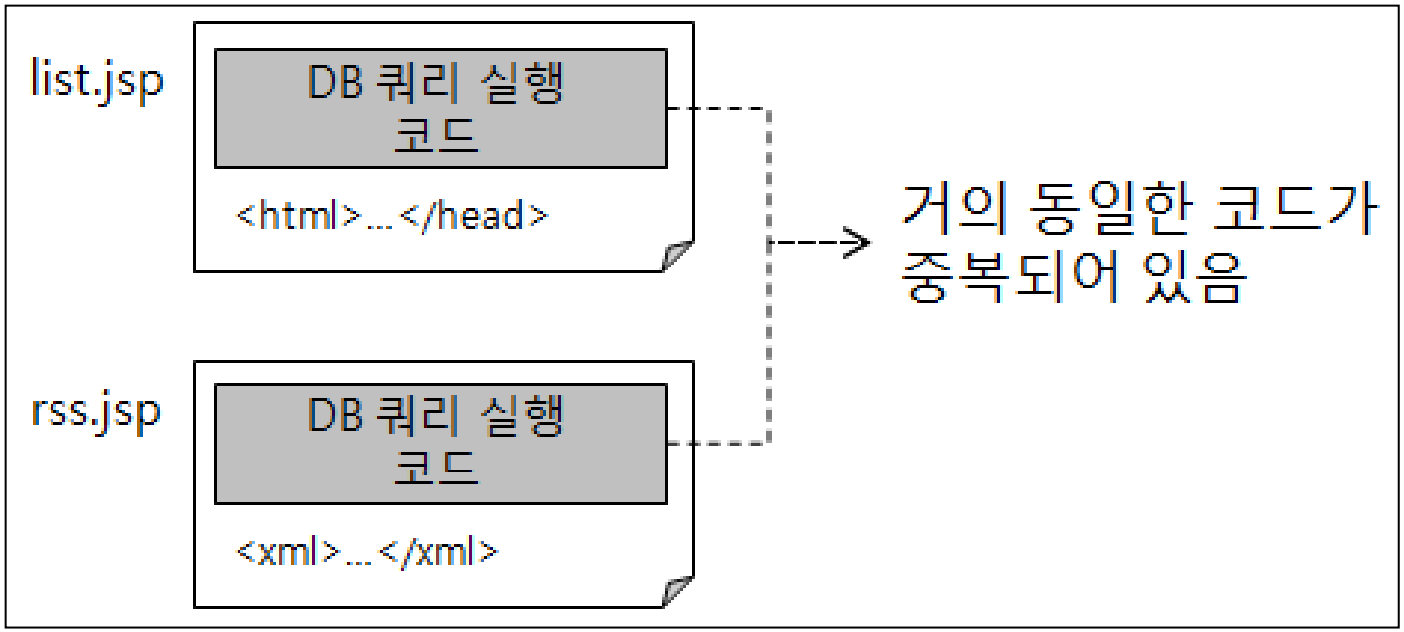
JSP (Java Sever Page)

- 웹 어플리케이션 구조

CONTENTS

■ JSP만을 이용하는 경우의 문제

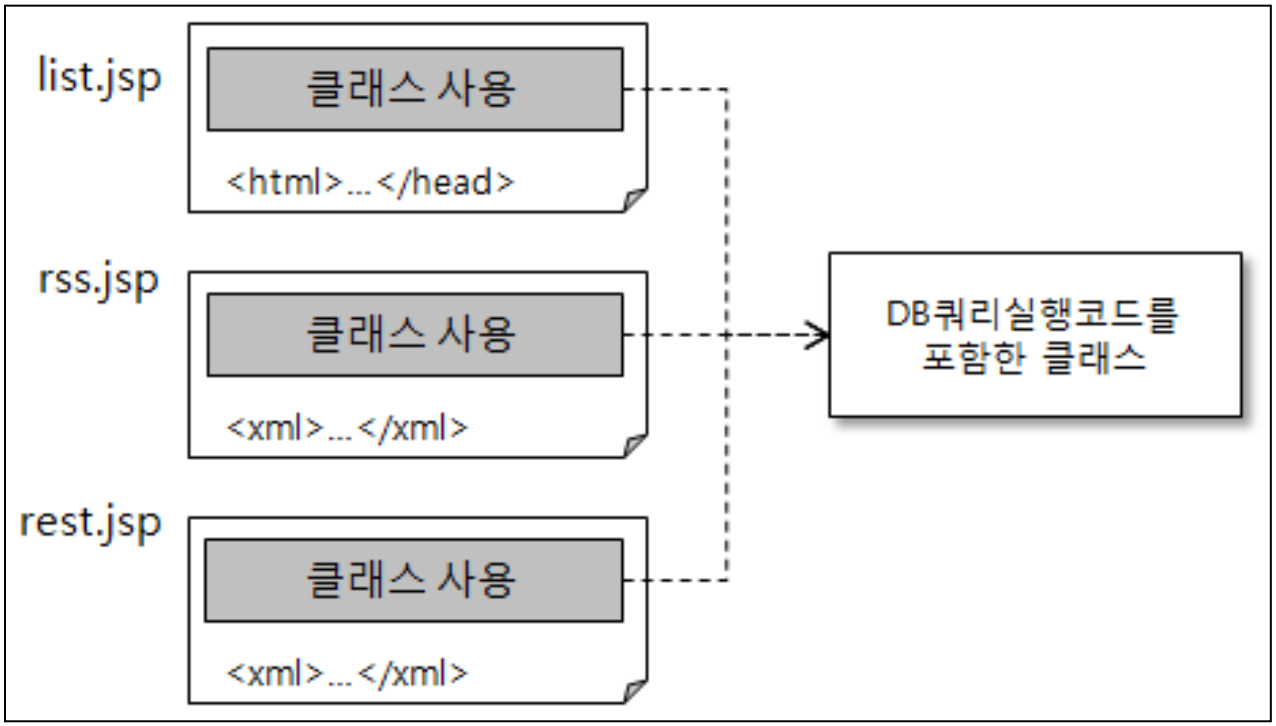
- 동일한 로직을 수행하는 코드가 중복될 가능성이 높음



- 문제점
  - 기능 변경 발생 시 여러 코드에 동일한 수정 반영해 주어야 함
    - 누락될 가능성 발생 → 버그 발생 가능성 높음

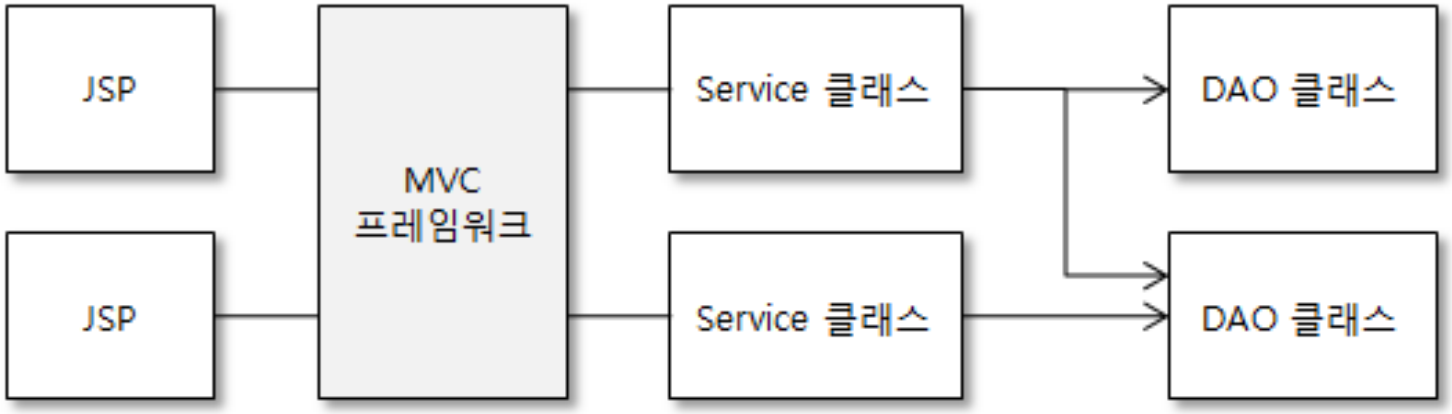
■ 클래스를 이용한 중복 제거

- 클래스를 이용해서 중복된 코드를 한 곳으로 분리



- 화면 요청 처리하는 JSP와 실제 로직을 수행하는 클래스로 분리하는 것이 일반적인 구성

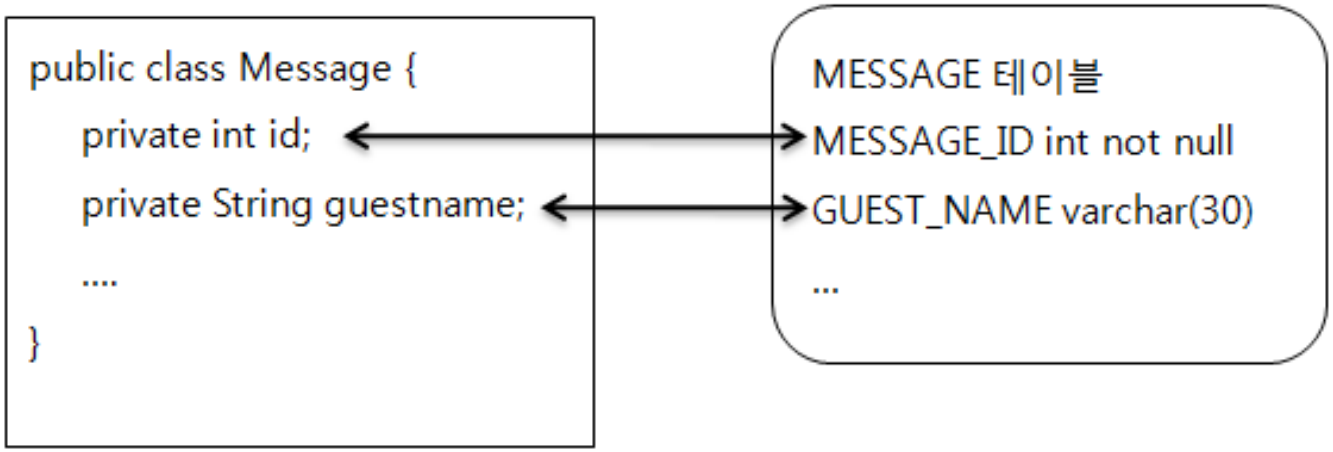
■ 웹 어플리케이션의 일반적인 구성



- Service 클래스 - 사용자의 요청을 처리하는 기능을 제공한다. DAO 클래스를 통해서 DB 연동을 처리한다. 가입 신청 처리, 글 목록 제공 등의 기능을 구현.
- DAO 클래스 - DB와 관련된 CRUD 작업을 처리한다. (SQL 쿼리를 실행)
- JSP(뷰) - Service 클래스가 실행한 결과를 화면에 출력해주거나 Service가 기능을 수행하는 데 필요한 데이터를 전달한다.
- MVC 프레임워크 - 사용자의 요청을 Service에 전달하고 Service의 실행 결과를 JSP와 같은 뷰에 전달한다. 스프링 MVC나 스트러츠와 같은 프레임워크가 MVC 프레임워크에 해당된다.

DAO의 구현

- CRUD를 위한 메서드 정의
  - insert() 메서드 - INSERT 쿼리를 실행한다.
  - select() 메서드 - SELECT 쿼리를 실행한다. 검색 조건에 따라서 한 개 이상의 select() 메서드를 제공한다.
  - update() 메서드 - UPDATE 쿼리를 실행한다.
  - delete() 메서드 - DELETE 쿼리를 실행한다.
- 테이블과 매핑될 클래스 작성



DAO에 Connection 전달 방법

- DAO 클래스의 메서드에서 직접 Connection을 생성
  - 1개 이상의 DAO 메서드 호출을 하나의 트랜잭션에서 처리 불가

- DAO 객체를 생성할 때 생성자에서 Connection을 전달 받기
  - 1개 이상의 DAO 메서드 호출을 트랜잭션으로 묶을 수 있음
  - 불필요하게 DAO 객체를 매번 생성하는 단점

- DAO 클래스의 메서드 파라미터로 Connection을 전달 받기
  - 1개 이상의 DAO 메서드 호출을 트랜잭션으로 묶을 수 있음
  - DAO 객체를 매번 생성할 필요 없음

서비스 클래스의 구현

- 사용자의 요청을 처리하기 위한 기능을 구현
- DAO를 통해서 데이터에 접근
- 코드 예

```
public class ReadArticleService {
    public Article read(int articleId) {
        Connection conn = null;
        try {
            conn = ...// Conneciton 구함
            conn.setAutoCommit(false);
            // 기능을 구현하는 데 필요한 DAO를 구한다.
            ArticleDao articleDao = ArticleDaoProvider.getInstance().getDao();
            Article article = articleDao.selectById(conn, articleId);
            if (article == null) { throw new ArticleNotFoundException(articleId); }
            article.increaseReadCount();
            articleDao.updateReadCount(conn, article);
            conn.commit();
            return article;
        } catch (Exception ex) {
            JdbcUtil.rollback(conn);
            throw new ArticleServiceException("에러 발생:"+ex.getMessage());
        } finally {
            JdbcUtil.close(conn);
        }
    }
```

서비스 클래스와 트랜잭션

- **서비스가 제공하는 기능이 트랜잭션 필요시**

  Connection.setAutoCommit(false)로 트랜잭션 처리

```
try {
    conn = ...// Connection을 구한다.
    conn.setAutoCommit(false); // 로직을 수행하기 전에 트랜잭션 시작
    ...
    ...
    conn.commit(); // 로직을 수행한 뒤 트랜잭션 커밋
} catch(SQLException ex) {
    if (conn != null)
        try {
            conn.rollback(); // 로직 실행 도중 예외가 발생한 트랜잭션 롤백
        } catch(SQLException ex1) {}
    ...
} finally {
    ...
}
```

서비스 클래스와 예외 처리

- **서비스 클래스의 메서드는 서비스가 제공하는 기능에 알맞은 예외를 발생시키는 것이 좋음**

```
public Article readArticle(..) throws SQLException { ... }
```

↓

```
public Article readArticle(..) throws ArticleNotFoundException { ... }
```

- **서비스 메서드 내부에서의 처리**

■ 싱글톤 패턴을 이용한 서비스/DAO 구현

- **싱글톤 패턴 - 클래스 당 한 개의 객체만 생성되도록 제약하는 구현 패턴**
  - **구현 방식**

```
public class ReadArticleService {
    // 유일한 객체를 정적 필드에 저장
    private static ReadArticleService instance = new ReadArticleService();

    // 유일한 객체에 접근할 수 있는 정적 메서드 정의
    public static ReadArticleService getInstance() {
        return instance;
    }

    // 생성자를 private으로 설정해서 외부에서 접근하지 못함
    private ReadArticleService() {}
    ...
}
```

- **서비스나 DAO는 매번 객체를 생성할 필요가 없으므로,**
  싱글톤 패턴 적용하기에 적합

ConnectionProvider의 구현

- 서비스에서 Connection을 직접 생성할 경우
  - JDBC URL 등이 변경될 경우 모든 코드를 변경할 가능성
  - 즉, 유지보수에 불리할 수 있음

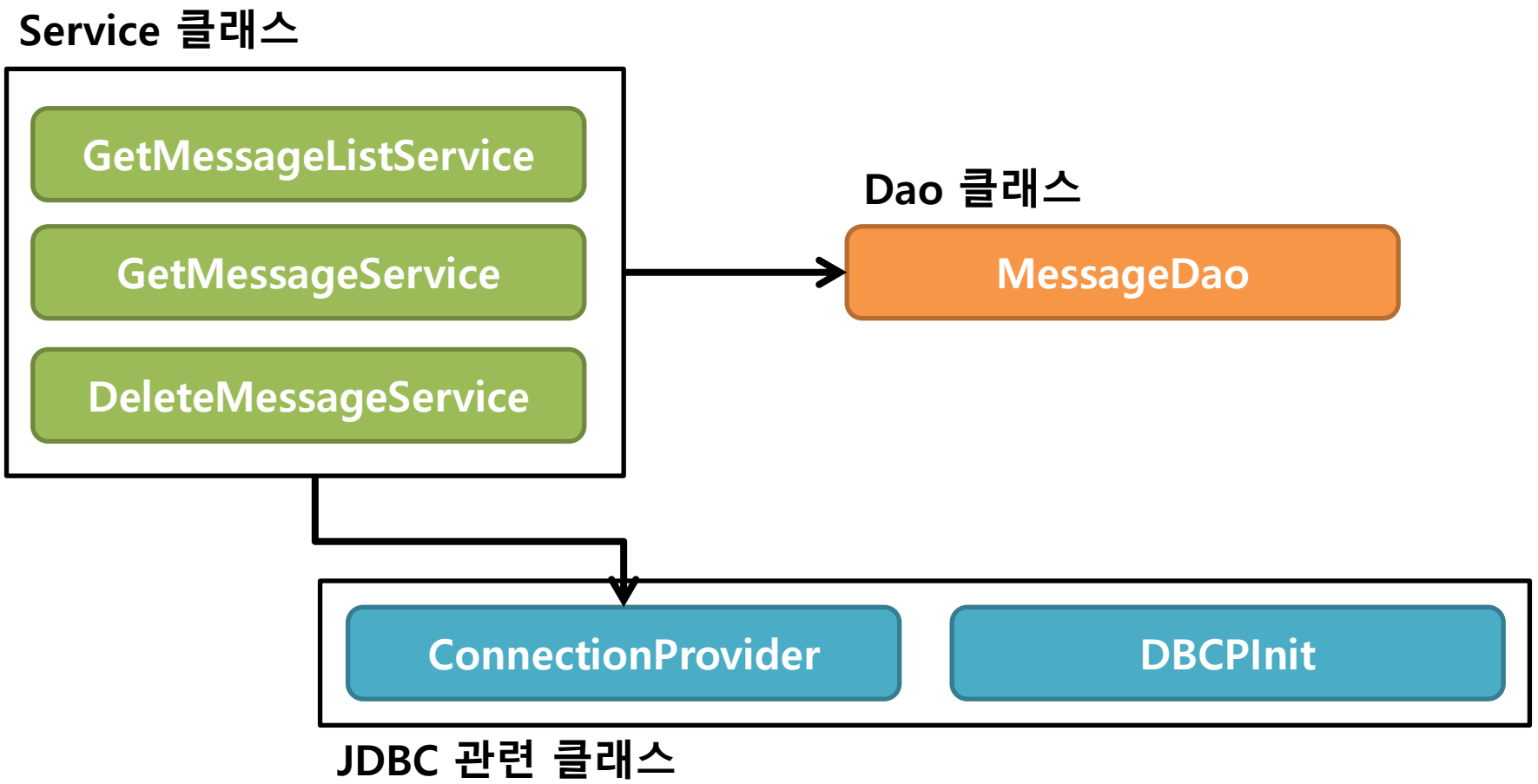
- 이런, 이유로 Connection을 제공하는 역할을 가진 클래스를 별도 작성

```
public class ConnectionProvider {
    public static Connection getConnection() throws SQLException {
        return ….; // 커넥션 생성 코드
    }
}
```

  - DataSource를 이용하기도 함

방명록의 구성 예

- 클래스 구조

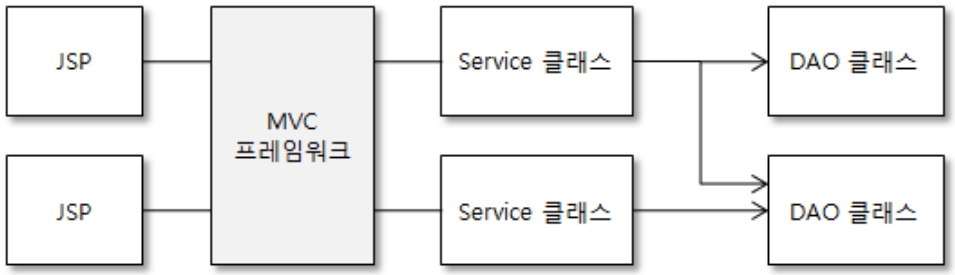


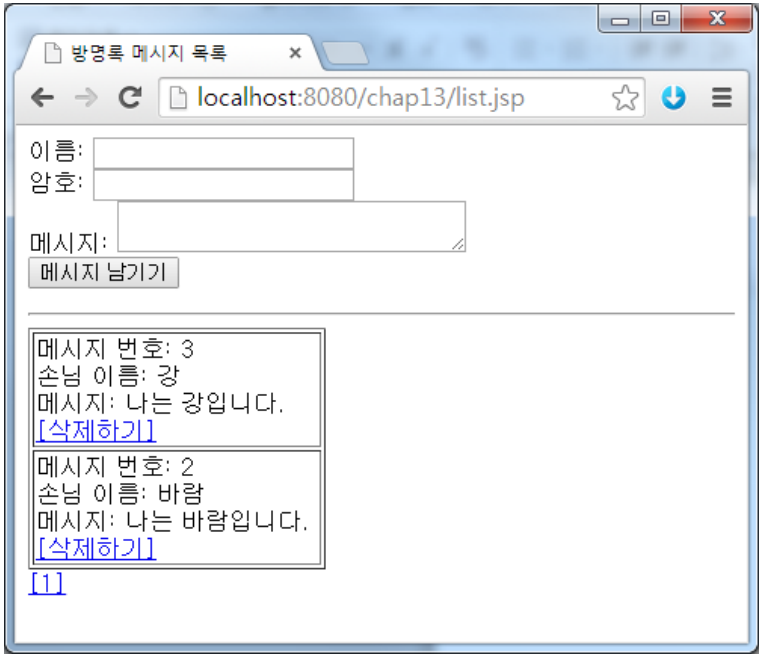
MISSION

팀장 : 내일까지 방명록을 구현해주세요.

SPEC

- 방명록은 이름, 암호, 메시지 를 입력 받아 저장.
- 게시물 리스트는 페이지당 3개 게시물이 노출되도록 처리.
- 게시물의 삭제는 암호를 확인 후 처리.
- 어플리케이션의 구조는 아래의 구조 형태로 개발.
  - VIEW(JSP), MODEL, SERVICE, DAO 각 영역별 역할에 맞게 구성.





구현 1 : DBMS 준비

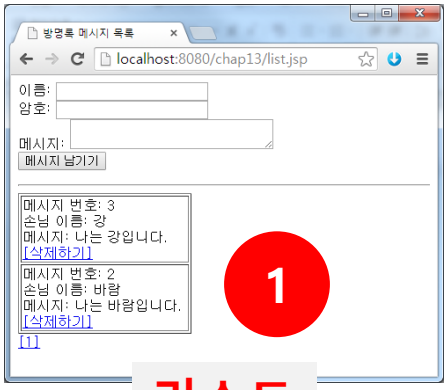
- 구현하고자 하는 어플리케이션은 DBMS를 통해서 데이터 입출력을 하는 프로그램.
- 프로젝트에서 구현해야 하는 DBMS를 준비, 라이브러리 API 준비
- 데이터 저장을 위한 DATABASE 생성 및 기능 구현을 위한 TABLE 생성
- VIEW 에 표현해야 하는 데이터 질의를 위한 select 구문, 입력/삭제/수정을 위한 쿼리문 구성
- 프론트 개발에 들어가기 전에 쿼리문을 구성해 하는 것이 좋다.
  - 개발과정에서 쿼리문은 추가/삭제/ 수정 됨.

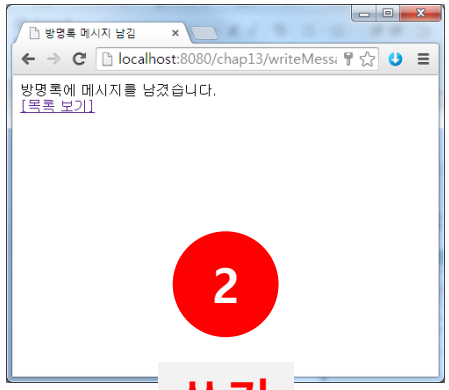
구현 2 : 개발에 필요한 파일 준비

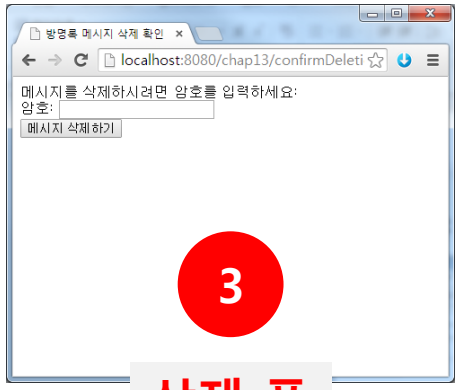
- 개발에 필요한 API (jar) 파일 준비
  - ORACLE, JDBD POOL, 등.
- DATABASE 연결 테스트
  - 컨넥션풀 초기화
  - 초기화를 선 작업하는 이유
    : 구현 해야 하는 부분들이 모두 DB와 연결되어 처리되어야 함.
  - 테스트를 위해 더미 데이터 입력 후 테스트

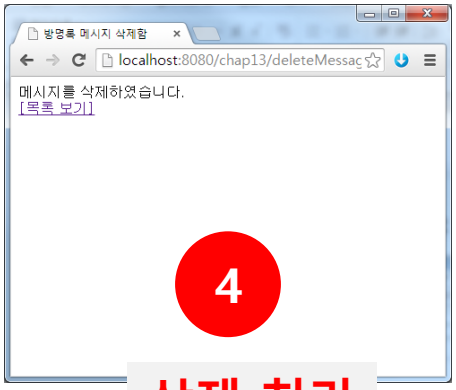
구현 3 : 세부 스팩 확인

- 화면 (VIEW) : 화면의 표현은 Model 객체를 참조해서 한다.









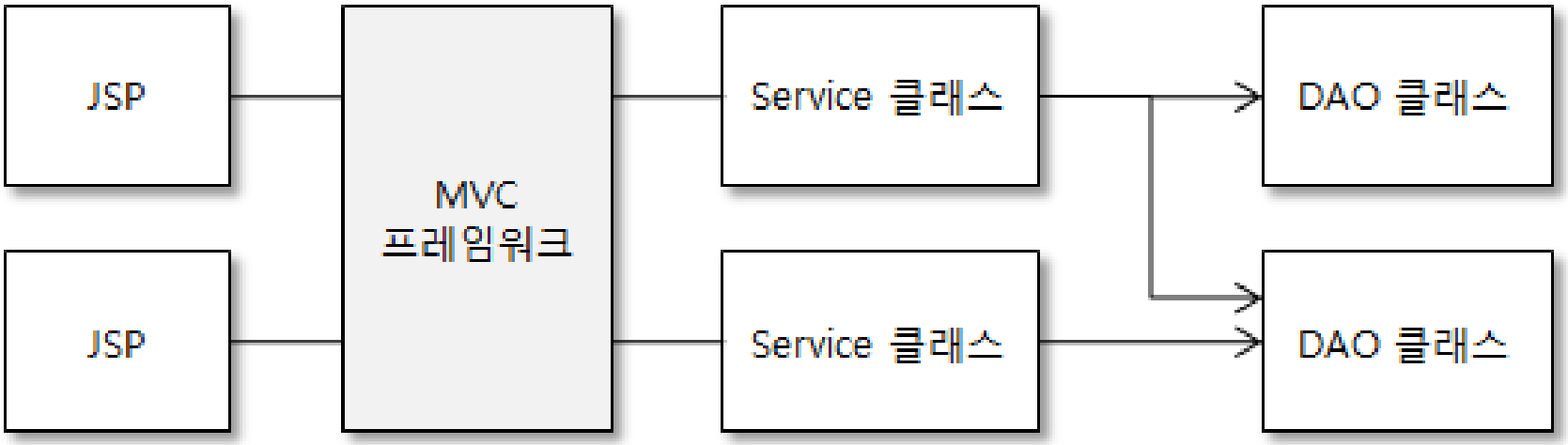
- Model
  - List, Contents
- Service : 사용자가 요청하는 기능을 구현 해서 Model 객체에 담는다.
  - View (LIST, Contents), delete 등
  - 참조 객체 생성은 중복 생성이 되지 않도록 싱글톤 처리.
- DAO : 서비스에 요청하는 DB 작업
  - Insert, select(contents), select(List), delete

구현 4 : 구현 순서 결정

- 어느 부분은 먼저 해야 할까? (화면 단위로 구현하는 것이 편하다.)

  1. ①　　2. ②　　3. ③　　4. ④

- 화면 구성 단위도 구조에 맞추어 개발.



구현 5 : 세부 구현 (입력)

- Model 클래스 구성
- Service (요청에 응답 메서드 구현)
- Dao (insert 메서드 구현)
- View
  - form 구현
  - sevice insert 객체 생성, insert 메서드 호출

구현 6 : 세부 구현 (삭제)

- Model 클래스 : 불 필요
- Service (요청에 응답 메서드 구현)
- Dao (select 메서드, delete 메서드)
- View
  - form 구현
  - sevice delete 객체 생성, delete 메서드 호출

구현 7 : 세부 구현 (리스트)

- Model 클래스 : Message
  MessageListView
- Service (요청에 응답 메서드 구현)
- Dao (select 메서드, delete 메서드)
- View
- sevice MessageListView 객체 생성, getMessageList 메서드 호출,
  getMessageList()