿 메소드들이 제공하는 예외는 Runtime계열의 메소드이므로 따로 예외를 잡을 필요 없다.

# Spring에서 JDBC 사용

- JDBC 템플릿 사용 - select

- JdbcTemplate를 사용한 데이터 접근

실행할 SQL 구문

```
public Motorist selectMemberById(String id) {  
    List matches = jdbcTemplate.query(MEMBER_SELECT_BY_ID,  
        new Object[] {id},  
        new RowMapper() {  
            @Override  
            public Object mapRow(ResultSet rs, int rowNum) throws SQLException {  
                return new Motorist(rs.getInt(1), rs.getString(2), rs.getString(3),  
                    rs.getString(4), rs.getString(5));  
            }  
        });  
    return matches.size() > 0 ? (Motorist)matches.get(0) : null;  
}
```

쿼리의 인덱싱된 파라미터 값

ResultSet에서 값을 추출하여 도메인 객체를 만들어 냄.

쿼리 실행으로부터 결과인 모든 행(row)에 대해서 JdbcTemplate은 RowMapper의 mapRow() 메소드를 호출

RowMapper 내에서 VO(DTO)객체를 생성하고 ResultSet에서 값을 세팅함.