Chapter3_개인발표

1. WebMvcConfigurer 인터페이스

스프링 MVC의 개별 설정을 조정할 때 사용
기본 구현은 모두 빈 내용의 구현(재정의가 필요한 메서드만 구현)

```
@Configuration
@ComponentScan
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

  // Case1. WebMvcConfigurer를 구현하였을 경우
  @Override
  public void configureViewResolvers(ViewResolverRegistry registry) {
    registry.jsp("/WEB-INF/",".jsp");
  }
}
```

```
@Configuration
@ComponentScan
@EnableWebMvc
public class WebConfig {

  // Case2. WebMvcConfigurer를 구현하지 않고 직접 Bean을 정의하는 경우
  @Bean
  public ViewResolver customViewResolver() {
      InternalResourceViewResolver internalResourceViewResolver = new InternalResourceViewResolver();
      internalResourceViewResolver.setPrefix("/WEB-INF/");
      internalResourceViewResolver.setSuffix(".jsp");
      return internalResourceViewResolver;
    }
}
```

https://intrepidgeeks.com/tutorial/springboot-actual-development-webmvcconfigurator-interface-47

2. JDBC

> 자바 어플리케이션에서 표준화된 방법으로 다양한 데이터베이스에 접속할 수 있도록 설계된 인터페이스

- 애플리케이션에서는 드라이버 관리자를 통해 필요한 데이터베이스 드라이버를 로드한 다음 해당 데이터베이스에 접속
- 응용 프로그램과 DBMS간의 통신을 중간에서 번역(DBMS의 종류에 상관 X)

JDBC Driver
- 실제 DB관련 기능이 동작하려면 java.sql의 인터페이스들을 상속하여 메서드의 몸체를 구현한 클래스파일들이 필요
- 이 파일들을 JDBC Driver라 부름

JDBC 프로그래밍 코딩 흐름
1. JDBC 드라이버 로드
2. DB연결
3. Statement 생성
4. SQL문 전송 & 결과받기
5. 연결해제

1. JDBC 드라이버 로드

> 데이터베이스에 접속하려면 먼저 해당 데이터베이스의 JDBC 드라이버를 로드

JDBC 드라이버를 로드하는 방법
- jdbc.drivers
  - System.setProperty("jdbc.drivers", "com.mysql.jdbc.Driver");
  - 환경변수를 이용하는 경우 나중에 DriverManager 클래스가 초기화될 때 자동으로 해당 드라이버를 로드
- **Class.forName() 메서드 사용(많이 사용)**
  - Class.forName("com.mysql.jdbc.Drivers");
    - com.mysql.jdbc.Drivers -> Mysql의 JDBC 메인 클래스

2. DB연결

> 드라이버가 로드되면 해당 데이터베이스의 JDBC 드라이버를 이용해 프로그램을 작성할 수 있는 상태가 된 것을 의미실제 데이터베이스와 연결하려면 Connection 클래스의 인스턴스가 필요

- 최근 추세가 DataSource를 DriverManager의 대체제로 사용

JDBC URL

jdbc:[DBMS]:[데이터베이스식별자]jdbc:oracle:thin:@localhost:1521:xepdb1

- DBMS와의 연결을 위한 식별값
- JDBC 드라이버에 따라 형식이 다름

```
String url = "jdbc:oracle:thin:@localhost:1521:xepdb1"
Connection con = DriverManager.getConnection(url, "kim", "1234");
```

- DriverManager.getConnection(url, user, password)
  - String url : 접속할 서버의 URL이며, 프로토콜, 서버주소, DB이름으로 구성
  - String user : DB서버에 로그인할 계정
  - String password : DB서버에 로그인할 비밀번호

DBMS별 JDBC driver명, URL 형식

1. Oracle

- Driver
  - oracle.jdbc.driver.OracleDriver
- URL
  - jdbc:oracle:thin:@HOST:PORT:SID

2. MS-SQL

- Driver
  - com.microsoft.sqlserver.jdbc.SQLServerDriver
- URL
  - jdbc:sqlserver://HOST:[:PORT];DatabaseName=DB

3. MySQL

- Driver
  - com.mysql.jdbc.Driver
- URL
  - jdbc:mysql://HOST[:PORT]/DBNAME[?param=value&param1=value2&..]

3. Statement 생성

> 자바프로그램은 DB쪽으로 SQL문을 전송하고, DB는 처리된 결과를 다시 자바프로그램 쪽으로 전달데이터베이스 연결로부터 SQL문을 수행할 수 있도록 해주는 클래스

```
Statement stmt = con.createStatement();
```

4. SQL문 전송, 결과 받기

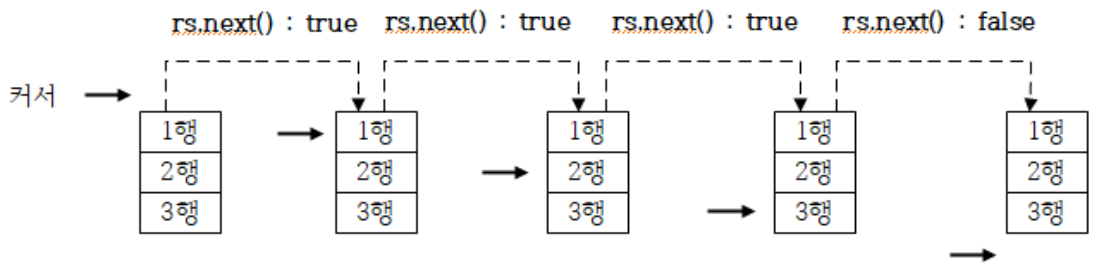
Statement 객체를 이용해 DB서버로 SQL문을 전송하고 처리결과를 받아옴

- ResultSet executeQuery(String sql)
  - select 문 수행 시 사용
  - 메서드가 반환하는 반환값인 ResultSet은 select문을 실행한 후의 결과값을 가지고 있음
- int executeUpdate(String sql)
  - SQL문이 실행된(update, delete, insert) 후 영향받은(처리된) 레코드(데이터)의 개수를 반환

```
...
    Statement stmt = con.createStatement();
    String sql = "insert into test values(' "+request.getParameter("username")+" ',
        '"request.getParmaeter("email")+" ')";
    ResultSet rs = stmt.executeUpdate(sql);
```

※ ResultSet

- executeQuery()메서드에서 실행된 select 문의 결과값을 가지고 있는 객체
- ResultSet 객체의 메서드를 활용해 추출해서 select 결과값을 사용
- ResultSet 객체는 내부적으로 위치를 나타내느 커서(Cursor)가 존재
- Boolean next() 메서드는 커서 다음 레코드가 있는지 판단하여 없으면 false, 있으면 true 반환 후 커서를 다음 레코드(데이터)로 이동
- ResultSet 객체는 현재 커서의 컬럼의 값을 데이터 타입에 따라 추출하는 getter 메서드를 가지고 있음
- getter 메서드의 인자값으로는 칼럼의 이름이나 인덱스(결과값으로 나타난 칼럼의 순서) 값을 지정



커서 동작 예시

```
...

        while(rs.next()){
        int id = rs.getInt("ID");
        String title = rs.getString("TITLE");
        String writer = rs.getString("WRITER_ID");
        Date regDate = rs.getDate("REGDATE");
    }
```

- PreparedStatement
  - SQL문을 미리 만들어 두고 변수를 따로 입력하는 방식
  - 효율성, 유지보수에 용이
  - Statement 클래스 상속
  - SQL문에서 변수가 와야할 위치에 ?만 적어두고 ?자리에는 setXxx()메서드로 값을 설정(Xxx = 자료형)

```
...
    PreparedStatement pstmt = con.prepareStatement("insert into test values(?,?)");
    pstmt.setString(1, request.getParameter("username");
    pstmt.setString(2, request.getParameter("email");
    pstmt.executeUpdate();
```

5. 연결(메모리) 해제

Statement, PreparedStatement 는 사용하지 않을 때 닫아줘야 함.

```
...

    rs.close();
    stmt.close(); //pstmt.close();
    con.close():
```

- 해체 순서는 최근에 사용했던 객체부터 거꾸로 올라가며 해제

3. DataSource

DB와 관계된 커넥션 정보를 담고 있으며 빈으로 등록하여 인자로 넘겨줌

- DB 서버와의 연결을 해줌

- **DB Connection pooling(DBCP)** 기능
  - DB Connection Pooling
    - 자바 프로그램에서 데이터베이스 연결(커넥션 객체를 얻는것)은 오래 걸림
    - 일정량의 Connection 객체를 미리 만들어 저장해두었다가 요청시 사용
    - 속도와 퍼포먼스 증가
    - 커넥션 풀을 관리하고 커넥션 객체를 풀에서 꺼냈다 반납하는 과정을 DataSource가 수행

DataSource를 설정하고 빈에 등록, 주입하는 방법

- DB와의 연결을 위해 DB서버정보(Property)를 설정
- 설정한 property file을 통해 DataSource를 빈으로 등록
  - Spring JDBC를 이용하려면 DB 커넥션을 가져오는 DataSource를 빈으로 먼저 등록해줘야함
- 생성된 DataSource 빈을 Spring JDBC에 주입

4. JDBC 재정리

자바 어플리케이션에서 표준화된 방법으로 다양한 데이터베이스에 접속할 수 있도록 설계된 인터페이스DB에 접근할 수 있도록 Java에서 제공하는 API

- 애플리케이션에서는 드라이버 관리자를 통해 필요한 데이터베이스 드라이버를 로드한 다음 해당 데이터베이스에 접속
- 응용 프로그램과 DBMS간의 통신을 중간에서 번역(DBMS의 종류에 상관 X)

Plain JDBC의 문제점

- 쿼리를 실행하기 전과 후에 많은 코드를 작성해야한다. (연결생성, 명령문, 등등)
- 예외처리코드와 트랜잭션 처리등에 시간과 자원이 소모
  - jdbc에서 발생하는 에러는 Runtime Exception이다. 따라서 모두 예외처리를 해줘야함
- 이러한 문제점을 보완하여 생겨난것이 Spring JDBC

Spring JDBC

> JDBC의 단점을 보완하여 더 편리한 기능을 제공

Spring JDBC가 하는일

- Connection 열기와 닫기
- Statement 준비와 닫기
- Statement 실행
- ResultSet Loop처리
- Exception 처리와 반환
- Transaction 처리

Spring JDBC에서 개발자가 할 일

핵심적으로 해야될 작업만 해주면 나머지는 Framwork가 알아서 처리해준다.

- datasource 설정
- sql문 작성
- 결과 처리

JDBC Template이란?

> Spring JDBC접근 방법중 하나로, 내부적으로 Plain JDBC API를 사용하지만 위와 같은 문제점을 해결한 Spring에서 제공하는 클래스

JdbcTemplate이 제공하는 기능

- 실행 : Insert나 Update같이 DB의 데이터에 변경이 일어나는 쿼리를 수행하는 작업
- 조회 : Select를 이용해 데이터를 조회하는 작업
- 배치 : 여러 개의 쿼리를 한 번에 수행해야 하는 작업
- jdbc template을 사용하면 커넥션 연결/종료와같은 세부적인 작업을 개발자가 직접 처리하지 않아도 되게됨

DataSource설정

- JDBC템플릿 클래스가 JDBC API를 이용하고 DB연동을 처리하려면 DB로부터 커넥션을 얻어야한다. 따라서 JdbcTemplate 객체가 사용할 데이터소스를 빈에 등록하고 주입하도록 한다.

```
@Repository
public class JdbcIngredientRepository implements IngredientRepository {
        private JdbcTemplate jdbc;
        @Autowired
        public JdbcIngredientRepository(JdbcTemplate jdbc) {
                this.jdbc = jdbc;
        }
        @Override
        public Iterable<Ingredient> findAll() {
                return jdbc.query("select id, name, type from Ingredient",
                                this::mapRowToIngredient);
        }
        @Override
        public Ingredient findById(String id) {
                return jdbc.queryForObject(
                                "select id, name, type from Ingredient where id=?",
                                this::mapRowToIngredient, id);
        }
```

<코드 분석>

- @Repository로 인해 Class명의 레포지토리(DataSource)가 자동으로 생성
- @Autowired로 인해 해당 Class명의 레포지토리와 JdbcTemplate이 자동으로 연결
    - 빈으로 등록되어있는 DataSource를 JDBC 템플릿에 주입
- JDBC템플릿은 JDBC Driver를 이용해 DB에 접근한다.