Ch 8 DB 연동

DB 연동, JDBC와 JdbcTemplate

- JDBC는 자바 프로그램 안에서 SQL을 실행하기 위해 데이터베이스를 연결해주는 API를 말한다.
- JDBC를 사용하면
  - 1. Connection을 구한 다음
  - 2. 쿼리 실행을 위한 PreparedStatement를 생성하고
  - 3. 쿼리 실행 이후 finally 블록에서 ResultSet, PreparedStatement, Connection을 닫는다.

- 여기서 JDBC의 문제점은 중복되는 코드가 다수 발생한다는 것.

JdbcTemplate

- JDBC의 구조적인 반복을 줄이기 위해 등장한 것이 스프링의 JdbcTemplate이다.
- JdbcTemplate을 사용하면 코드 길이가 단축되고 더 나아가 트랜잭션 관리가 쉬워진다. (추후 설명)

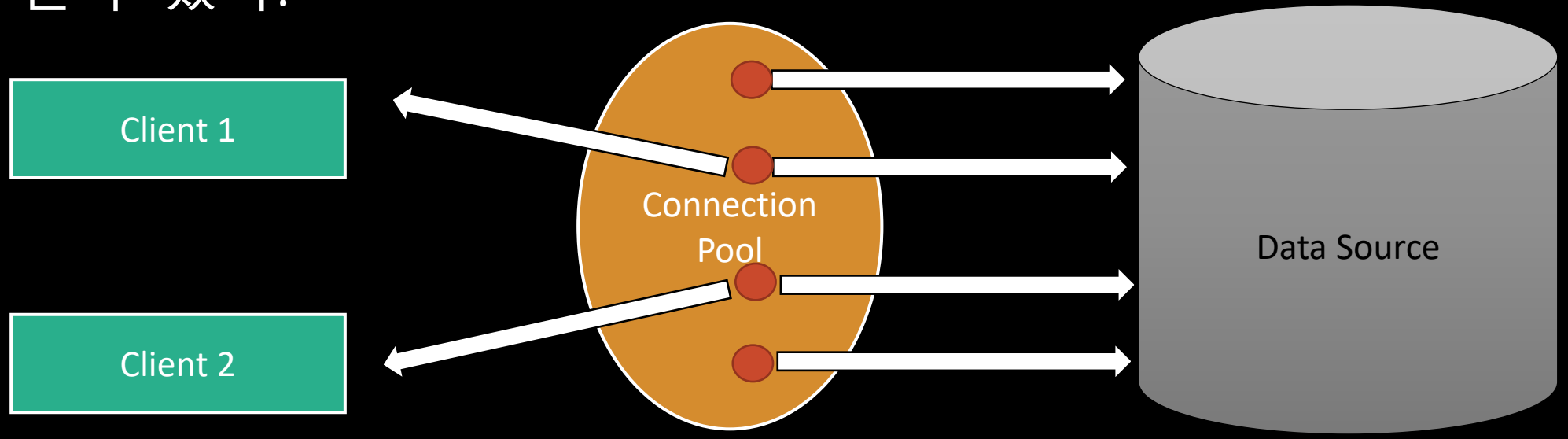
커넥션 풀(Connection pool)

- 일정 개수의 DB Connection을 미리 만들어 두는 기법이 커넥션 풀이다.

- 실제 서비스 운영 환경에서는 서로 다른 장비를 이용해서 자바 프로그램과 DBMS를 실행한다.
- 이때 커넥션을 생성하는 시간은 컴퓨터의 입장에서 매우 길기 때문에 전체 성능에 영향을 줄 수 있다.
- 커넥션 풀은 이러한 응답 속도 저하와 동시 접속자로 인한 부하를 줄이기 위해 사용한다.

커넥션 풀(Connection pool)

- 커넥션이 필요한 프로그램은 커넥션 풀에서 커넥션을 가져와 사용한 뒤 커넥션을 다시 풀에 반납한다.
- 커넥션을 미리 생성해 두기 때문에 커넥션을 생성하는 시간을 아낄 수 있다.



커넥션 풀(Connection pool)

- 만약 풀의 설정 값을 적절하지 못하게 설정한다면 커넥션 풀이 애플리케이션에서 병목 지점이 되는 경우도 있다.
- 웹 애플리케이션의 요청은 대부분 DBMS로 연결되기 때문에 커넥션 풀의 설정은 전체 애플리케이션의 성능과 안정성에 영향을 미치는 핵심이다.

Tomcat JDBC

- 이 책에서는 커넥션 풀을 구현하기 위해 Tomcat JDBC을 사용한다.
- Tomcat JDBC는 javax.sql.DataSource를 구현한 DataSource 클래스를 제공한다.

```
import org.apache.tomcat.jdbc.pool.DataSource;
```

- 이 DataSource 클래스를 설정 클래스에 스프링 빈으로 등록해서 커넥션 풀을 만든다.

DataSource 스프링 빈 등록

- DataSource를 스프링 빈으로 등록할 때 여러가지 설정을 할 수 있다.

```
@Bean(destroyMethod = "close")
public DataSource dataSource() {
    DataSource ds = new DataSource();
    ds.setDriverClassName("com.mysql.jdbc.Driver");
    ds.setUrl("jdbc:mysql://localhost/spring5fs?characterEncoding=utf8");
    ds.setUsername("spring");
    ds.setPassword("spring5");
    ds.setInitialSize(2);
    ds.setMaxActive(10);
    ds.setTestWhileIdle(true);
    ds.setMinEvictableIdleTimeMillis(60000 * 3);
    ds.setTimeBetweenEvictionRunsMillis(10 * 1000);
    return ds;
}
```

DataSource 클래스의 주요 프로퍼티

- 커넥션 풀을 초기화할 때 생성할 초기 커넥션 개수
- 커넥션 풀에서 동시에 사용할 수 있는 최대 커넥션 개수
- 커넥션 풀에 커넥션을 요청하고 대기하는 최대 시간
- 커넥션 풀에 커넥션을 유휴 상태로 유지할 최소 시간
- 커넥션 풀의 유휴 커넥션을 검사할 주기
- .
- .

커넥션의 상태

- 커넥션 풀에 커넥션을 요청하면 해당 커넥션은 활성 상태(active)가 되고, 커넥션을 다시 커넥션 풀에 반환하면 유휴 상태(idle)가 된다.

```
Connection conn = null;
try {
    conn = dataSource.getConnection();
```

```
finally {
  if (conn != null)
      try {
          conn.close();
```

- 요청은 DataSource의 getConnection() 메소드를, 반환은 Connection의 close() 메소드를 사용한다.

JdbcTemplate으로 DB 연동

- 커넥션 풀을 생성했으니 이제 DB 연동을 할 차례
- 스프링의 JdbcTemplate을 이용하여 쿼리를 실행할 수 있다.

1) JdbcTemplate 객체 생성

- MemberDao 클래스에 JdbcTemplate 객체를 생성한다.
  - DAO란 Data Access Object의 약자로, 데이터베이스 관련 작업을 전담하는 클래스이다.
  - 데이터베이스에 연결하여 데이터 입력/수정/삭제/조회 등의 작업을 하는 클래스를 말한다.

- 의존 주입(DI)
- 생성자, setter

```
public class MemberDao {

    private JdbcTemplate jdbcTemplate;

    public MemberDao(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }
```

2) 설정 클래스에 빈 등록

- 스프링 설정 클래스에 MemberDao 를 빈으로 등록한다.

```
@Bean
public MemberDao memberDao() {
    return new MemberDao(dataSource());
}
```

JdbcTemplate 쿼리 실행-query()

- SELECT 쿼리 실행을 위해 query() 메소드를 사용한다.
- query() 메소드는 전달받은 쿼리를 실행하고 그 결과를 자바 객체로 변환하여 리턴해준다.
  - 자바 객체로 변환하는 과정에서 RowMapper의 mapRow() 메소드가 실행된다.
  - 이때, RowMapper 객체를 익명 클래스나 람다로 전달할 수 있다.

```
List<Member> results = jdbcTemplate.query(
        "select * from MEMBER where EMAIL = ?",
        new RowMapper<Member>() {
            @Override
            public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
                Member member = new Member(
                        rs.getString("EMAIL"),
                        rs.getString("PASSWORD"),
                        rs.getString("NAME"),
                        rs.getTimestamp("REGDATE").toLocalDateTime());
                member.setId(rs.getLong("ID"));
                return member;
            }
        }, email);
```

JdbcTemplate 쿼리 실행-queryForObject()

- 만약 쿼리 결과가 한 행 뿐이라면 특정한 타입으로 받는 것이 더 편리하다.
- 이때 queryForObject() 메소드를 사용한다.
- 쿼리 실행 결과가 한 행이 아니라면 익셉션이 발생한다.

```
public int count() {
    Integer count = jdbcTemplate.queryForObject(
            "select count(*) from MEMBER", Integer.class);
    return count;
}
```

JdbcTemplate 쿼리 실행-update()

- INSERT, UPDATE, DELETE 쿼리는 update() 메소드를 사용한다.
- update() 메소드는 쿼리 실행 결과로 변경된 행의 수를 리턴한다.

MySQL의 AUTO_INCREMENT

- AUTO_INCREMENT 칼럼은 행이 추가되면 자동으로 값이 할당되는 칼럼으로 primary key ID에 사용된다.

```
create table spring5fs.MEMBER (
    ID int auto_increment primary key,
    EMAIL varchar(255),
    PASSWORD varchar(100),
```

- AUTO_INCREMENT와 같은 자동 증가 칼럼을 가진 테이블에 값을 삽입하면 해당 값이 자동으로 생성된다. 따라서 이 칼럼에 해당하는 값은 따로 지정할 필요가 없다.

JdbcTemplate의 KeyHolder

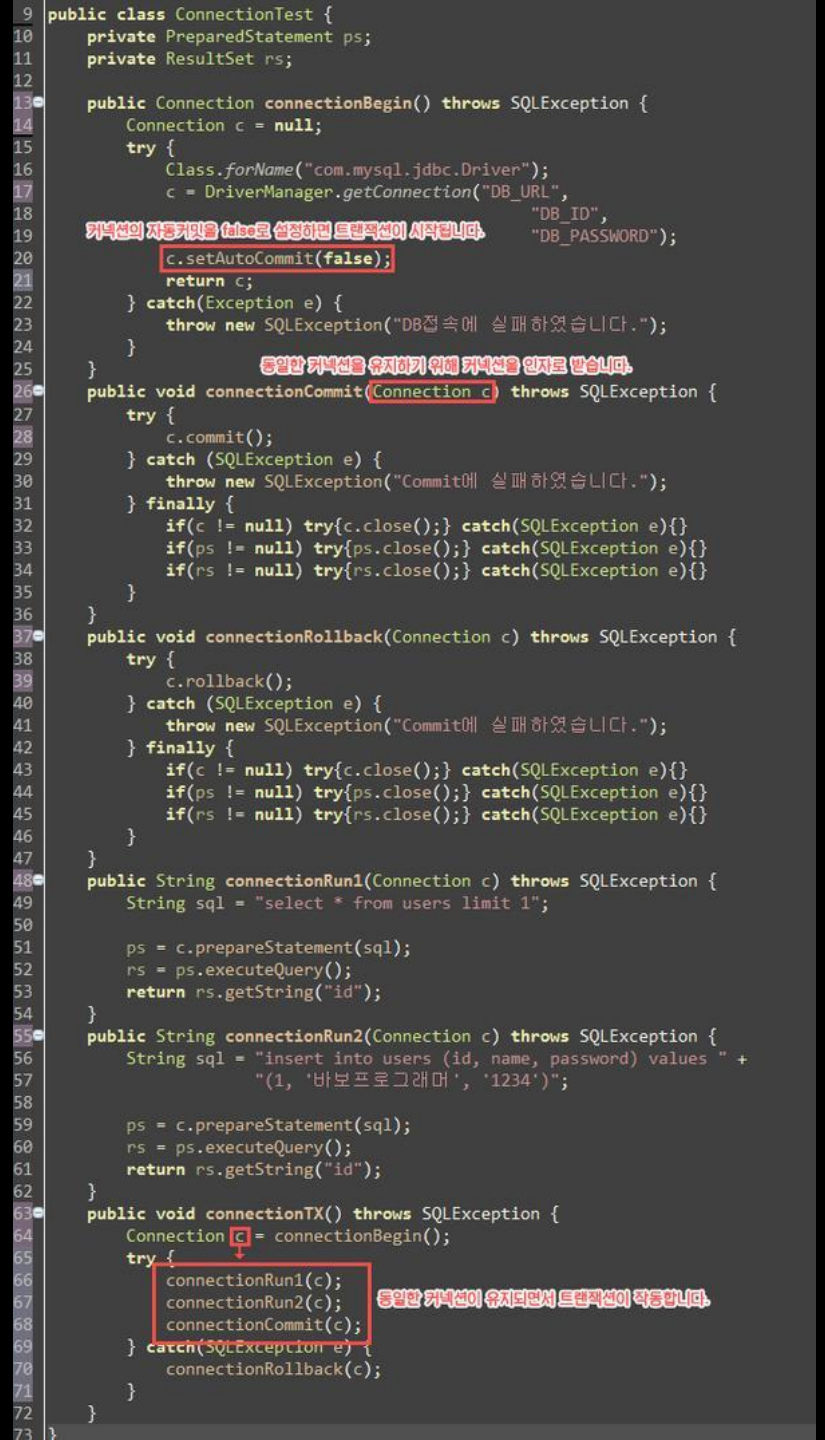
- AUTO_INCREMENT 칼럼처럼 자동으로 생성된 키 값을 알고 싶다면 KeyHolder를 사용하면 된다.
- 보관된 키 값은 getKey() 메소드를 이용하여 구한다.
- 이 메소드는 java.lang.Number를 리턴하므로 intValue(), longValue() 등의 메소드를 사용해서 원하는 타입의 값으로 변환할 수 있다.

트랜잭션 처리

- 두 개 이상의 쿼리를 하나의 작업으로 실행해야 할 때 사용하는 것이 트랜잭션(Transaction)이다.
- 한 트랜잭션으로 묶인 쿼리 중 하나라도 실패하면 전체 쿼리를 실패로 간주하고, 실패 이전에 실행한 쿼리를 취소하여 DB를 기존 상태로 되돌린다. 이를 롤백(Rollback)이라고 부른다.
- 반면에, 트랜잭션으로 묶인 모든 쿼리가 성공해서 그 결과를 DB에 반영하는 것을 커밋(Commit)이라고 한다.

트랜잭션 처리

- JDBC는 Connection의 setAutoCommit(false)를 이용해서 트랜잭션을 시작하고 commit()과 rollback()을 이용해서 커밋하거나 롤백한다.
- 이렇게 작성된 코드는 길이가 매우 길고, 프로그래머가 직접 트랜잭션 범위를 관리하기 때문에 커밋/롤백 코드를 누락하거나 중복할 가능성이 있다.



```java
public class ConnectionTest {
    private PreparedStatement ps;
    private ResultSet rs;

    public Connection connectionBegin() throws SQLException {
        Connection c = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            c = DriverManager.getConnection("DB_URL",
                                            "DB_ID",
                                            "DB_PASSWORD");
            c.setAutoCommit(false);
            return c;
        } catch(Exception e) {
            throw new SQLException("DB접속에 실패하였습니다.");
        }
    }
    public void connectionCommit(Connection c) throws SQLException {
        try {
            c.commit();
        } catch (SQLException e) {
            throw new SQLException("Commit에 실패하였습니다.");
        } finally {
            if(c != null) try{c.close();} catch(SQLException e){}
            if(ps != null) try{ps.close();} catch(SQLException e){}
            if(rs != null) try{rs.close();} catch(SQLException e){}
        }
    }
    public void connectionRollback(Connection c) throws SQLException {
        try {
            c.rollback();
        } catch (SQLException e) {
            throw new SQLException("Commit에 실패하였습니다.");
        } finally {
            if(c != null) try{c.close();} catch(SQLException e){}
            if(ps != null) try{ps.close();} catch(SQLException e){}
            if(rs != null) try{rs.close();} catch(SQLException e){}
        }
    }
    public String connectionRun1(Connection c) throws SQLException {
        String sql = "select * from users limit 1";

        ps = c.prepareStatement(sql);
        rs = ps.executeQuery();
        return rs.getString("id");
    }
    public String connectionRun2(Connection c) throws SQLException {
        String sql = "insert into users (id, name, password) values " +
                     "(1, '바보프로그래머', '1234')";

        ps = c.prepareStatement(sql);
        rs = ps.executeQuery();
        return rs.getString("id");
    }
    public void connectionTX() throws SQLException {
        Connection c = connectionBegin();
        try {
            connectionRun1(c);
            connectionRun2(c);
            connectionCommit(c);
        } catch(SQLException e) {
            connectionRollback(c);
        }
    }
}
```


- 스프링같이 트랜잭션을 위한 프레임워크가 없던 시절...
- 단 2개의 쿼리문을 실행하기 위해 거의 100줄에 가까운 코드를 작성해야 했다.

- 출처: http://springmvc.egloos.com/v/498979

스프링의 트랜잭션 처리

- 스프링이 제공하는 트랜잭션 처리를 사용하면 중복이 발생할 걱정 없이 매우 간단하게 범위를 지정할 수 있다.

- 스프링은 @TransactionaI 어노테이션을 제공하여 트랜잭션을 처리한다.

1) PlatformTransactionManager 빈 설정

- PlatformTransactionManager는 스프링이 제공하는 트랜잭션 매니저 인터페이스이다.
- 구현기술에 상관없이 동일한 방식으로 트랜잭션을 처리하기 위해 이 인터페이스를 사용한다.
- setDataSource() 메소드를 통해 트랜잭션에 사용할 DataSource를 지정한다.

```
@Bean
public PlatformTransactionManager transactionManager() {
    DataSourceTransactionManager tm = new DataSourceTransactionManager();
    tm.setDataSource(dataSource());
    return tm;
}
```

2)@EnableTransactionManagement

- @EnableTransactionManagement 어노테이션은 @TransactionaI 어노테이션이 붙은 메소드를 트랜잭션 범위에서 실행하는 기능을 활성화한다.

```
@Configuration
@EnableTransactionManagement
public class AppCtx {
```

3) @Transactional 활성화 설정

- 트랜잭션 범위에서 실행하고 싶은 빈 객체의 메소드에 @Transactional 어노테이션을 붙인다.

```
@Bean
public ChangePasswordService changePwdSvc() {
```

```
public class ChangePasswordService {

    private MemberDao memberDao;

    @Transactional
    public void changePassword(String email, String oldPwd, String newPwd) {
```

트랜잭션과 프록시(proxy)

- 트랜잭션은 공통 기능의 하나이다.
- 따라서 스프링은 @Transactional 어노테이션으로 트랜잭션을 처리하는 과정에서 내부적으로 AOP를 사용한다.
- 스프링에서 AOP는 프록시를 통해서 구현되므로 트랜잭션도 마찬가지로 프록시로 이루어진다.