# JSP 프로그래밍

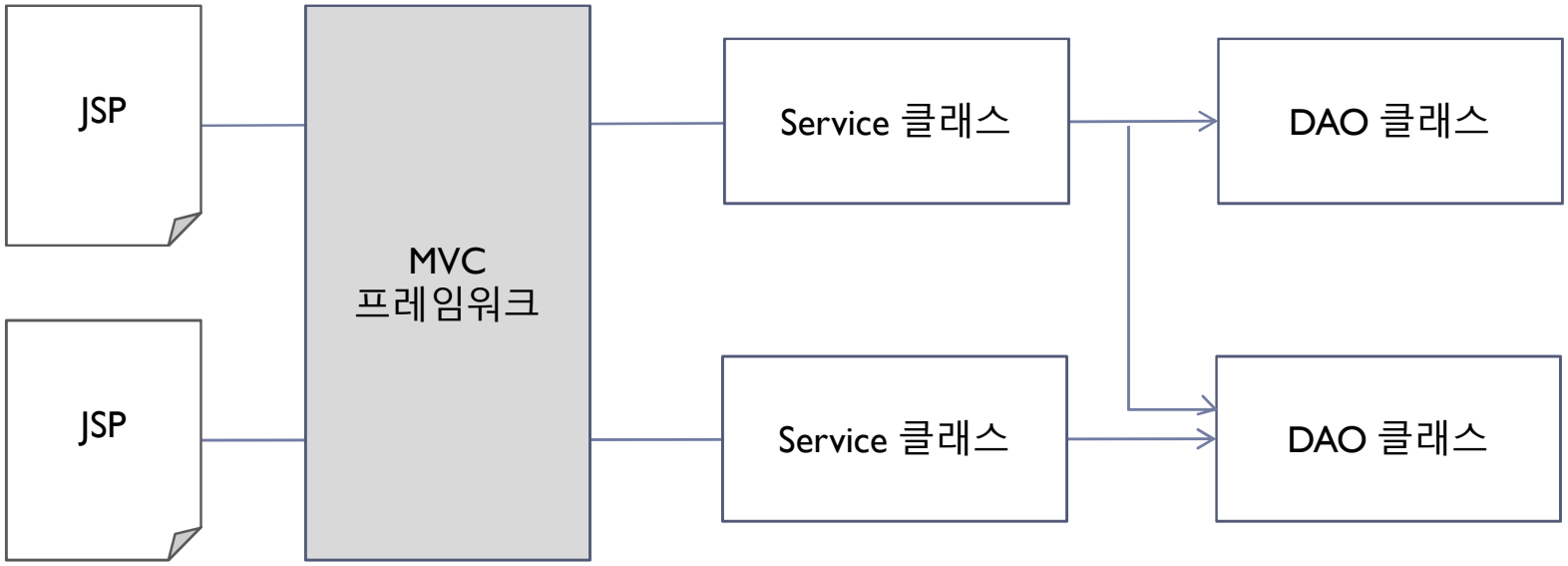
(재)대덕인재개발원

# 13 웹 어플리케이션의 일반적인 구성 및 방명록 구현

- 1. 어플리케이션의 전형적인 구성 요소
- 2. 방명록 구현

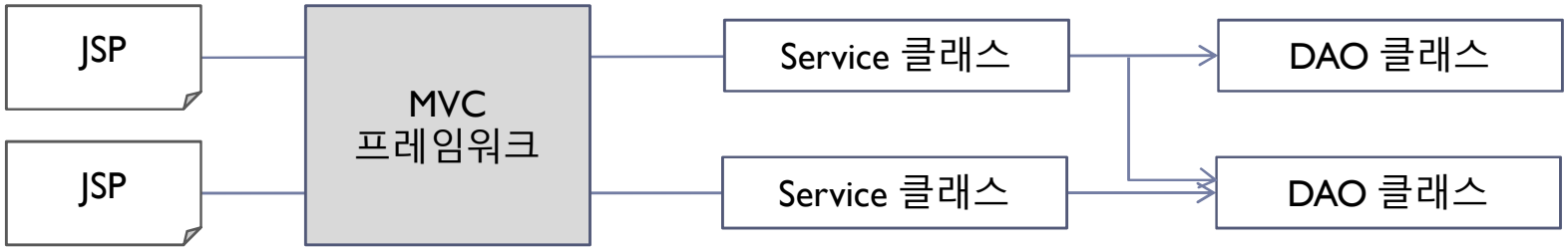
# 13.1.1 어플리케이션의 전형적인 구성 요소

- 웹 어플리케이션의 주요 구성 요소




# 13.1.1 어플리케이션의 전형적인 구성 요소

- 웹 어플리케이션의 주요 구성 요소




| 요 소 | 역 할 |
|---|---|
| JSP(뷰) | Service 클래스가 실행한 결과를 화면에 출력해 주거나 Service가 수행하는 데 필요한 데이터를 전달한다. |
| MVC 프레임워크 | 사용자의 요청을 Service에 전달하고 Service의 실행 결과를 JSP와 같은 뷰에 전달한다. 스프링 MVC나 스트럿츠와 같은 프레임워크가 MVC 프레임워크에 해당된다. |
| Service 클래스 | 사용자의 요청을 처리하는 기능을 제공한다. 기능을 제공하기 위한 로직을 구현하고 있으며 DAO 클래스를 통해서 DB 연동을 처리한다. 가입신청처리, 글 목록 제공 등의 기능을 구현한다. |
| DAO 클래스 | DB과 관련된 CRUD 작업을 처리한다. Service 클래스들은 데이터를 DB에서 읽어오거나 DB에 데이터를 저장할 때 DAO 클래스를 사용한다. |

# 13.1.2 데이터 접근 객체(Data Access Object)의 구현

- 정의
  - DAO 클래스는 데이터에 접근할 때 사용되는 클래스.
  - INSERT, SELECT, UPDATE, DELETE 쿼리를 실행해 주는 메서드를 제공한다.

- 1. DAO에서 Connection 에 접근하는 방식
  - DAO 클래스의 메서드에서 직접 Connection을 생성
  - DAO 객체를 생성할 때 생성자에서 Connection을 전달 받기
  - DAO 클래스의 메서드 패러미터로 Connection을 전달 받기

# 13.1.3 DAO 객체를 제공하는 DaoProvider

- 각 기능별 DAO 객체를 제공해 주는 기능을 수행하는 클래스.(교재와 다름)

```
public class DaoProvider {

   private static Map<String, Object> daoObjectMap = new HashMap();
   static{
      ResourceBundle bundle = ResourceBundle.getBundle("dao");
      Enumeration<String> enumKeys = bundle.getKeys();
      while(enumKeys.hasMoreElements()){
         String key = enumKeys.nextElement();
         String className = bundle.getString(key);
         Class daoClass = null;
         Object instance = null;

         try{
            daoClass = Class.forName(className);
            try{
               daoClass.newInstance();
            }catch (InstantiationException e){
            }catch (IllegalAccessException e){
                Method method = daoClass.getMethod("getInstance", null);
                instance = method.invoke(daoClass, null);
                daoObjectMap.put(key, instance);
                System.out.println(key + " = " + instance.getClass());
             }
         }catch(ClassNotFoundException e){
         }catch(Exception e){
         }
      }
   }
```

# 13.1.3 DAO 객체를 제공하는 DaoProvider

- 각 기능별 DAO 객체를 제공해 주는 기능을 수행하는 클래스.(교재와 다름)

```
// Singleton 방식으로 객체 생성.
  private static DaoProvider instance = new DaoProvider();

  private DaoProvider(){}

  public static DaoProvider getInstance(){
     if(instance == null){
       instance = new DaoProvider();
     }
     return instance;
  }

  public static Object getDao(Class interfaceClass){
     String interfaceName = interfaceClass.getName();

     if(!daoObjectMap.containsKey(interfaceName)){
       throw new RuntimeException("Class that implements " + interfaceName + " is not Exist...");
     }
     return daoObjectMap.get(interfaceName);
  }
}
```

## 13.1.4 서비스 클래스의 구현

- DAO가 데이터에 접근할 때 사용되는 기능을 제공한다면 서비스 클래스는 사용자의 요청을 처리하기 위한 기능을 제공한다.
- 서비스 클래스는 주로 DAO를 통해서 데이터에 접근하고 기능을 수행하는 데 필요한 로직을 수행한다.

# 13.1.4 서비스 클래스의 구현

- 1. 서비스 클래스와 트랜잭션 처리
  - 서비스 클래스가 제공하는 메서드에서 실행하는 코드가 하나의 트랜잭션 범위에서 처리되어야 한다면, 다음과 같이 메서드에서 로직을 수행하기 전에 Connection.setAutoCommit(false)로 하고 트랜잭션을 시작하고 모든 코드가 종료되면 Connection.commit()을 실행하고, 로직 수행 도중 예외 발생시 트랜잭션을 롤백해서 잘못된 데이터가 DB에 반영되지 않도록 한다.

```
try{
    conn = DriverManager.getConnection(….);          // Connection을 구한다.
    conn.setAutoCommit(false);                       // 로직 수행하기 전에 트랜잭션 시작
    ……..
    conn.commit();                                   // 로직을 수행한 뒤 트랜잭션 커밋

}catch(SQLException e){
    if(conn != null){
        try{
            conn.rollback();                 // 로직 실행 도중 예외가 발생하면 트랜잭션 롤백
        }catch(SQLException ex){}
    }
    ……
}finally{
     ……
}
```

# 13.1.4 서비스 클래스의 구현

## 2. 서비스 클래스의 예외 처리

- 서비스 클래스는 내부적으로 데이터베이스 처리 실패와 같은 예외가 발생한 경우, 서비스에 알맞은 예외를 생성해서 발생시켜 주는 것이 좋다.
- 서비스 클래스는 내부적으로 발생한 예외뿐만 아니라 논리적으로 잘못된 경우에도 예외를 발생시켜야 한다.

```
public void deleteMessage(int messageID) throws Exception{
    Connection conn = null;
    try{
        conn = ...;        // Connection을 구한다.
        conn.setAutoCommit(false);
        ...
        MessageDao dao = MessageDao.getInstance();
        Message message = dao.selectById(messageID);
        if(message == null){
            // 논리적인 잘못된 경우에도 알맞은 예외를 발생시킨다.
            throw new MessageNotFoundException(messageID);
        }
         dao.deleteById(messageID);
    }catch(SQLException e){
        conn.rollback();
        // 내부적으로 발생한 예외에 대해 알맞은 예외를 발생시킨다.
        throw new FailedDeleteException("삭제에 실패했습니다.", e);
    }finally{
        conn.close();
    }
}
```

# 13.1.5 싱글톤(Singleton) 패턴을 이용한 구성 요소 구현

- 객체를 여러 번 사용하는 것과 매번 새로운 객체를 생성해서 사용하는 것과의 기능상 차이가 없다면, 매번 새로운 서비스 객체를 생성하지 않고 한 개의 객체를 재사용 하도록 구현해도 된다.
- 이럴 때 적용 가능한 것이 싱글톤 패턴이다.
- 싱글톤 패턴은 특정 클래스의 객체가 단 한 개만 존재하도록 제약하는 구현 패턴이다.

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

    // 생성자를 private 으로 설정해서 외부에서 접근하지 못하게 함
    private MemberService(){}

    // 유일한 객체에 접근할  수 있는 정적 메서드 정의
    public static MemberService getInstance(){
        if(instance == null){
            instance = new MemberService();
        }
        return instance;
    }
     .....
}
```

# 13.1.6 Connection을 제공해 주는 ConnectionProvider

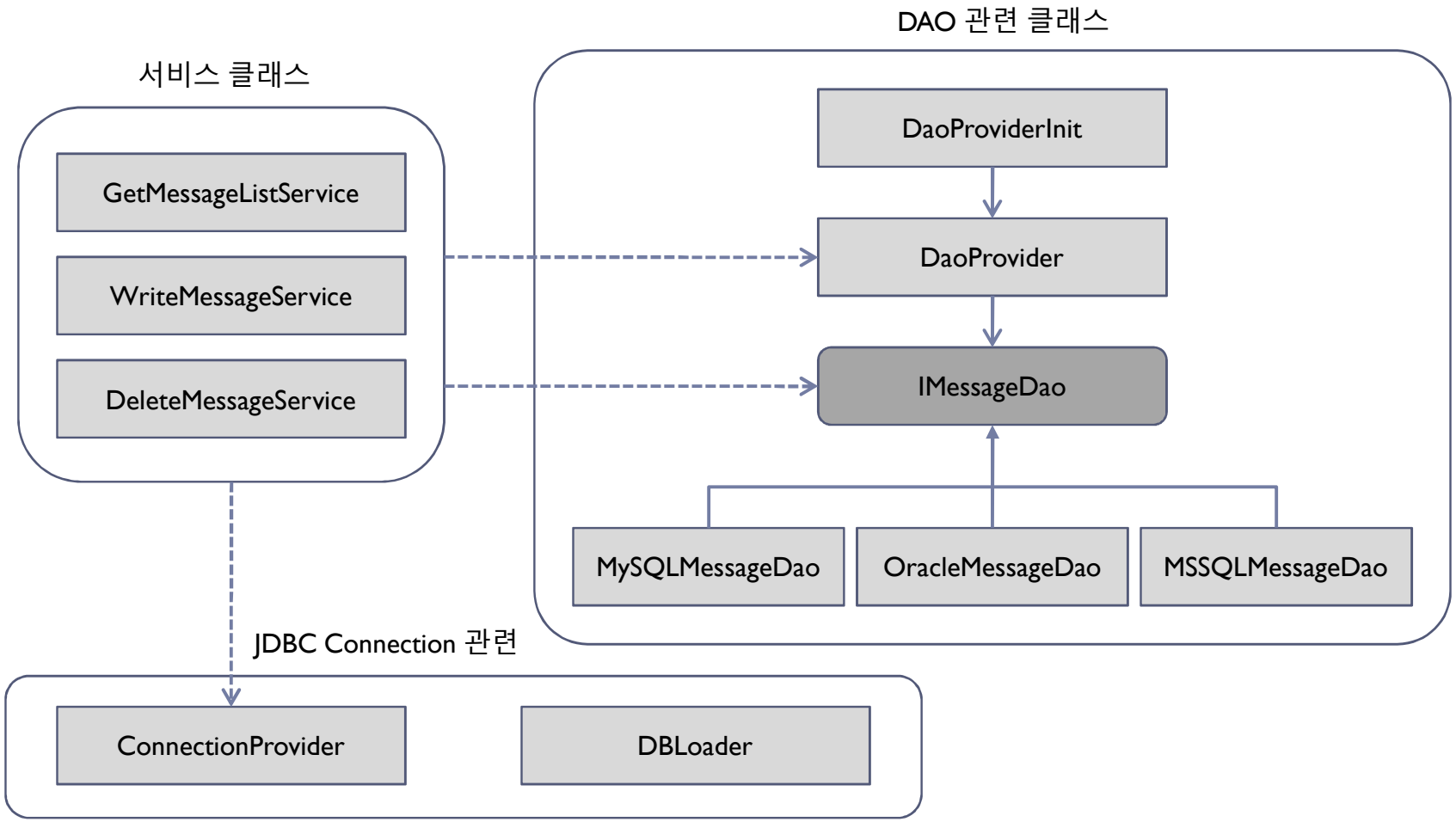
- DB 연동에 필요한 모든 코드에서 DriverManager.getConnection() 메서드를 사용해서 Connection 객체를 구하기 보다는 Connection을 제공해 주는 기능을 별도의 클래스로 분리해 주는 것이 개발이나 유지 보수 하는데 장점을 갖는다.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionProvider {
    public static Connection getConnection() throws SQLException{
        String url = "jdbc:apache:commons:dbcp:/pool";
        return DriverManager.getConnection(url);
    }
}
```

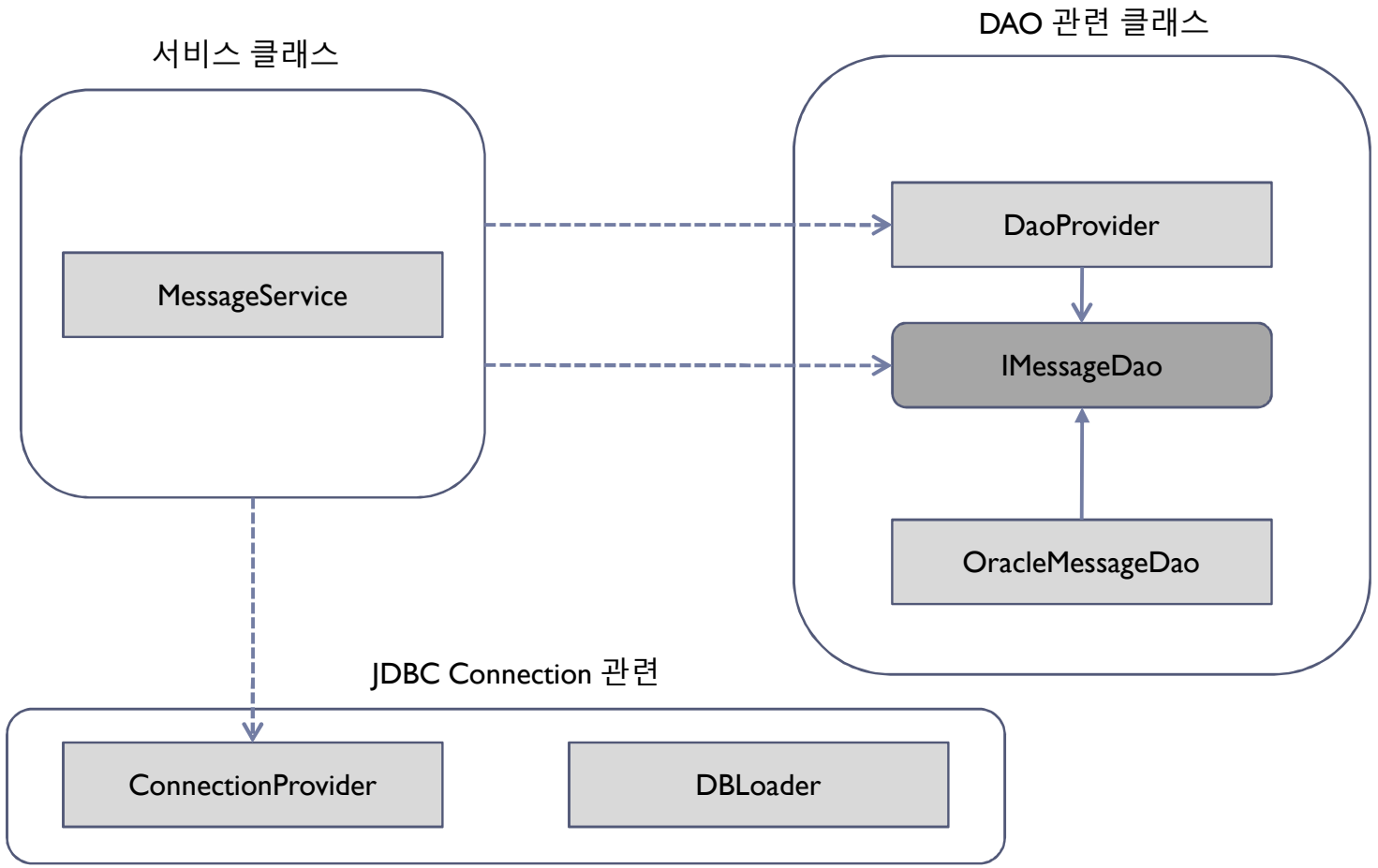
```
// 컨넥션이 필요한 곳에서는 ConnectionProvider를 통해 컨넥션을 얻을 수 있다.
Connection conn = null;
conn = ConnectionProvider.getConnection();
```

# 13.2 방명록 구현(교재)



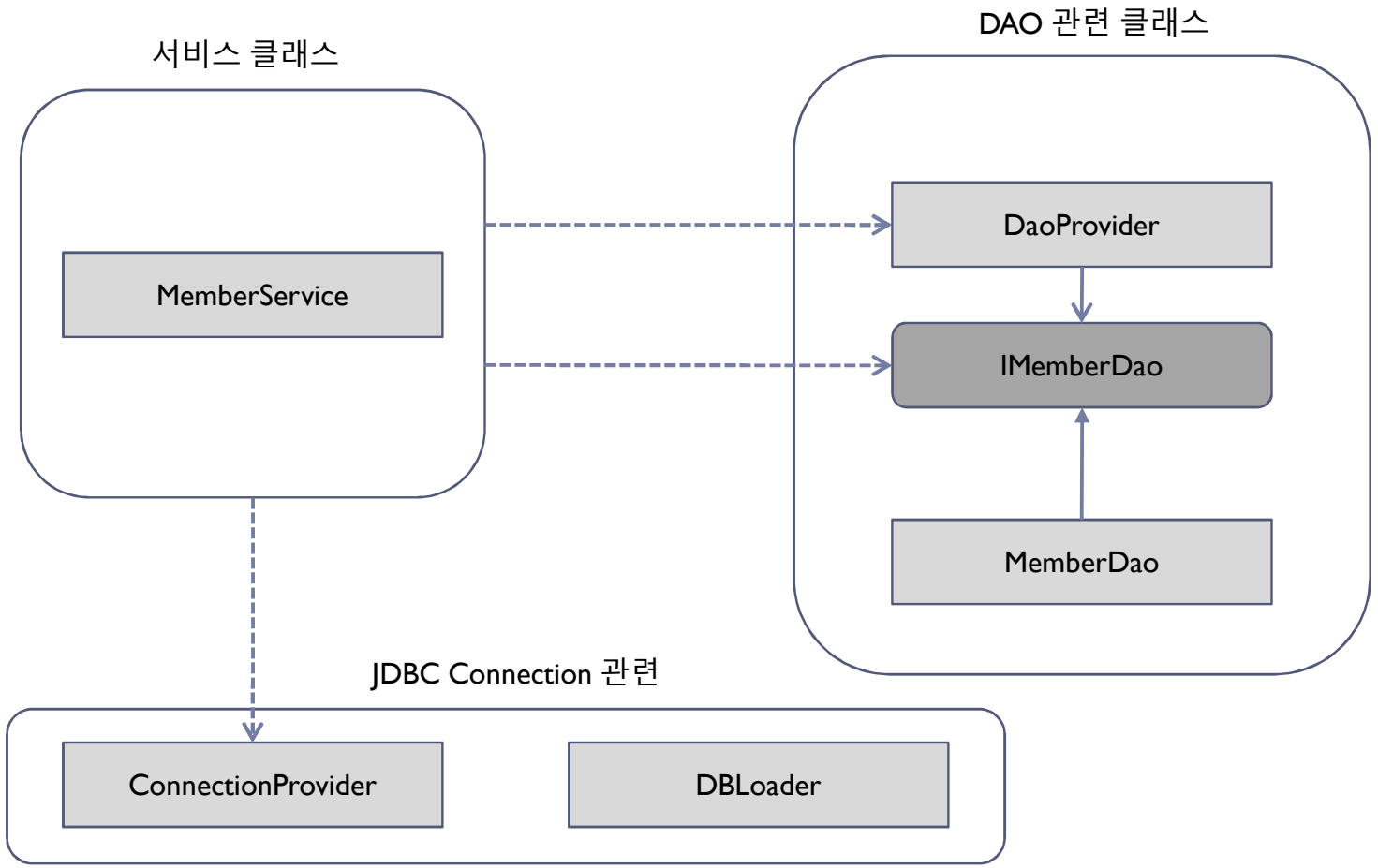

**[그림] 방명록 구현에 관련된 클래스 구조**

# 13.2 방명록 구현(간략)




**[그림] 방명록 구현에 관련된 클래스 구조**

# 13.2 회원관리 구현(간략)




**[그림] 회원관리 구현에 관련된 클래스 구조**

# 13.2 회원관리 - JSP 페이지에서 클래스 추출

```
// 1. 드라이버 로딩
Class.forName("oracle.jdbc.driver.OracleDriver");
```

JDBC Driver 로딩을 위한 영역

```
// 2. 연결객체 얻기
conn = DriverManager.getConnection(
"jdbc:oracle:thin:@127.0.0.1:1521:xe","sem","java");
```

DB에 접속하기 위한 Connectiion 객체를 얻어 오는 영역

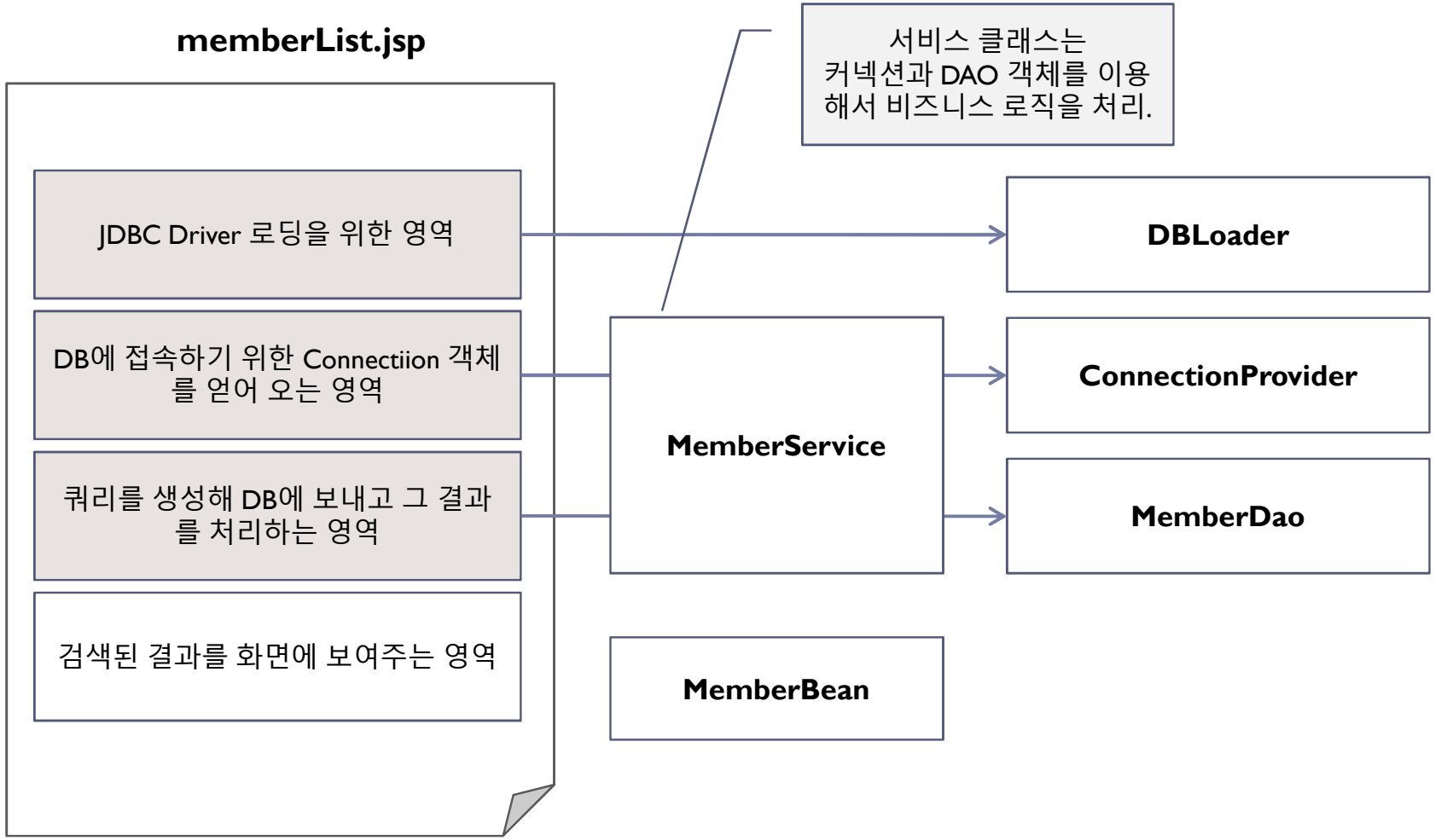
```
// 3. 구문객체 얻기
stmt = conn.createStatement();
// 4. 구문실행, 결과 얻기
rs = stmt.executeQuery("Select * From member");
```

쿼리를 생성해 DB에 보내고 그 결과를 처리하는 영역

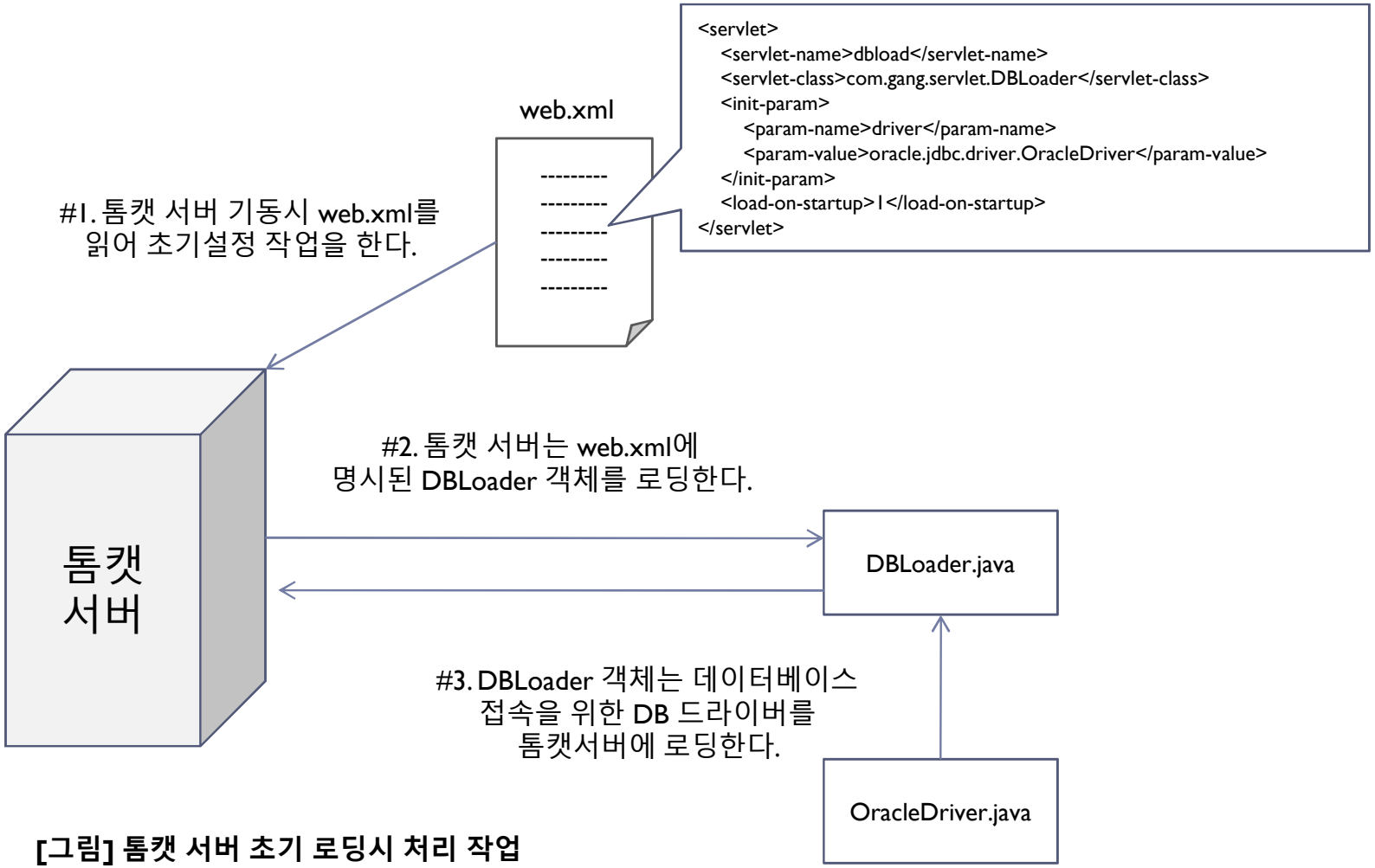
```
while(rs.next()){
%>
<tr>
<td><%=rs.getString(1)%></td>
<td><a href="mem_view.jsp?mem_id=<%=rs.getString("mem_id")%>" >
<%=rs.getString("mem_name")%> </a>
</td>
<td><%=rs.getString("mem_hp")%></td>
<td>(<%=rs.getString("mem_zip")%>)
<%=rs.getString("mem_add1")%>
<%=rs.getString("mem_add2")%>
</td>
<td><%=rs.getString("mem_mileage")%></td>
</tr>
<%
}
```

검색된 결과를 화면에 보여주는 영역

# 13.2 회원관리 - JSP 페이지에서 클래스 추출

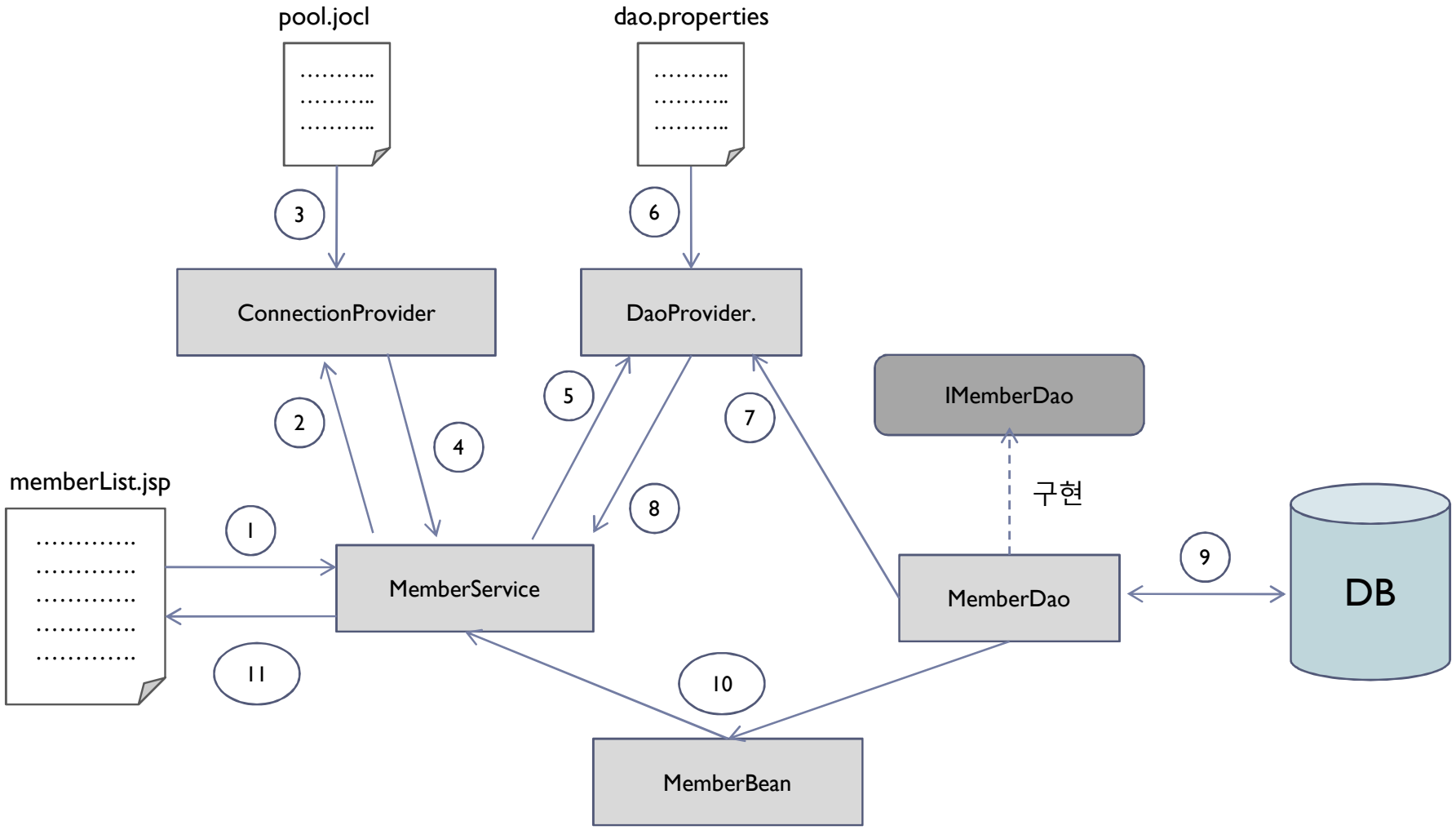


# 13.2 회원관리 구현 – 서버 초기 로딩시




**[그림] 톰캣 서버 초기 로딩시 처리 작업**

# 13.2 회원관리 구현 – 프로세스 흐름



# 13.2 회원관리 구현 프로세스 흐름 상세

| 단계 | 설명 |
|---|---|
| 1 | 클라이언트로 부터 요청을 받은 mem_list.jsp는 MemberService 객체를 생성해 멤버 리스트 정보를 요청한다. |
| 2 | MemberService 객체는 DB접근을 위해 ConnectionProvider 객체에게 Connection 객체를 요구한다. |
| 3 | ConnectionProvider 객체는 pool.jocl 설정 파일에서 Connection Pool관련 정보를 읽어들인다. |
| 4 | ConnectionProvider 객체는 pool.jocl 설정 파일에 의거해 Connection Pool에서 Connection 객체 하나를 MemberService 객체에게 반환한다. |
| 5 | MemberService 객체는 Dao 객체를 생성하기 위해 DaoProvider 객체에게 Dao객체를 요구한다. ( DaoProvider.getDao( ...) ) |
| 6 | DaoProvider 는 dao.properties 파일에서 Dao 인터페이스 Dao 구현객체의 맵핑 정보를 읽어 들인다. |
| 7 | DaoProvider 는 dao.properties 파일에 맵핑 된 Dao객체를 미리 Map객체(objectMap)에 Key와 Value의 쌍으로 저장해 둔다. |
| 8 | MemberService 객체가 요청한 Dao객체를 DaoProvider는 제공한다. |
| 9 | MemberService 객체는 MemberDao 객체를 이용해서 DB에 쿼리를 수행한다. |
| 10 | MemberDao 객체는 처리된 쿼리 결과를 MemberVO 객체에 담아 MemberService 객체에게 반환한다. |
| 11 | MemberService 객체는 MemberDao로 부터 반환된 MemberVO객체를 mem_list.jsp에게 반환한다. |