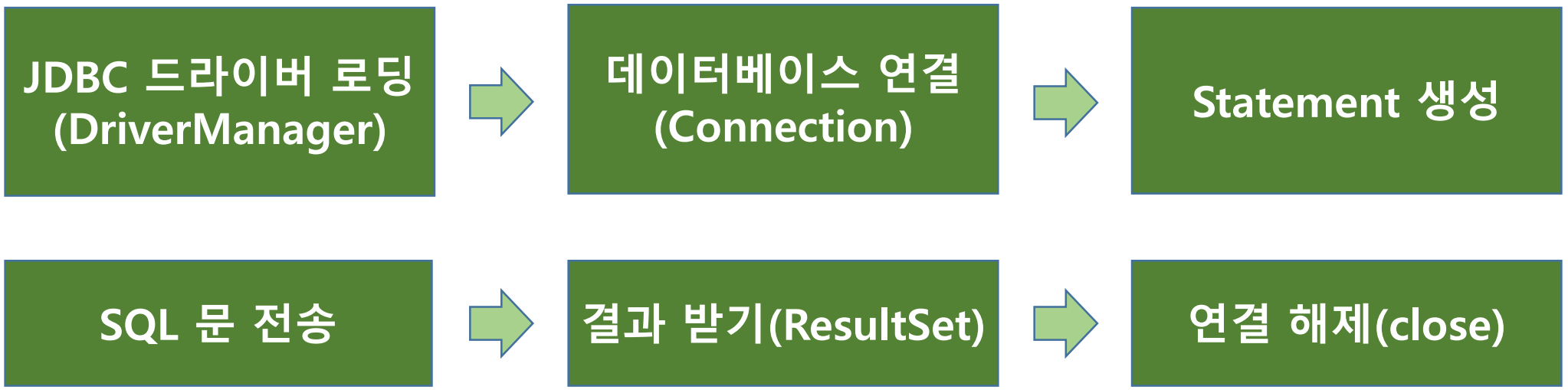
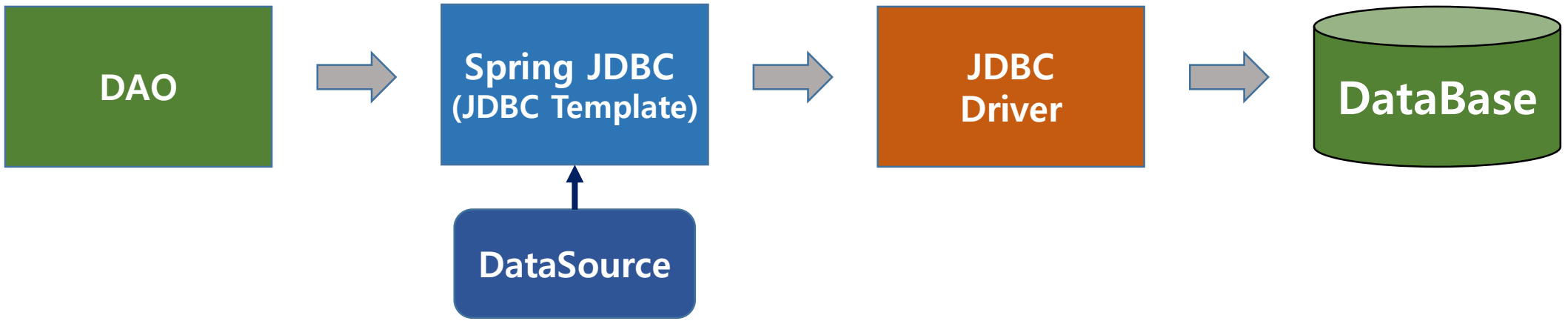
Java DataBase Connectivity



JDBC(Java DataBase Connectivity)는 데이터베이스에 연결 및 작업을 하기 위한 자바 표준 인터페이스이다. JDBC 를 사용하여 데이터베이스 작업을 하여도 되지만 반복적인 작업이 계속되는 단점이 있다.

Spring JDBC



Spring JDBC는 JDBC의 모든 반복적인 로직의 처리를 스프링 프레임워크에 위임하므로 Connection의 연결과 생성 및 종료 Statement 준비와 실행 및 종료 ResultSet 처리와, 예외 처리, 트랙젝션 등의 반복되는 작업을 Spring이 해줌으로 개발자가 직접 작성하지 않고 Database에 대한 작업을 수행할 수 있다.

Spring Connection Pool

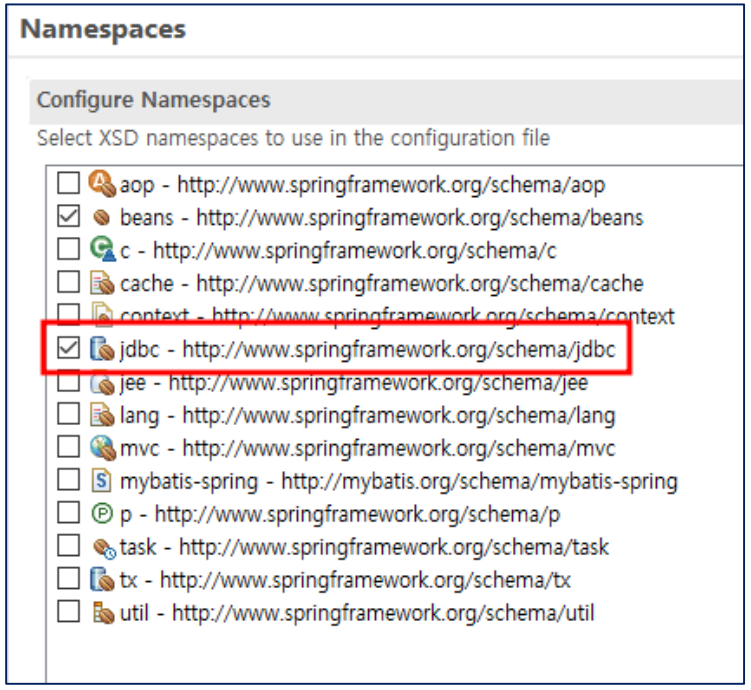
웹 컨테이너(WAS)가 실행되면서 DB와 미리 connection(연결)을 해놓은 객체들을 pool에 저장해 두었다가, 클라이언트 요청이 오면 connection을 빌려주고, 처리가 끝나면 다시 connection을 반납 받아 pool에 저장하는 방식을 말한다.
스프링의 JDBC Template과 커넥션풀을 관리하는 HikariCP 라이브러리를 사용하기 위해 <dependency> 태그로 pom.xml 에 추가한다.

```
<!-- spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.3.23</version>
</dependency>
```

```
<!-- zaxxer/HikariCP -->
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>5.0.1</version>
</dependency>
```

root-context.xml 빈 등록

root-context.xml에 다음과 같은 설정을 해준다.




```xml
<!-- 히카리 커넥션풀 빈 등록 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="username" value="C##SPRINGUSER" />
    <property name="password" value="spring123" />
</bean>

<!-- 히카리 데이터소스 클래스 빈 등록하기 -->
<bean id="ds" class="com.zaxxer.hikari.HikariDataSource">
    <constructor-arg ref="hikariConfig" />
</bean>


<!-- JDBC 템플릿 클래스 빈 등록하기 -->
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="ds" />
</bean>
```

Spring JDBCTemplate

스프링에서 제공해주는 JDBC Template은 순수하게 JDBC Connection을 얻어서 DB 를 사용하는 것 보다 편하고 자주 쓰는 반복코드를 많이 줄일 수 있다.

```
@Repository
public class BoardDAOImpl implements BoardDAO {

    @Autowired
    private JdbcTemplate jt;
```

DAO 클래스를 빈으로 등록하기 위해 @Repository 어노테이션으로 설정한 후 JdbcTemplate 필드에 @Autowired 어노테이션으로 객체 자동주입을 해준다.

Spring JDBCTemplate

SELECT 문		
메소드	**설명**	**반환타입**
query	결과가 여러행인 경우 사용한다.	List<T> 타입으로 반환해준다.
queryForObject	결과가 하나의 행인 경우 사용한다.	지정한 객체타입으로 반환해준다.

INSERT / UPDATE / DELETE 문		
메소드	**설명**	**반환타입**
update	테이블의 내용만 변경되는 경우 사용한다.	변경된 행의 수를 정수로 반환해준다.

Spring JDBCTemplate

```
jt.query("SELECT * FROM BOARD", new RowMapper</*타입*/>() {

    @Override
    public Object mapRow(ResultSet rs, int rowNum) throws SQLException {

        return null;
    }

});
```

query 메소드는 RowMapper<T> 타입에 해당하는 객체를 인자로 주어 List<T> 타입으로 반환한다. RowMapper 인터페이스의 mapRow 메소드를 오버라이딩 하여 ResultSet 에서 읽은 데이터를 <T> 타입으로 반환을 하도록 작성한다.

Spring JDBCTemplate

```
@Override
public BoardDTO contentView(int id) {

    BoardDTO dto = jt.queryForObject("select * from board where id = ?",
                                     new BoardMapper(), id);
    return dto;
}
```

queryForObject 메소드는 쿼리 결과가 하나의 행일 경우 사용하는 메소드이다. 마찬가지로 RowMapper 인터페이스의 타입의 객체를 주어 해당하는 결과의 타입으로 객체를 얻을 수 있다. (BoardMapper 객체는 RowMapper를 구현한 객체)

Spring JDBCTemplate

```
@Override
public int delete(int id) {

    int result = jt.update("delete from board where id = ?", id);

    return result;
}
```

update()는 데이터베이스 테이블의 내용을 변경할 때 (INSERT, DELETE, UPDATE) 사용하는 메소드이다. 쿼리문을 작성할때 데이터가 들어갈 자리에 ? 로 표시하고 가변인자로 선언된 매개변수의 파라미터를 제공해준다.