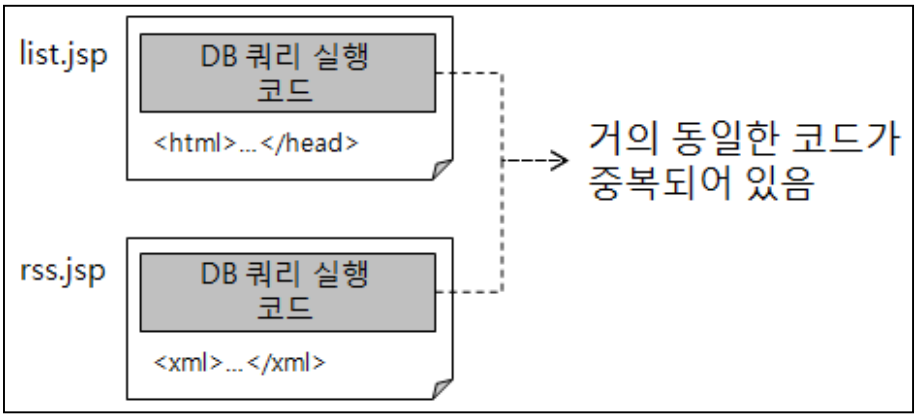
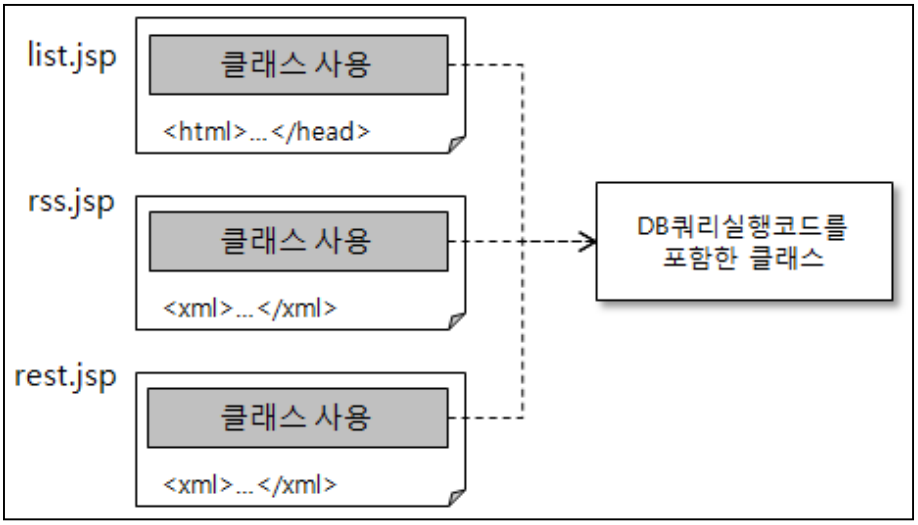
15장 웹 어플리케이션의 일반적인 구성

15.1 웹 어플리케이션의 전형적인 구성 요소

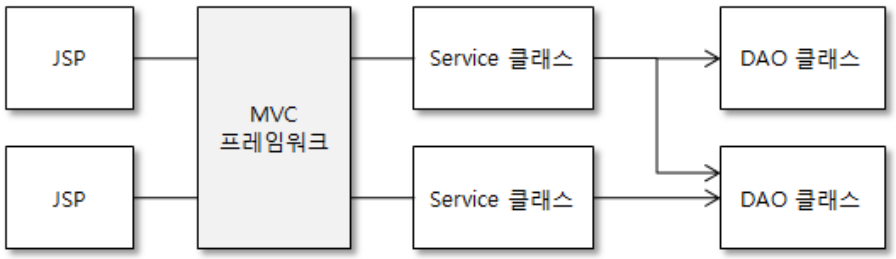
■ JSP만을 이용하는 경우의 문제
- 동일한 로직을 수행하는 코드가 중복될 가능성이 높음
- 기능 변경 발생 시 여러 코드에 동일한 수정 반영해 주어야 함
- 누락될 가능성 발생 → 버그 발생 가능성 높음



■ 클래스를 이용한 중복 제거
- 클래스를 이용해서 중복된 코드를 한 곳으로 분리
- 화면 요청 처리하는 JSP와 실제 로직을 수행하는 클래스로 분리하는 것이 일반적인 구성



15.1.1 웹 어플리케이션의 주요 구성 요소



- JSP(뷰) - Service 클래스가 실행한 결과를 화면에 출력해주거나 Service가 기능을 수행하는데 필요한 데이터를 전달한다.

- MVC 프레임워크 - 사용자의 요청을 Service에 전달하고 Service의 실행 결과를 JSP와 같은 뷰에 전달한다. 스프링 MVC나 스트러츠와 같은 프레임워크가 MVC 프레임워크에 해당된다.

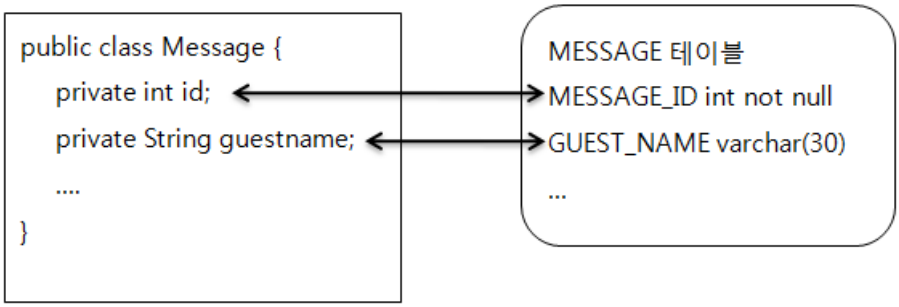
- Service 클래스 - 사용자의 요청을 처리하는 기능을 제공한다. DAO 클래스를 통해서 DB 연동을 처리한다. 가입 신청 처리, 글 목록 제공 등의 기능을 구현.

- DAO 클래스 - DB와 관련된 CRUD(Create Read Update Delete) 작업을 처리한다. (SQL 쿼리를 실행)

15.1.2 데이터 접근 객체(Data Access Object)의 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(Java Bean) 작성




```
// DAO 클래스의 insert() 메서드 작성 예
public void insert(Connection conn, Message message) throws SQLException {
   PreparedStatement pstmt = null;
   try {
      pstmt = conn.prepareStatement("insert into message values (?, ?, ?)");
      pstmt.setInt(1, message.getId());
      pstmt.setString(2, message.getGuestname());
      ...
      return pstmt.executeUpdate();
   } finally {
      if (pstmt != null) try { pstmt.close(); } catch (SQLException ex) {}
   }
}
```

(1) DAO에서 Connection에 접근하는 방식

- DAO 클래스의 메서드에서 직접 Connection을 생성
  - 1개 이상의 DAO 메서드 호출을 하나의 트랜잭션을 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

```
// 1. DAO 클래스의 메서드에서 직접 Connection을 생성
public class MessageDao {
   public Message selectById(int messageId) throws SQLException {
      Connection conn = null;
      PreparedStatement pstmt = null;
      ResultSet rs = null;
      try {
         conn = DriverManager.getConnection(jdbcUrl, userId, userPassword);
         ...
      } finally {
         ...
         if (conn != null) try { conn.close(); } catch(SQLException ex) {}
      }
   }
}

// 2. DAO 클래스를 생성할 때 생성자로 Connection을 전달받기
public class MessageDao {
   private Connection conn;
   public MessageDao(Connection conn) {
      this.conn = conn;
   }
   public Message selectById(int messageId) throws SQLException {
      PreparedStatement pstmt = null;
      ResultSet rs = null;
      try {
         psmt = conn.prepareStatement(query, userId, password);
         ...
      } finally { ... }
   }
}

// 3. DAO 클래스의 메서드 파라미터로 Connection을 전달받기
public class MessageDao {
   public Message selectById(Connection conn, int messageId) throws SQLException {
      PreparedStatement pstmt = null;
      ResultSet rs = null;
      try {
         psmt = conn.prepareStatement(query, userId, password);
         ...
      } finally { ... }
   }
}
```

(2) 간단한 close() 및 rollback() 처리 코드를 위한 JdbcUtil

■ JdbcUtil 클래스는 close() 메서드를 호출하는 코드를 제공하므로 이 클래스를 이용하면 작성할 코드를 줄일 수 있다.

```
//  close() 메서드를 호출해서 자원을 반환한다.
try {
   ...
} finally {
   if (rs != null) try { rs.close(); } catch(SQLException ex) {}
   if (pstmt != null) try { pstmt.close(); } catch(SQLException ex) {}
   if (conn != null) try { conn.close(); } catch(SQLException ex) {}
}

// JdbcUtil 클래스를 이용한다.
try {
   ...
} finally {
   JdbcUtil.close(rs);
   JdbcUtil.close(pstmt);
   JdbcUtil.close(conn);
}
```

[chap15\src\jdbc\JdbcUtil.java]

```
01    package jdbc;
02
03    import java.sql.Connection;
04    import java.sql.ResultSet;
05    import java.sql.SQLException;
06    import java.sql.Statement;
07
08    public class JdbcUtil {
09
10              public static void close(ResultSet rs) {
11                        if (rs != null) {
12                                  try {
13                                            rs.close();
14                                  } catch (SQLException ex) {
15                                  }
16                        }
17              }
18
19              public static void close(Statement stmt) {
20                        if (stmt != null) {
21                                  try {
22                                            stmt.close();
23                                  } catch (SQLException ex) {
24                                  }
25                        }
26              }
27
28              public static void close(Connection conn) {
29                        if (conn != null) {
30                                  try {
31                                            conn.close();
32                                  } catch (SQLException ex) {
33                                  }
34                        }
35              }
36
37              public static void rollback(Connection conn) {
38                        if (conn != null) {
39                                  try {
40                                            conn.rollback();
41                                  } catch (SQLException ex) {
42                                  }
43                        }
```

```
44             }
45    }
```

15.1.3 서비스 클래스의 구현

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

```
public class ReadArticleService {
    public Article read(int articleId) {
        Connection conn = null;
        try {
            conn = ...// Conneciton 구함
            conn.setAutoCommit(false); // 트랜잭션 시작

            // 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);
        }
    }
}
```

(1) 서비스 클래스와 트랜잭션 처리

- 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 {
    ...
}
```

(2) 서비스 클래스의 익셉션 처리

- 서비스 클래스의 메서드는 서비스가 제공하는 기능에 알맞은 예외를 발생시키는 것이 좋음
- 서비스 메서드 내부에서의 처리

```
public void deleteMessage(int messageId) {
   Connection conn = null;
   try {
      conn = ...; // Connection을 구한다.
      conn.setAutoCommit(false);
      ...
   } catch (SQLException ex) {
      JdbcUtil.rollback(conn);
      throw new DeleteFailException("삭제에 실패했습니다.", ex);
   } finally {
      JdbcUtil.close(pstmt);
   }
}
```

15.1.4 싱클톤(Singleton) 패턴을 이용한 구성 요소 구현

- 싱글톤 패턴 - 클래스 당 한 개의 객체만 생성되도록 제약하는 구현 패턴
- 서비스나 DAO는 매번 객체를 생성할 필요가 없으므로, 싱글톤 패턴 적용하기에 적합

```
// 객체를 여러 번 만들더라도 실제로 수행하는 기능은 동일하다.
ReadArticleService service1 = new ReadArticleService();
ReadArticleService service2 = new ReadArticleService();
Article article1 = service1.read(messageId);
Article article2 = service2.read(messageId);
Article article3 = service1.read(messageId);

// 한 개의 객체를 재사용하도록 구현한다. -> Singleton
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() {}
    ...
}
```

15.1.5 Connection을 제공하는 ConnectionProvider 만들기

- 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
- DataSource를 이용하기도 함

```
Connection conn = null;
try {
   conn = ConnectionProvider.getConnection();
}
```

[chap15\src\jdbc\connection\ConnectionProvider.java]

```
01    package jdbc.connection;
02
03    import java.sql.Connection;
04    import java.sql.DriverManager;
05    import java.sql.SQLException;
06
07    public class ConnectionProvider {
08
09              public static Connection getConnection() throws SQLException {
10                        return DriverManager.getConnection(
11                                                  "jdbc:apache:commons:dbcp:guestbook");
12              }
13    }
```

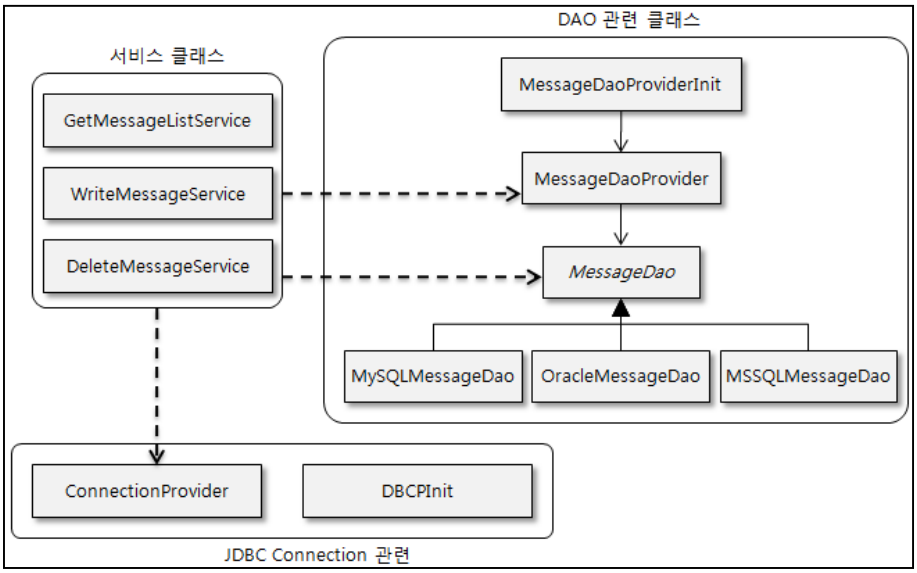
15.2 방명록 구현

15.2.1 방명록을 구성하는 클래스의 구조

■ 서비스 관련 클래스
- GetMessageListService: 요청한 페이지 번호에 포함된 메시지 목록을 구한다.
- WriteMessageService: 메시지를 작성하는 기능을 제공한다.
- DeleteMessageService: 작성한 메시지를 삭제하는 기능을 제공한다.

■ DAO 관련 클래스
- MessageDao: guestbook_message 테이블에 대한 쿼리를 실행한다.

■ JDBC Connection 관련 클래스
- ConnectionProvider
- DBCPInit
- JdbcUtil



15.2.2 데이터베이스와 테이블 생성

■ MySQL 워크벤치에서 MySQL에 root 계정으로 연결한 뒤 다음 쿼리를 실행시킨다.

```
-- root 계정으로 실행
create database guestbook default character set utf8;

create user 'jspexam'@'localhost' identified by 'jsppw';
grant all privileges on guestbook.* to 'jspexam'@'localhost';

create user 'jspexam'@'%' identified by 'jsppw';
grant all privileges on guestbook.* to 'jspexam'@'%';

-- jspexam 계정으로 실행
create table guestbook_message (
    message_id int not null auto_increment primary key,
    guest_name varchar(50) not null,
    password varchar(10) not null,
    message text not null
) engine=InnoDB default character set = utf8
```

15.2.3 이클립스 프로젝트 생성과 필요 모듈 복사

■ 이클립스에서 File > Import > Existing Projects into Workspace 메뉴를 이용하여 프로젝트(chap15)를 임포트한다.

15.2.4 커넥션 풀 설정 위한 DBCPInit 클래스 구현과 web.xml 설정

■ DBCPInit 클래스에서 JDBC 드라이버를 로딩하고 커넥션 풀을 초기화하므로, 톰캣을 실행할 때 DBCPInt을 실행하도록 web.xml 파일에 DBCPInit을 등록한다.

[chap15/WebContent/WEB-INF/web.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://xmlns.jcp.org/xml/ns/javaee"
          xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
          version="3.1">

          <servlet>
                    <servlet-name>DBCPInit</servlet-name>
                    <servlet-class>jdbc.DBCPInit</servlet-class>
                    <load-on-startup>1</load-on-startup>
          </servlet>

</web-app>
```

15.2.5 Message 클래스 작성

- guestbook_message 테이블에서 읽어온 값을 저장하거나 또는 사용자가 입력한 값을 DAO에 전달할 때 사용되는 클래스이다.

[chap15/src/guestbook/model/Message.java]

```
package guestbook.model;

public class Message {

          private int id;
          private String guestName;
          private String password;
          private String message;

          public int getId() {
                    return id;
          }

          public void setId(int id) {
                    this.id = id;
          }

          public String getGuestName() {
                    return guestName;
          }

          public void setGuestName(String guestName) {
                    this.guestName = guestName;
          }

          public String getPassword() {
                    return password;
          }

          public void setPassword(String password) {
                    this.password = password;
          }

          public String getMessage() {
                    return message;
          }

          public void setMessage(String message) {
                    this.message = message;
```

```
	}

	public boolean hasPassword() {
		return password != null && !password.isEmpty();
	}

	public boolean matchPassword(String pwd) {
		return password != null && password.equals(pwd);
	}
}
```

15.2.6 MessageDao 클래스 구현

■ MessageDao 클래스는 guestbook_message 테이블에 대한 CRUD 메서드를 제공한다.

[chap15/src/guestbook/dao/MessageDao.java]

```
package guestbook.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import guestbook.model.Message;
import jdbc.JdbcUtil;

public class MessageDao {
	private static MessageDao messageDao = new MessageDao();
	public static MessageDao getInstance() {
		return messageDao;
	}

	private MessageDao() {}

	public int insert(Connection conn, Message message) throws SQLException {
		PreparedStatement pstmt = null;
		try {
			pstmt = conn.prepareStatement(
					"insert into guestbook_message " +
					"(guest_name, password, message) values (?, ?, ?)");
			pstmt.setString(1, message.getGuestName());
			pstmt.setString(2, message.getPassword());
			pstmt.setString(3, message.getMessage());
			return pstmt.executeUpdate();
		} finally {
			JdbcUtil.close(pstmt);
		}
	}

	public Message select(Connection conn, int messageId) throws SQLException {
		PreparedStatement pstmt = null;
		ResultSet rs = null;
		try {
			pstmt = conn.prepareStatement(
					"select * from guestbook_message where message_id = ?");
			pstmt.setInt(1, messageId);
			rs = pstmt.executeQuery();
```

```
                        if (rs.next()) {
                                return makeMessageFromResultSet(rs);
                        } else {
                                return null;
                        }
                } finally {
                        JdbcUtil.close(rs);
                        JdbcUtil.close(pstmt);
                }
        }

        private Message makeMessageFromResultSet(ResultSet rs) throws SQLException {
                Message message = new Message();
                message.setId(rs.getInt("message_id"));
                message.setGuestName(rs.getString("guest_name"));
                message.setPassword(rs.getString("password"));
                message.setMessage(rs.getString("message"));
                return message;
        }

        public int selectCount(Connection conn) throws SQLException {
                Statement stmt = null;
                ResultSet rs = null;
                try {
                        stmt = conn.createStatement();
                        rs = stmt.executeQuery("select count(*) from guestbook_message");
                        rs.next();
                        return rs.getInt(1);
                } finally {
                        JdbcUtil.close(rs);
                        JdbcUtil.close(stmt);
                }
        }

        public List<Message> selectList(Connection conn, int firstRow, int endRow)
                        throws SQLException {
                PreparedStatement pstmt = null;
                ResultSet rs = null;
                try {
                        pstmt = conn.prepareStatement(
                                        "select * from guestbook_message " +
                                        "order by message_id desc limit ?, ?");
                        pstmt.setInt(1, firstRow - 1);
                        pstmt.setInt(2, endRow - firstRow + 1);
                        rs = pstmt.executeQuery();
                        if (rs.next()) {
                                List<Message> messageList = new ArrayList<Message>();
                                do {
                                        messageList.add(makeMessageFromResultSet(rs));
                                } while (rs.next());
                                return messageList;
                        } else {
                                return Collections.emptyList();
                        }
                } finally {
                        JdbcUtil.close(rs);
                        JdbcUtil.close(pstmt);
                }
        }

        public int delete(Connection conn, int messageId) throws SQLException {
                PreparedStatement pstmt = null;
                try {
                        pstmt = conn.prepareStatement(
                                        "delete from guestbook_message where message_id = ?");
```

```
                                pstmt.setInt(1, messageId);
                                return pstmt.executeUpdate();
                        } finally {
                                JdbcUtil.close(pstmt);
                        }
                }

}
```

15.2.7 서비스 클래스의 구현

(1) GetMessageListService 클래스의 구현

■ GetMessageListService는 요청한 페이지 번호에 해당하는 메시지 목록을 제공한다. 다음 순서로 필요한 작업을 실행한다.
1. 전체 메시지 개수를 구한다. MessageDao의 selectCount() 메서드를 사용한다.
2. 요청한 페이지 번호에 해당하는 메시지의 시작 행과 끝 행을 구한다.
3. 시작 행과 끝 행에 포함된 메시지 목록을 구한다.
4. MessageViewList 객체를 리턴한다.

[chap15/src/guestbook/service/GetMessageListService.java]

```
package guestbook.service;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.Collections;
import java.util.List;

import guestbook.dao.MessageDao;
import guestbook.model.Message;
import jdbc.JdbcUtil;
import jdbc.connection.ConnectionProvider;

public class GetMessageListService {
          private static GetMessageListService instance = new GetMessageListService();

          public static GetMessageListService getInstance() {
                    return instance;
          }

          private GetMessageListService() {
          }

          private static final int MESSAGE_COUNT_PER_PAGE = 3;

          public MessageListView getMessageList(int pageNumber) {
                    Connection conn = null;
                    int currentPageNumber = pageNumber;
                    try {
                              conn = ConnectionProvider.getConnection();
                              MessageDao messageDao = MessageDao.getInstance();

                              int messageTotalCount = messageDao.selectCount(conn);

                              List<Message> messageList = null;
                              int firstRow = 0;
                              int endRow = 0;
                              if (messageTotalCount > 0) {
                                        firstRow =
```

```
                                                (pageNumber - 1) * MESSAGE_COUNT_PER_PAGE + 1;
                                        endRow = firstRow + MESSAGE_COUNT_PER_PAGE - 1;
                                        messageList =
                                                messageDao.selectList(conn, firstRow, endRow);
                                } else {
                                        currentPageNumber = 0;
                                        messageList = Collections.emptyList();
                                }
                                return new MessageListView(messageList,
                                                messageTotalCount, currentPageNumber,
                                                MESSAGE_COUNT_PER_PAGE, firstRow, endRow);
                        } catch (SQLException e) {
                                throw new ServiceException("목록 구하기 실패: " + e.getMessage(), e);
                        } finally {
                                JdbcUtil.close(conn);
                        }
                }
        }
```

- MessageListView 클래스는 요청한 페이지 번호, 요청한 페이지의 메시지 목록, 전체 메시지 개수, 페이지 개수, 페이지 당 보여줄 메시지 개수 등의 정보를 제공한다.

[chap15/src/guestbook/service/MessageListView.java]

```
package guestbook.service;

import java.util.List;

import guestbook.model.Message;

public class MessageListView {

        private int messageTotalCount;
        private int currentPageNumber;
        private List<Message> messageList;
        private int pageTotalCount;
        private int messageCountPerPage;
        private int firstRow;
        private int endRow;

        public MessageListView(List<Message> messageList, int messageTotalCount,
                        int currentPageNumber, int messageCountPerPage,
                        int startRow, int endRow) {
                this.messageList = messageList;
                this.messageTotalCount = messageTotalCount;
                this.currentPageNumber = currentPageNumber;
                this.messageCountPerPage = messageCountPerPage;
                this.firstRow = startRow;
                this.endRow = endRow;

                calculatePageTotalCount();
        }

        private void calculatePageTotalCount() {
                if (messageTotalCount == 0) {
                        pageTotalCount = 0;
                } else {
                        pageTotalCount = messageTotalCount / messageCountPerPage;
                        if (messageTotalCount % messageCountPerPage > 0) {
                                pageTotalCount++;
                        }
                }
        }
```

```
	public int getMessageTotalCount() {
		return messageTotalCount;
	}

	public int getCurrentPageNumber() {
		return currentPageNumber;
	}

	public List<Message> getMessageList() {
		return messageList;
	}

	public int getPageTotalCount() {
		return pageTotalCount;
	}

	public int getMessageCountPerPage() {
		return messageCountPerPage;
	}

	public int getFirstRow() {
		return firstRow;
	}

	public int getEndRow() {
		return endRow;
	}

	public boolean isEmpty() {
		return messageTotalCount == 0;
	}
}
```

(2) WriteMessageService 클래스의 구현

■ 별도 로직을 수행하지 않고 MessageDao의 insert() 메서드를 실행한다.

[chap15/src/guestbook/service/WriteMessageService.java]

```
package guestbook.service;

import java.sql.Connection;
import java.sql.SQLException;

import guestbook.dao.MessageDao;
import guestbook.model.Message;
import jdbc.JdbcUtil;
import jdbc.connection.ConnectionProvider;

public class WriteMessageService {
	private static WriteMessageService instance = new WriteMessageService();

	public static WriteMessageService getInstance() {
		return instance;
	}

	private WriteMessageService() {
	}

	public void write(Message message) {
		Connection conn = null;
```

```
                    try {
                              conn = ConnectionProvider.getConnection();
                              MessageDao messageDao = MessageDao.getInstance();
                              messageDao.insert(conn, message);
                    } catch (SQLException e) {
                              throw new ServiceException(
                                        "메시지 등록 실패: " + e.getMessage(), e);
                    } finally {
                              JdbcUtil.close(conn);
                    }
          }

}
```

(3) DeleteMessageService 클래스의 구현

■ 메시지 삭제 기능을 제공하는 DeleteMessageService 클래스는 다음과 같은 순서로 기능을 실행한다.

1. 지정한 번호에 해당하는 메시지를 검색한다.
2. 메시지가 존재하지 않으면 익셉션을 발생한다.
3. 메시지에 암호가 지정되어 있지 않으면 익셉션을 발생한다.
4. 메시지의 메시지와 파라미터로 전달받은 암호가 다르면 익셉션을 발생한다.
5. 메시지를 삭제한다. MessageDao.delete() 메서드를 사용한다.

[chap15/src/guestbook/service/DeleteMessageService.java]

```
package guestbook.service;

import java.sql.Connection;
import java.sql.SQLException;

import guestbook.dao.MessageDao;
import guestbook.model.Message;
import jdbc.JdbcUtil;
import jdbc.connection.ConnectionProvider;

public class DeleteMessageService {

          private static DeleteMessageService instance = new DeleteMessageService();

          public static DeleteMessageService getInstance() {
                    return instance;
          }

          private DeleteMessageService() {
          }

          public void deleteMessage(int messageId, String password) {
                    Connection conn = null;
                    try {
                              conn = ConnectionProvider.getConnection();
                              conn.setAutoCommit(false); // 트랜잭션을 시작한다.

                              MessageDao messageDao = MessageDao.getInstance();
                              Message message = messageDao.select(conn, messageId); // 삭제할 메시지에 해당하
는 Message 객체를 구한다.
                              if (message == null) {
                                        throw new MessageNotFoundException("메시지 없음");
                              }
                              if (!message.matchPassword(password)) {
```

```
                                        throw new InvalidPassowrdException("bad password");
                                }
                                messageDao.delete(conn, messageId); // 지정한 ID에 해당하는 메시지를 삭제.

                                conn.commit(); // 트랜잭션을 커밋한다.
                        } catch (SQLException ex) {
                                JdbcUtil.rollback(conn); // 트랜잭션을 롤백한다.
                                throw new ServiceException("삭제 실패:" + ex.getMessage(), ex);
                        } catch (InvalidPassowrdException | MessageNotFoundException ex) {
                                JdbcUtil.rollback(conn);
                                throw ex;
                        } finally {
                                JdbcUtil.close(conn);
                        }
                }
}
```

15.2.8 JSP에서 서비스 사용하기

(1) 메시지 목록을 보여주는 list.jsp

[chap15/WebContent/list.jsp]

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ page import="guestbook.model.Message"%>
<%@ page import="guestbook.service.MessageListView"%>
<%@ page import="guestbook.service.GetMessageListService"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
        String pageNumberStr = request.getParameter("page");
        int pageNumber = 1;
        if (pageNumberStr != null) {
                pageNumber = Integer.parseInt(pageNumberStr);
        }

        GetMessageListService messageListService =
                        GetMessageListService.getInstance();
        MessageListView viewData =
                        messageListService.getMessageList(pageNumber);
%>
<c:set var="viewData" value="<%= viewData %>"/>
<html>
<head>
        <title>방명록 메시지 목록</title>
</head>
<body>

<form action="writeMessage.jsp" method="post">
이름: <input type="text" name="guestName"> <br>
암호: <input type="password" name="password"> <br>
메시지: <textarea name="message" cols="30" rows="3"></textarea> <br>
<input type="submit" value="메시지 남기기" />
</form>
<hr>
<c:if test="${viewData.isEmpty()}">
등록된 메시지가 없습니다.
</c:if>

<c:if test="${!viewData.isEmpty()}">
<table border="1">
        <c:forEach var="message" items="${viewData.messageList}">
        <tr>
```

```
                    <td>
                    메시지 번호: ${message.id} <br/>
                    손님 이름: ${message.guestName} <br/>
                    메시지: ${message.message} <br/>
                    <a href="confirmDeletion.jsp?messageId=${message.id} ">[삭제하기]</a>
                    </td>
            </tr>
            </c:forEach>
</table>

<c:forEach var="pageNum" begin="1" end="${viewData.pageTotalCount} ">
<a href="list.jsp?page=${pageNum} ">[${pageNum}]</a>
</c:forEach>

</c:if>
</body>
</html>
```

(2) 메시지 등록을 처리하는 writeMessage.jsp

■ 전달받은 요청 파라미터를 이용해서 Message 객체를 생성한 뒤, WriteMessageService를 이용해서 메시지를 기록한다.

[chap15/WebContent/writeMessage.jsp]

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ page errorPage="errorView.jsp" %>
<%@ page import="guestbook.model.Message" %>
<%@ page import="guestbook.service.WriteMessageService" %>
<%
            request.setCharacterEncoding("utf-8");
%>
<jsp:useBean id="message" class="guestbook.model.Message">
            <jsp:setProperty name="message" property="*" />
</jsp:useBean>
<%
            WriteMessageService writeService = WriteMessageService.getInstance();
            writeService.write(message);
%>
<html>
<head>
            <title>방명록 메시지 남김</title>
</head>
<body>
방명록에 메시지를 남겼습니다.
<br/>
<a href="list.jsp">[목록 보기]</a>
</body>
</html>
```

(3) 메시지 삭제 폼을 제공하는 confirmDeletion.jsp

[chap15/WebContent/confirmDeletion.jsp]

```
<%@ page contentType="text/html; charset=utf-8" %>
<html>
<head>
    <title>방명록 메시지 삭제 확인</title>
</head>
```

```
<body>

<form action="deleteMessage.jsp" method="post">
<input type="hidden" name="messageId" value="${param.messageId}">
메시지를 삭제하시려면 암호를 입력하세요:<br>
암호: <input type="password" name="password"> <br>
<input type="submit" value="메시지 삭제하기">
</form>
</body>
</html>
```

(4) 메시지 삭제 요청을 처리하는 deleteMessage.jsp

[chap15/WebContent/deleteMessage.jsp]

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ page import="guestbook.service.DeleteMessageService" %>
<%@ page import="guestbook.service.InvalidPassowrdException" %>
<%
	int messageId = Integer.parseInt(request.getParameter("messageId"));
	String password = request.getParameter("password");
	boolean invalidPassowrd = false;
	try {
		DeleteMessageService deleteService =
				DeleteMessageService.getInstance();
		deleteService.deleteMessage(messageId, password);
	} catch(InvalidPassowrdException ex) {
		invalidPassowrd = true;
	}
%>
<html>
<head>
	<title>방명록 메시지 삭제함</title>
</head>
<body>
<%  if (!invalidPassowrd) { %>
메시지를 삭제하였습니다.
<%  } else { %>
입력한 암호가 올바르지 않습니다. 암호를 확인해주세요.
<%  }%>
<br/>
<a href="list.jsp">[목록 보기]</a>
</body>
</html>
```