JUNIT

JUnit

- JUnit
  - 자바 프로그래밍 언어용 유닛 테스트 프레임워크

- 단위 테스트 목적
  - 프로그램의 각 부분을 고립 시켜서 각각의 부분이 정확하게 동작하는지 확인하는 것
  - 프로그램을 작은 단위로 쪼개서 각 단위가 정확하게 동작하는지 검사
- 단위 테스트 장점
  - 문제 발생 시 정확하게 어느 부분이 잘못되었는지를 파악할 수 있다
  - 안정성이 높아진다.
  - 수정 코드가 정확하게 동작하는지 빠른 시간안에 확인이 가능하다

- 통합 테스트
  - 단위 테스트가 끝난 모듈을 결합해 가며 테스트 하는 방법
  - 모듈들을 좀 더 큰 단위의 집합으로 통합 구성한 후 테스트를 진행
- 통합 테스트 목적
  - 주요 설계 항목들이 기능, 성능, 안정성 요구사항을 잘 구현하고 있는지를 검증하는 것
  - 모듈을 통합하는 과정에서 모듈 간 호환성의 문제를 찾아내기 위해 수행되는 테스트

JUnit

- ❖ @RunWith
  - JUnit 프레임워크 테스트 실행 방법을 설정할때 사용한다.
- ❖ @WebAppConfiguration
  - 통합 테스트를 위해 만들어진 ApplicationContext클래스 객체가 WebApplicationContext의 인스턴스임을 알리는 역할을 하며 @ContextConfiguration의 설정 *.xml과 같이 사용한다.
- ❖ @ContextConfiguration
  - 각종 설정 파일의 위치를 지정할 때 사용한다.
- ❖ @Test
  - @Test가 선언된 메서드는 테스트 메소드가 된다
  - @Test가 붙은 메소드는 개별 테스트가 가능하다
- ❖ @Before
  - @Before가 선언된 메소드는 @Test메소드 실행전에 실행된다.
  - @Test실행전 설정할 값들은 @Before메소드에 설정한다.
- ❖ MockMvc 클래스
  - 웹 애플리케이션을 애플리케이션 서버에 배포하지 않고도 스프링 MVC의 동작을 재현할 수 있는 클래스

dependency

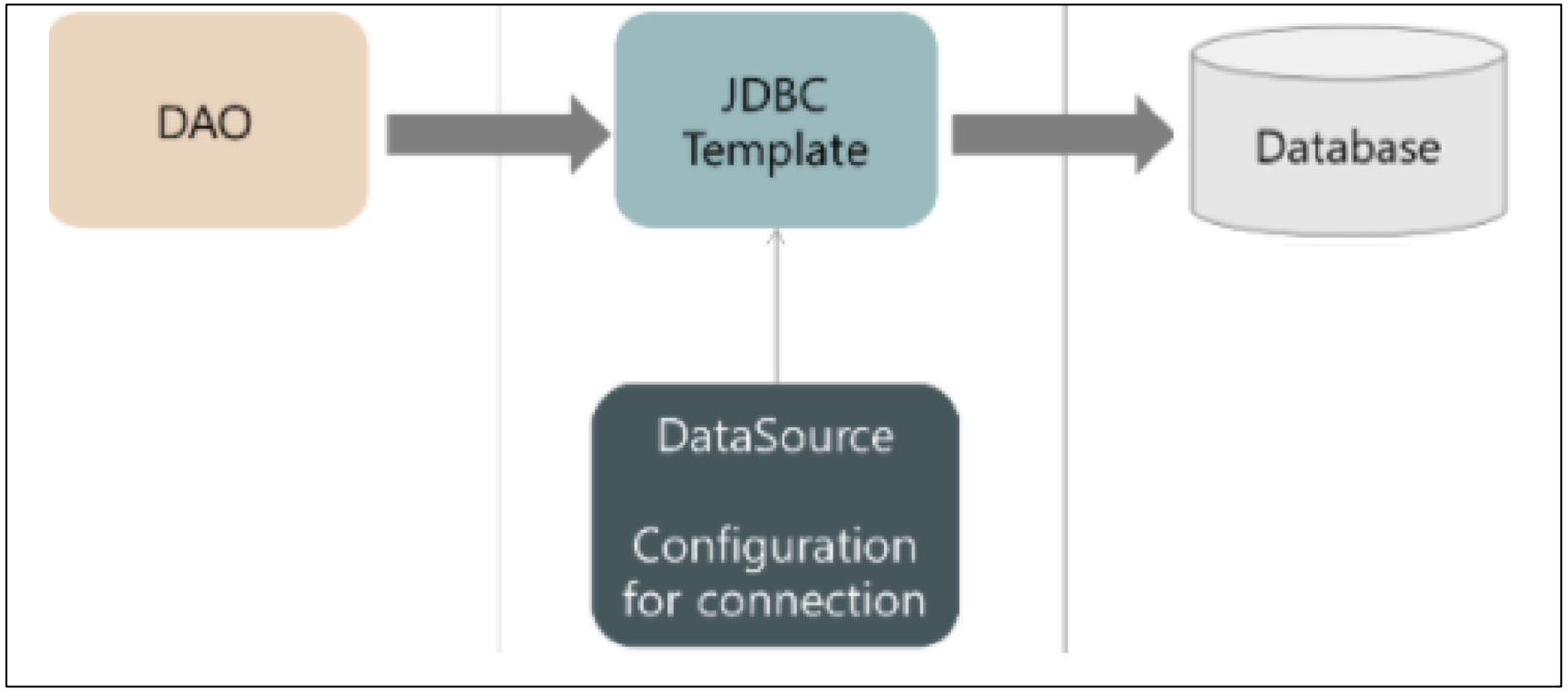
```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

SPRING JDBC & MYBATIS

DataSource

- DataSource는 JDBC 명세의 일부분으로 DB의 관계된 연결정보들을 담고 있으며, bean으로 등록하여 인자로 넘겨준다. 이 과정을 통해 Spring은 DataSource로 DB와의 연결을 획득한다.

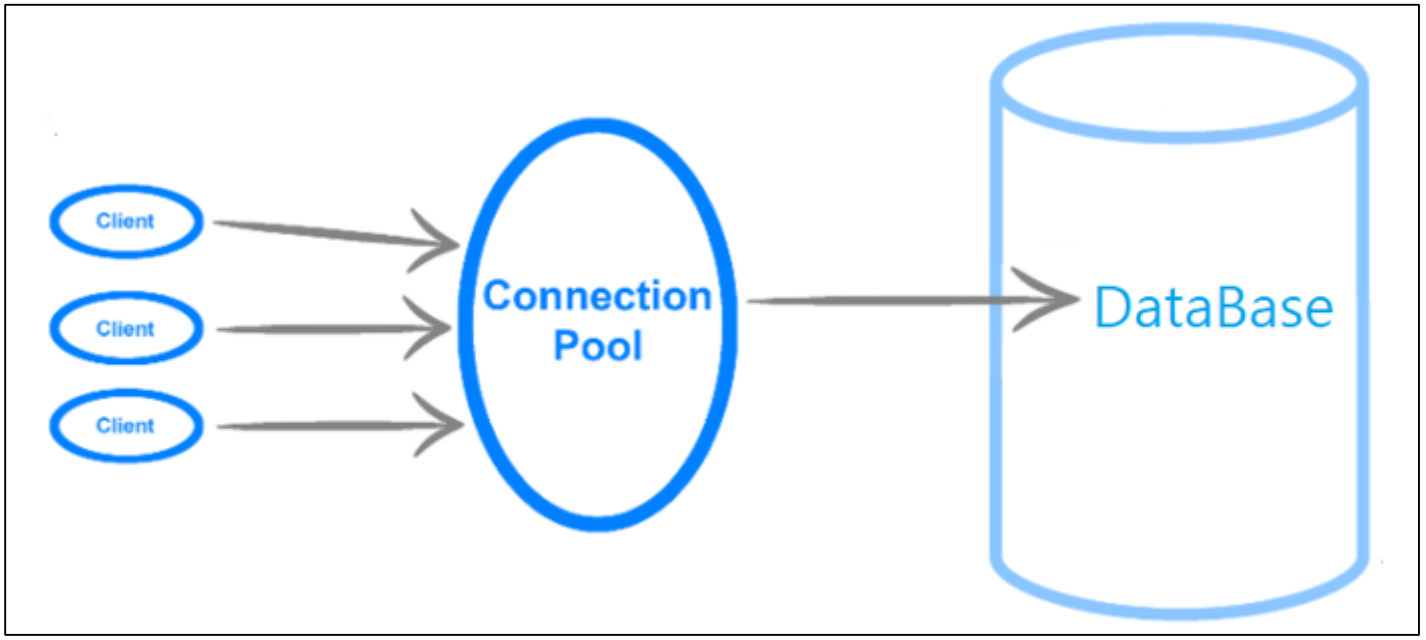
- 즉 DB와의 연결된 bean을 가지게 되며, 해당 bean은 커넥션 풀과 함께 사용된다.



Connection Pool (DBCP)

❖ 커넥션 풀?

웹 컨테이너(WAS)가 실행되면서 DB와 미리 connection(연결)을 해놓은 객체들을 pool에 저장해 두었다가 클라이언트 요청이 오면 connection을 빌려주고, 처리가 끝나면 다시 connection을 반납받아 pool에 저장하는 방식.



❖ **HikariCP**

- DB Connection Pool로써 Zero-Overhead가 특징으로 높은 성능을 자랑하는 DB Connection Pool이다. HikariCP는 미리 정해놓은 만큼의 Connection을 Connection Pool에 담아 놓는다. 그 후 요청이 들어오면 Thread 가 Connection을 요청하고, Hikari는 Connection Pool내에 있는 Connection을 연결해주는 역할을 한다.

dependency

```
<!-- HikariCP -->
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>3.3.1</version>
</dependency>
```

```
<!-- ojdbc6 -->
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0.4</version>
</dependency>
```

```
<!-- spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

```
<!-- mybatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.6</version>
</dependency>
```

```
<!-- 마이바티스와 스프링 연동 라이브러리 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.2</version>
</dependency>
```

데이터 베이스 설정

```
<!-- 히카리 커넥션풀 빈 등록 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
    <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="username" value="spring" />
    <property name="password" value="1234" />
</bean>
<!-- 히카리 데이터소스 빈 등록 -->
<bean id="ds" class="com.zaxxer.hikari.HikariDataSource">
    <constructor-arg ref="hikariConfig"/>
</bean>

<!-- 데이터소스 마이바티스에 등록 및 xml 위치 설정 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="ds"/>
    <property name="mapperLocations" value="classpath:/mappers/**/*Mapper.xml"/>
</bean>
<!-- 마이바티스 xml파일과 dao빈 연결  -->
<mybatis-spring:scan base-package="com.care.root.member.dao"/>
```

Mapper.xml 기본 코드 설정

데이터 베이스 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.care.root.member.dao.MemberDAO">

</mapper>
```