Seoul Software ACademy

웹 개발자 부트캠프 과정

SeSAC x CODINGOn

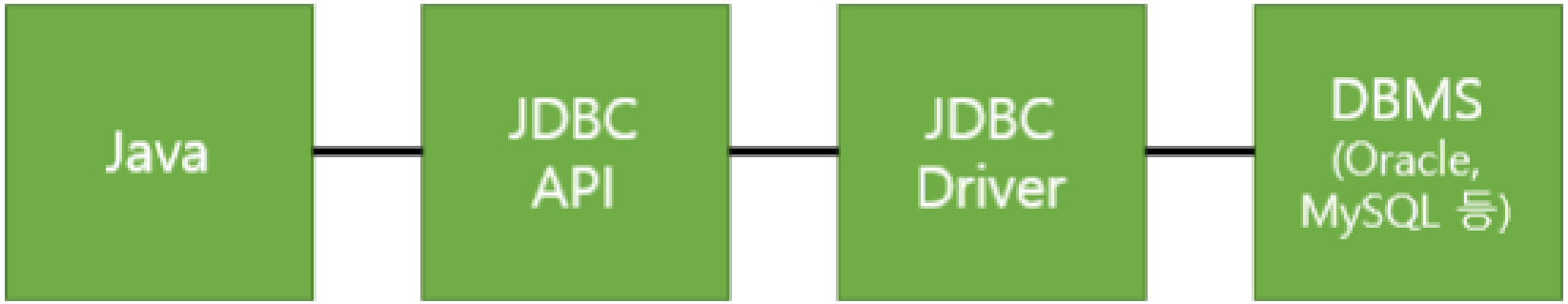
With. 팀 뤼쳐드

Spring Database

JDBC

JDBC 란?

- Java Database Connectivity
- 자바 언어와 DB를 연결해주는 통로로, 자바에서 데이터베이스에 접근할 수 있도록 해주는 자바 API



- 단점: 반복적이고 지루한 코드 작성이 필요하며, SQL 예외 처리, 리소스 관리(연결, 문, 결과 집합 닫기 등) 등의 복잡한 작업을 직접 처리

Spring JDBC

- 정의
  - JDBC의 복잡성을 줄이기 위한 Spring 프레임워크의 일부
  - 데이터베이스 작업을 단순화하고 코드의 양을 줄여줌
- 작동 방식
  - 기존 JDBC 코드의 많은 부분을 추상화하고, 템플릿 디자인 패턴을 이용

SQL Mapper 와 ORM

Sql Mapper와 ORM

- Spring은 대표적으로 두 가지 유형의 데이터베이스와의 상호작용 방식이 존재
- SQL Mapper와 ORM (Object-Relational Mapping).
- 공통점: 두 방식 모두 Java 객체와 데이터베이스 간의 상호 작용

Sql Mapper와 ORM 차이점

- SQL Mapper
  - 주요 목적: SQL 쿼리와 Java 메서드/객체 간의 매핑을 중심으로 함
  - 개발자는 SQL 쿼리를 직접 작성하고, 해당 쿼리의 결과를 Java 객체에 매핑함
  - 동적 SQL 생성, 조건문, 반복문 등의 SQL 작성에 있어서 유연
  - SQL 쿼리의 세밀한 튜닝이 필요할 때 주로 사용
  - 예: MyBatis, JDBCTemplate

Sql Mapper와 ORM 차이점

- ORM (예: Hibernate, JPA)
  - 주요 목적: Java 객체와 데이터베이스 테이블 간의 관계 매핑
  - Java 객체 (Entity)가 데이터베이스의 테이블과 어떻게 매핑 될지를 정의함
  - 대부분의 CRUD 연산에 대한 SQL을 자동으로 생성하며, 개발자는 객체 지향적으로 데이터를 다룰 수 있게 됨
  - 예: Hibernate, JPA

MyBatis

MyBatis 란?

- SQL Mapper 로써, JDBC로 처리하는 상당 부분의 코드와 파라미터 설정 및 결과 매핑을 대신해준다.
- 객체 지향 언어인 자바와 관계형 데이터베이스 프로그래밍을 좀 더 쉽게 할 수 있도록 도와주는 프레임워크

MyBatis 특징

- 쉬운 접근성과 코드의 간결함
  - JDBC의 모든 기능을 MyBatis에서 사용 가능
  - 복잡한 JDBC 코드를 걷어내 깔끔한 소스코드 유지 가능
- SQL 문과 프로그래밍 코드의 분리
  - SQL 문과 프로그래밍 코드가 분리되어 있어 SQL에 변경이 있을 때 다시 컴파일할 필요가 없다.
- 다양한 프로그래밍 언어로 구현 가능
  - Java, C#, .NET, Ruby

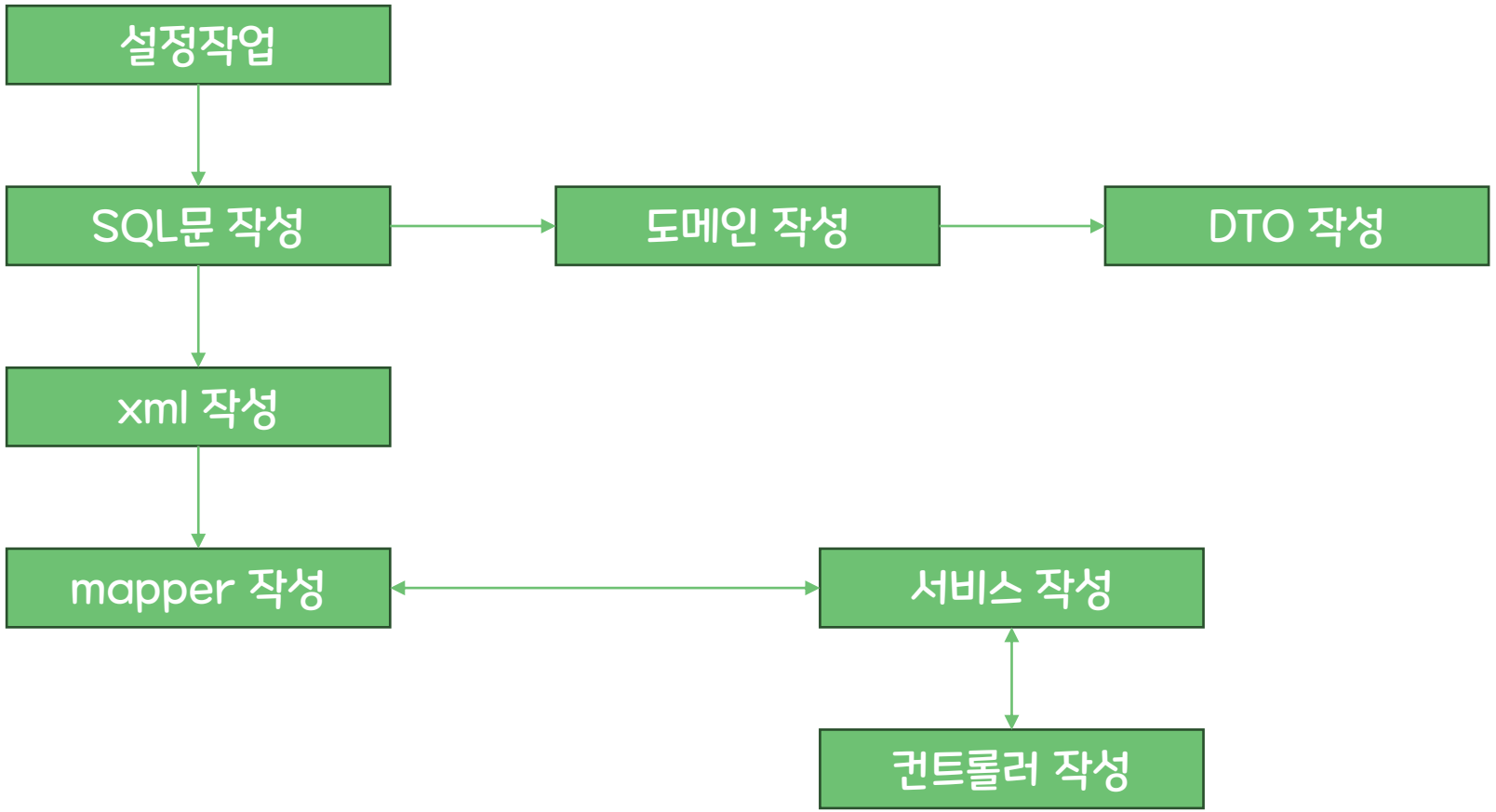
MyBatis 시작하기

- Build.gradle 수정

```
implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.2'
runtimeOnly 'com.mysql:mysql-connector-j'
```

- MyBatis를 사용하기 위해 dependency(라이브러리) 추가

흐름을 파악하기!



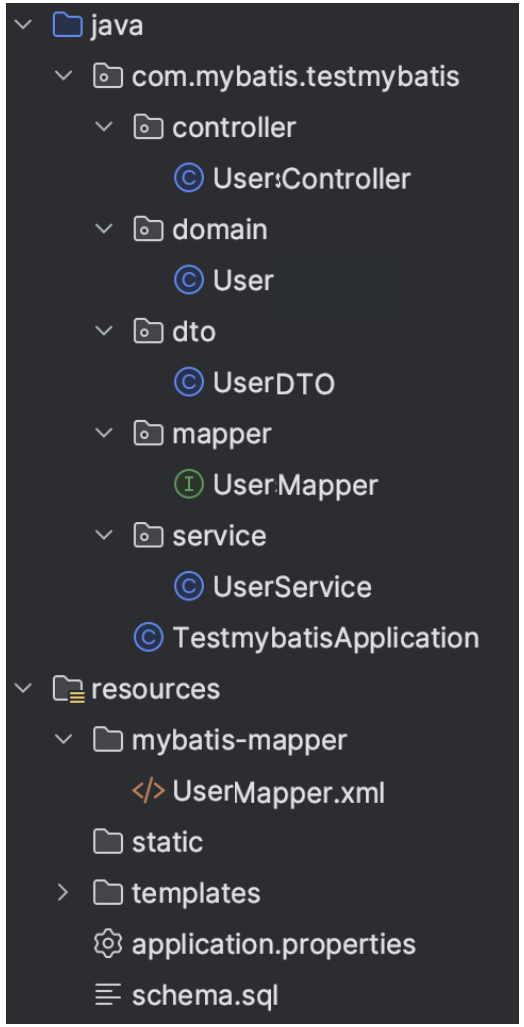
application.properties 수정

- application.properties
  - 런타임 시 다양한 환경에서 동작할 수 있도록 필요한 옵션들을 제공하는데 사용되는 파일

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul
spring.datasource.username=root
spring.datasource.password=1234

mybatis.type-aliases-package=com.spring.boot.mapper
mybatis.mapper-locations=mybatis-mapper/*.xml
```

설정 끝! 개발 시작



- controller
- domain
- dto
- mapper
- service
- resources/mybatis-mapper

schema.sql

```sql
create table users
(
    id bigint not null auto_increment,
    name varchar(255) not null,
    address varchar(255) not null,
    primary key(id)
);
```

User

```
10 usages
@Getter
@Setter
public class User {
    private int id;
    private String name;
    private String nickname;
}
```

UserDTO

```
9 usages
@Getter
@Setter
public class UserDTO {
    private int id;
    private String name;
    private String nickname;
    private int no;
}
```

UserMapper.xml

```xml
<?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="sesac.sesacmybatis.mapper.UserMapper">
    <select id="retrieveAll" resultType="sesac.sesacmybatis.domain.User">
        SELECT user.* FROM user
    </select>
</mapper>
```

〈!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd"〉

UserService

```java
@Service
public class UserService {
    @Autowired
    private UserMapper userMapper;

    1 usage
    public List<UserDTO> getUserList(){
        List<User> result = userMapper.retrieveAll();
        List<UserDTO> users = new ArrayList<~>();

        for ( int i = 0; i < result.size(); i++ ) {
            UserDTO user = new UserDTO();
            user.setId(result.get(i).getId());
            user.setName(result.get(i).getName());
            user.setNickname(result.get(i).getNickname());
            user.setNo(i+1);

            users.add(user);
        }
        return users;
    }
    1 usage
    public void addUser(User user) { userMapper.insertUser(user); }
}
```

UserController

```java
@Controller
public class UserController {
    @Autowired
    UserService userService;

    @GetMapping("/users")
    public String getUsers(Model model) {
        ArrayList<UserDTO> userList = (ArrayList<UserDTO>) userService.getUserList();
        model.addAttribute( attributeName: "list", userList);
        return "user";
    }

    @GetMapping("/user/insert")
    public String getInsertUser(@RequestParam String name, @RequestParam String nickname, Model model) {
        User user = new User();
        user.setName(name);
        user.setNickname(nickname);

        userService.addUser(user);

        model.addAttribute( attributeName: "list",  attributeValue: null);
        return "user";
    }
}
```

MyBatis 동작 원리

- 애플리케이션 시작 시 SqlSessionFactoryBuilder 가 설정 파일을 참고해 SqlSessionFactory 생성
  - SqlSessionFactory : 데이터베이스와의 연결과 SQL의 실행에 대한 모든 것을 가진 중요한 객체
- DB 작업 시 SqlSessionFactory 가 SqlSession 생성
  - SqlSession : Connection을 생성하거나 원하는 SQL을 전달하고, 결과를 Return 해주는 객체
- 생성된 SqlSession을 참고해 mapper 인터페이스 호출
- Mapper 가 SqlSession을 호출해 SQL 실행 후 결과 Return

실습. 게시판 시스템

- C : 게시판 글 작성
- R : 게시판 글 조회 (조건 추가해서 작성자가 a일 때)
- U : 게시판 글 업데이트
- D : 게시판 글 삭제

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
title	varchar(20)	NO		NULL	
content	varchar(100)	NO		NULL	
writer	varchar(10)	NO		NULL	
registered	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

- 지난 시간에 배운 API와 오늘 배운 MyBatis 를 이용해 DB 와 연동되는 게시판 시스템 제작해보기!