2020년도 1학기

자바실무프로젝트 과목 – 11주차

[교재 Day02-Class06 스프링JDBC] p209~

스프링 실습 프로젝트 정리 실습

- 1주차 ~ 4주차 : //스프링을 위한 선수 지식 : Servlet 요점 정리
- SpringProject1: (5주차) //개발자에 의한 객체 결합 (교재 p39~48)
- SpringProject2: (6주차) //스프링컨테이너에 의한 객체 결합 (XML 방식) - IoC- (교재 p49~91)
- SpringProject3: (7주차) //스프링컨테이너에 의한 객체 결합 (어노테이션 방식) - IoC- (교재 p93~108)
- SpringProject4_BizStep1: (8주차) //비즈니스 컴포넌트 실습 1 (게시판) -Business Layer- (교재 p109~127)
- SpringProject4_BizStep2: (9주차) // 비즈니스 컴포넌트 실습 2 (User추가) -Business Layer- (교재 p129~139)
- SpringProject5_AOP_step1: (9주차) // 스프링 AOP 개념 파악 - AOP - (교재 p143~170)
- SpringProject5_AOP_step2: (10주차) // AOP 동작 시점 (5개) 실습- AOP - (교재 p171~182)
- SpringProject5_AOP_step3: (10주차) // AOP 동작 시점과 JointPoint/바인드변수 실습- AOP - (교재 p183~192)
- SpringProject5_AOP_step4: (10주차) // 어노테이션 방식의 AOP 실습- AOP - (교재 p193~192)
- SpringProject6_Jdbc_step1: (11주차) // Spring JDBC -스프링 JDBC - (교재 p209~226)
- SpringProject6_Jdbc_step2: (11주차) // 트랜잭션 처리-트랜잭션 - (교재 p227~237)
- SpringProject7_MVC_step1 : (11주차) // MVC 패턴 - Model 1 아키텍처 - (교재 p241~261)
- SpringProject7_MVC_step2 : (11주차) // MVC 패턴 - Model 2 아키텍처 - (교재 p263~283)
- SpringProject7_MVC_step3 : (11주차) // MVC 패턴 - MVC 프레임워크 아키텍처 - (교재 p285~311)

9주 과제

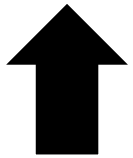
10주 과제

11주 과제

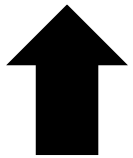
SpringProject6_Jdbc_step1 프로젝트

Mybatis 개발자가 지정한 SQL, 저장프로시저, 고급 매핑을 지원하는 Framework

[교재] p467



스프링 JDBC



JDBC

JDBC 프로그램의 문제

1. 정해진 순서대로 프로그램을 구현해야 한다.
2. 모든 DB 연동 메소드에서 동일한 로직이 반복적으로 등장한다.
3. 많은 코드로 인해 실수할 가능성이 높다.
4. 결과적으로 유지보수가 어렵다.

Spring JDBC

Spring JDBC - 개발자가 해야 할 일은?

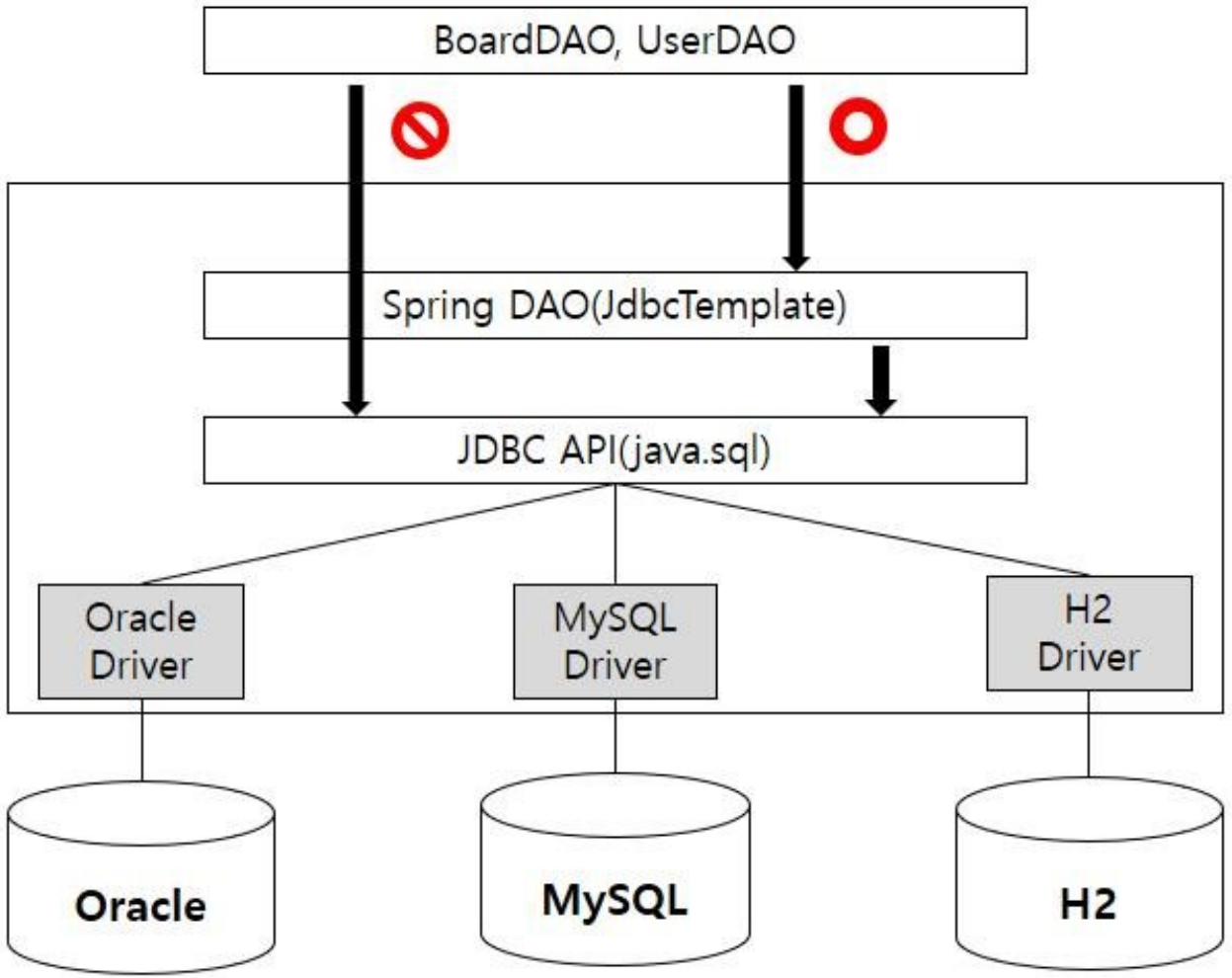
동작	스프링	어플리케이션 개발자
연결 파라미터 정의.		0
연결 오픈.	0	
SQL 문 지정.		0
파라미터 선언과 파라미터 값 제공		0
스테이트먼트(statement) 준비와 실행.	0	
(존재한다면)결과를 반복하는 루프 설정	0	
각 이터레이션에 대한 작업 수행.		0
모든 예외 처리.	0	
트랜잭션 제어.	0	
연결, 스테이트먼트(statement), 리절트셋(resultset) 닫기.	0	

```
private final String USER_GET = "select * from users where id=? and password=?";

public UserVO getUser(UserVO vo) {
    UserVO user = null;
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","book_ex","book_ex123");
        PreparedStatement stmt = conn.prepareStatement(USER_GET);
        stmt.setString(1, vo.getId());
        stmt.setString(2, vo.getPassword());
        rs = stmt.executeQuery();
        if (rs.next()) {
            user = new UserVO();
            user.setId(rs.getString("ID"));
            user.setPassword(rs.getString("PASSWORD"));
            user.setName(rs.getString("NAME"));
            user.setRole(rs.getString("ROLE"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        rs, stmt, conn 의 close( );
    }
    return user;
}
```

SpringProject6_Jdbc_step1 프로젝트

- Spring JDBC 구조



Spring JDBC 사용하려면 pom.xml 추가

```
<!-- DBCP -->
<dependency>
	<groupId>commons-dbcp</groupId>
	<artifactId>commons-dbcp</artifactId>
	<version>1.4</version>
</dependency>
```

Spring JDBC

SpringProject6_Jdbc_step1 프로젝트

Spring JDBC 패키지

org.springframework.jdbc.core

- JdbcTemplate 및 관련 Helper 객체 제공

org.springframework.jdbc.datasource

- DataSource를 쉽게 접근하기 위한 유틸 클래스, 트랜젝션매니져 및 다양한 DataSource 구현을 제공

org.springframework.jdbc.object

- RDBMS 조회, 갱신, 저장등을 안전하고 재사용 가능한 객제 제공

org.springframework.jdbc.support

- jdbc.core 및 jdbc.object를 사용하는 JDBC 프레임워크를 지원

JDBC Template

- org.springframework.jdbc.core에서 가장 중요한 클래스입니다.
- 리소스 생성, 해지를 처리해서 연결을 닫는 것을 잊어 발생하는 문제 등을 피할 수 있도록 합니다.
- 스테이먼트(Statement)의 생성과 실행을 처리합니다.
- SQL 조회, 업데이트, 저장 프로시저 호출, ResultSet 반복호출 등을 실행합니다.
- JDBC 예외가 발생할 경우 org.springframework.dao패키지에 정의되어 있는 일반적인 예외로 변환시킵니다.

JdbcTemplate 메소드

메소드	설 명
update()	INSERT, UPDATE, DELETE 명령어를 처리한다.
queryForInt()	정수 하나를 검색하여 리턴한다.
queryForObject()	검색 결과를 객체(Value Object) 하나에 매핑하여 리턴한다.
query()	검색 결과 여러 개를 java.util.List에 저장하여 리턴한다.

Spring JDBC - JdbcTemplate 메소드

SpringProject6_Jdbc_step1 프로젝트

- **update( ) 메소드 :** insert, update, delete 구문 처리

메소드	**int update(String sql, Object .. args)**

사용예	public void updateBoard(BoardVO vo){
	String BOARD_UPDATE = "update board set title=?, content=? where seq=?";
	int cnt = jdbcTemplate.update(BOARD_UPDATE, vo.getTitle( ), vo.getConttnet( ), vo.getSeq( ));
	System.out.println(cnt+ '건 데이터 수정');
	}

메소드	**int update(String sql, Object[ ] args)**

사용예	public void updateBoard(BoardVO vo){
	String BOARD_UPDATE = "update board set title=?, content=? where seq=?";
	Object[ ] args = { vo.getTitle( ), vo.getConttnet( ), vo.getSeq( ) }
	int cnt = jdbcTemplate.update(BOARD_UPDATE, args);
	System.out.println(cnt+ '건 데이터 수정');
	}

- **queryForInt( ) 메소드 :** select 구문으로 검색된 정수값 리턴

메소드	**int queryForInt(String sql)** **int queryForInt(String sql, Object .. args)** **int queryForInt(String sql, Object[ ] args)**

사용예	public int getBoardTotalCount(BoardVO vo){
	String BOARD_TOT_COUNT = "select count(*) from board";
	int cnt = jdbcTemplate.queryForInt(BOARD_TOT_COUNT);
	System.out.println("전체 게시글 수 : " + cnt+ '건');
	}

Spring JDBC - JdbcTemplate 메소드

SpringProject6_Jdbc_step1 프로젝트

- **queryForObject( ) 메소드 :** select 구문의 실행 결과를 VO 객체로 매핑하여 리턴

메소드	**int queryForObject(String sql)** **int queryForObject (String sql, RowMappter<T> rowMapper)** **int queryForObject (String sql, Object[ ] args, RowMappter<T> rowMapper)**

사용예	public BoardVO getBoard(BoardVO vo){
	String BOARD_GET = "select * from board where seq=?";
	Object[ ] args = { vo.getSeq( ) } ;
	return jdbcTemplate.queryForObject(BOARD_GET, args, new BoardRowMapper( )) ;
	}

- **query( ) 메소드 :** select 구문의 실행 결과가 목록일때

메소드	**int queryForObject(String sql)** **int queryForObject (String sql, RowMappter<T> rowMapper)** **int queryForObject (String sql, Object[ ] args, RowMappter<T> rowMapper)**

사용예	public BoardVO getBoard(BoardVO vo){
	String BOARD_LIST = "select * from board order by seq desc";
	return jdbcTemplate.queryForObject(BOARD_LIST, new BoardRowMapper( )) ;
	}

Spring JDBC – DataSource설정

SpringProject6_Jdbc_step1 프로젝트

- **DataSource 설정 (1)**

```
<bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
      <property name="driverClassName" value="  ... " />
      <property name="url" value="     " />
      <property name="username" value=" " />
      <property name="password" value=""  />
</bean>
```

- **DataSource 설정 (2)**

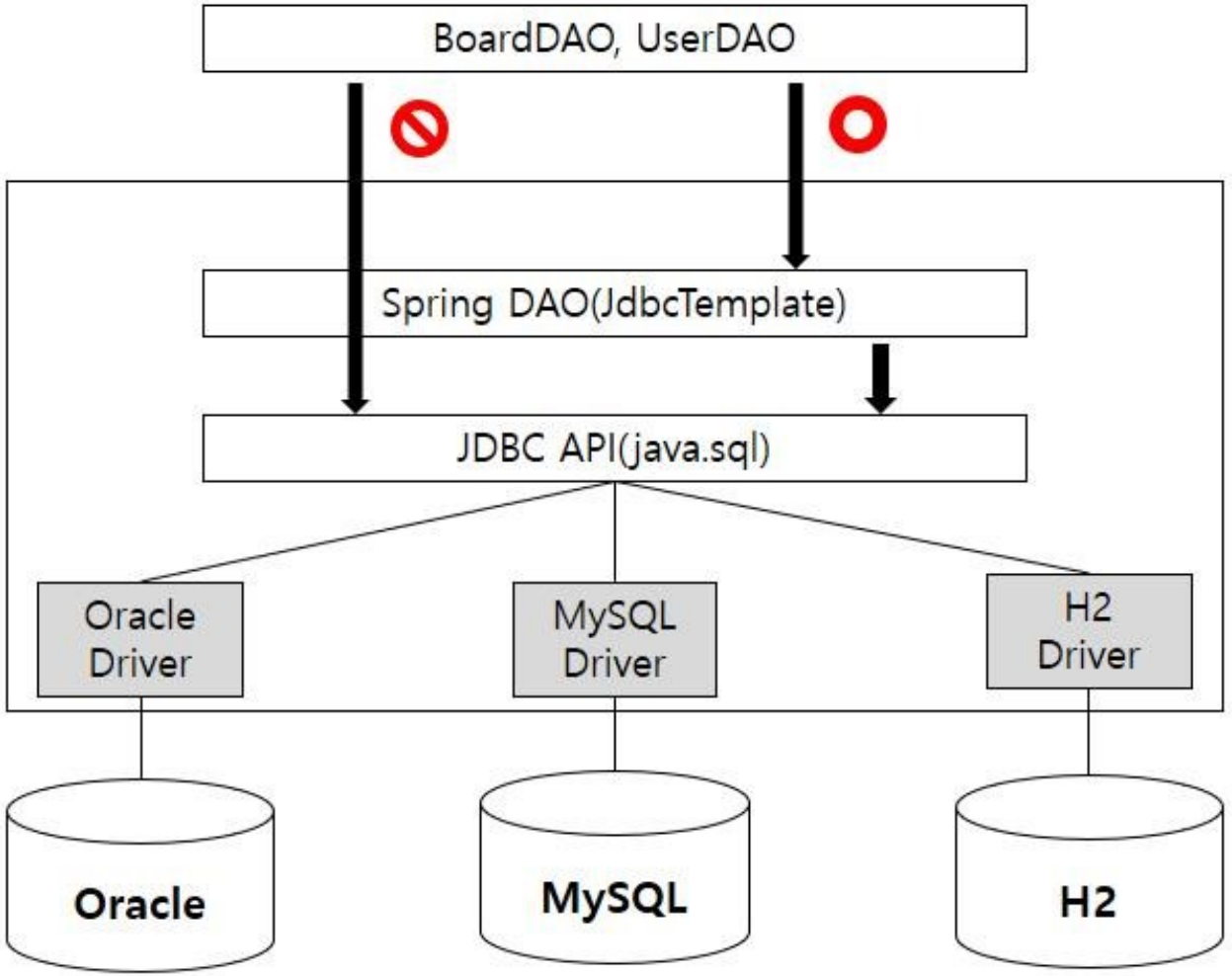
```
<context:property-placeholder location="classpath:database.properties" />

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
                  destroy-method="close">
    <property name="driverClassName" value="${jdbc.driver}"/>
    <property name="url"                value="${jdbc.url}"/>
    <property name="username"        value="${jdbc.username}"/>
    <property name="password"        value="${jdbc.password}"/>
</bean>
```

database.properties

```
jdbc.driver=
jdbc.url=
jdbc.username=
jdbc.password=
```

- Spring JDBC 구조



Spring JDBC – DAO클래스에서 JdbcTemplate객체 얻는 법

- **2가지 방법**
  1. **첫번째 방법 :**
     JdbcDaoSupport클래스를 상속하여 → 수퍼클래스에 dataSource 객체를 setting한 후, getJdbcTemplate( )메소드를 통해 JdbcTemplatem 객체를 리턴받음

```
@Repository
public class BoardDAOSpring extends JDbcDaoSupport{
    @Autowired
     public void setSuperDataSource( DataSource dataSource) {
            super. setSuperDataSource (dataSource) ;
     }
       // 글                  
     public BoardVO getBoard( BoardVO vo ){
          Object[] args = {vo.getSeq()};
          return getJdbcTemplate( ).queryForObject( "select * from board where seq=?", args, new BoardRowMapper( ) );
      }
}
```

Spring JDBC – DAO클래스에서 JdbcTemplate객체 얻는 법

SpringProject6_Jdbc_step1 프로젝트

2. **두번째 방법:**
   JdbcTemplate 클래스를 <bean> 등록하여 의존성 주입

```
<bean class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
</bean>
```

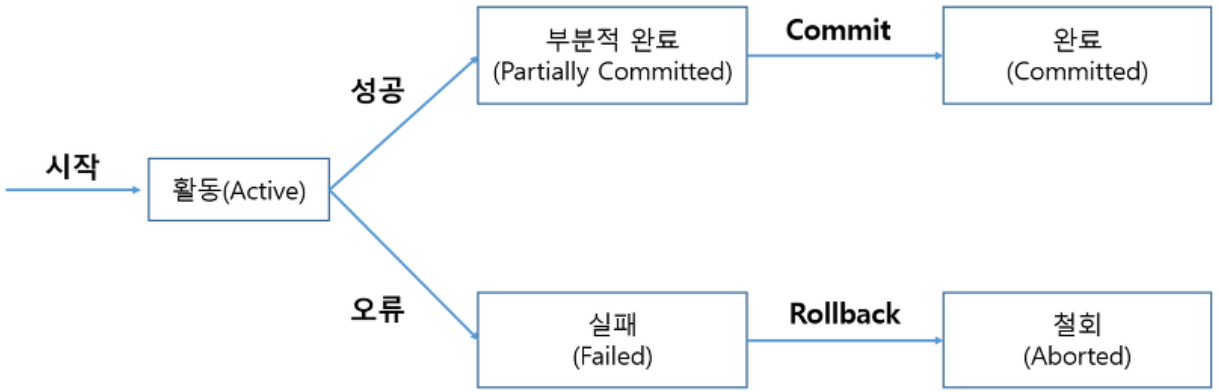
```
@Repository
public class BoardDAOSpring {
     @Autowired
      private JdbcTemplate spring;

        // 글                    
      public BoardVO getBoard( BoardVO vo ){
           Object[] args = {vo.getSeq()};
           return spring.queryForObject( "select * from board where seq=?", args, new BoardRowMapper( ) );
       }
}
```

- **Transaction 이란?**
  데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위
  또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 의미

보다 복잡한 프로그램을 개발하다 보면 쿼리 한 줄로 해결할 수 없는 로직을 처리해야하는 경우가 많습니다. 여러 개의 쿼리가 처리되는 상황에서 문제가 생겨버린다면 시스템에 큰 결함을 남기게 된다. 예를 들어 쇼핑몰 서비스를 구현한다고 했을때 ...

먼저 쇼핑몰에서 상품을 구매할 때 회원의 잔여 금액이 충분한지 확인하고 잔여 금액이 상품 가격보다 높을 때 구매 로직으로 넘어가야 한다. 그리고 상품의 재고가 있는지 확인 후에 회원의 잔여 금액을 상품 가격만큼 감소시키고 로직을 종료해야 한다. 그런데 선택상품구매 단계에서 Exception()이 발생하여 상품이 없음에도 불구하고 있다고 판단하였거나 잔여 금액이 감소하는 찰나에 서버의 전원이 나가서 상품을 구매했는데도 회원의 잔여 금액이 감소하지 않을 수 있는 상황이 발생 될 수 있다. 이러한 문제를 해결하기 위해 Transaction 기술이 필요한 것이다.



Active(활동)	트랜잭션이 실행 중에 있는 상태, 연산들이 정상적으로 실행 중인 상태
Failed(장애)	트랜잭션이 실행에 오류가 발생하여 중단된 상태
Aborted(철회)	트랜잭션이 비정상적으로 종료되어 Rollback 연산을 수행한 상태
Partially Committed (부분 완료)	트랜잭션이 마지막 연산까지 실행했지만, Commit 연산이 실행되기 직전의 상태
Committed(완료)	트랜잭션이 성공적으로 종료되어 Commit 연산을 실행한 후의 상태

Spring JDBC – 트랜잭션 설정하는 법

SpringProject6_Jdbc_step2 프로젝트

- **2가지 방법**
  1. **첫번째 방법 :**
     ①트랜잭션 관리자를 xml에 등록 → ②트랜잭션 어드바이스를 xml에 설정 → ③AOP 설정을 통한 적용

①

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="${jdbc.driver}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
</bean>
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

속 성	의 미
name	트랜잭션이 적용될 메소드 이름 지정
read-only	읽지 전용 여부 지정(기본값 false)
no-rollback-for	트랜잭션을 롤백하지 않을 예외 지정
rollback-for	트랜잭션을 롤백할 예외 지정

②

```
<tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
                <tx:method name="get*" read-only="true"/>
                <tx:method name="*"/>
        </tx:attributes>
</tx:advice>
```

③

```
<aop:config>
    <aop:pointcut id="txPointcut"
                expression="execution(* com.springbook.biz..*(..))"/>
    <aop:advisor pointcut-ref="txPointcut" advice-ref="txAdvice"/>
</aop:config>
```

2. 두번째 방법 :

①트랜잭션 관리자를 xml에 등록 → ②<tx:annotation-driven/> xml에 추가 → ③ @Transactional 추가

①

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="${jdbc.driver}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
</bean>
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>
```

②

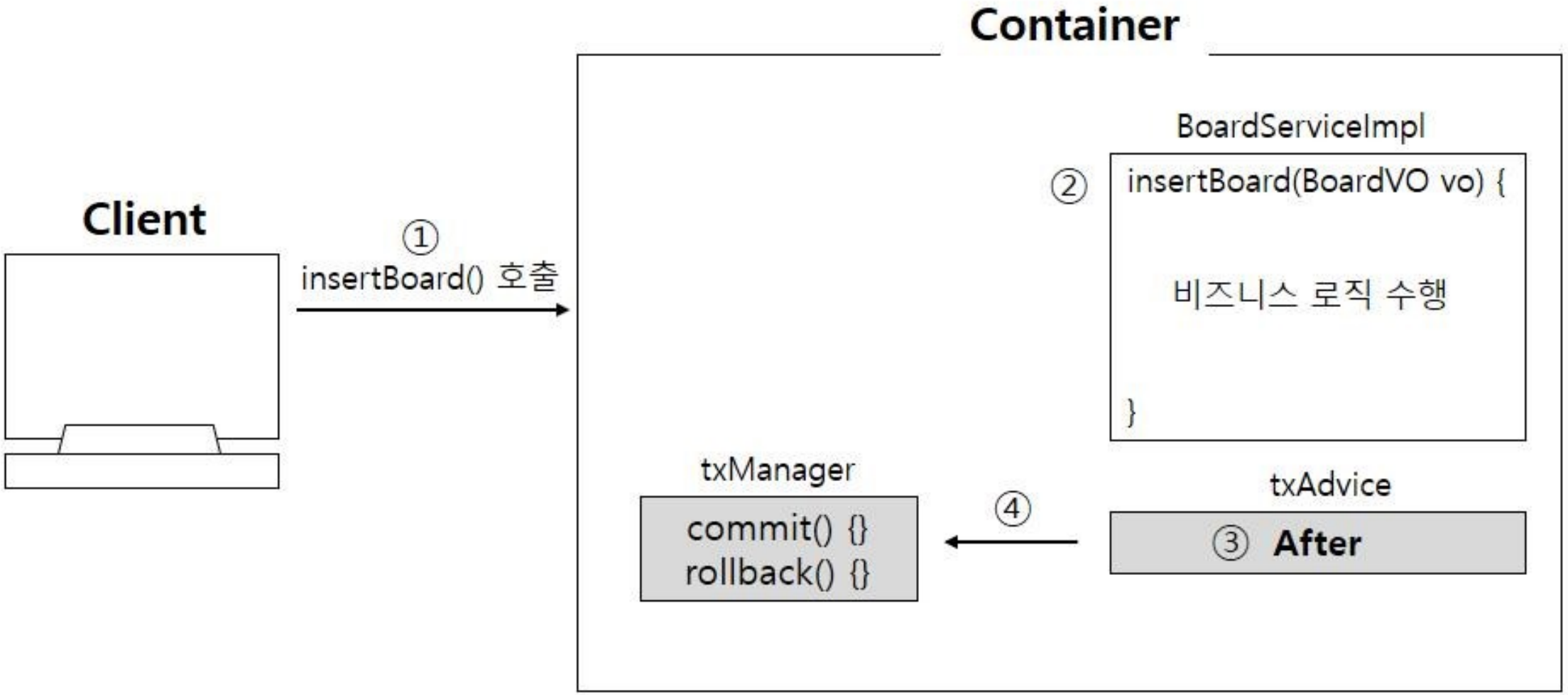
```
<tx:annotation-driven/>
```

③

```
@Transactional("txManager")
public void insertBoard(BoardVO vo) {
        ......
}
```

Spring JDBC – 트랜잭션 동작

SpringProject6_Jdbc_step2 프로젝트



Spring Layered architecture 구조 – MVC 패턴 p241

1) Presentation Layer

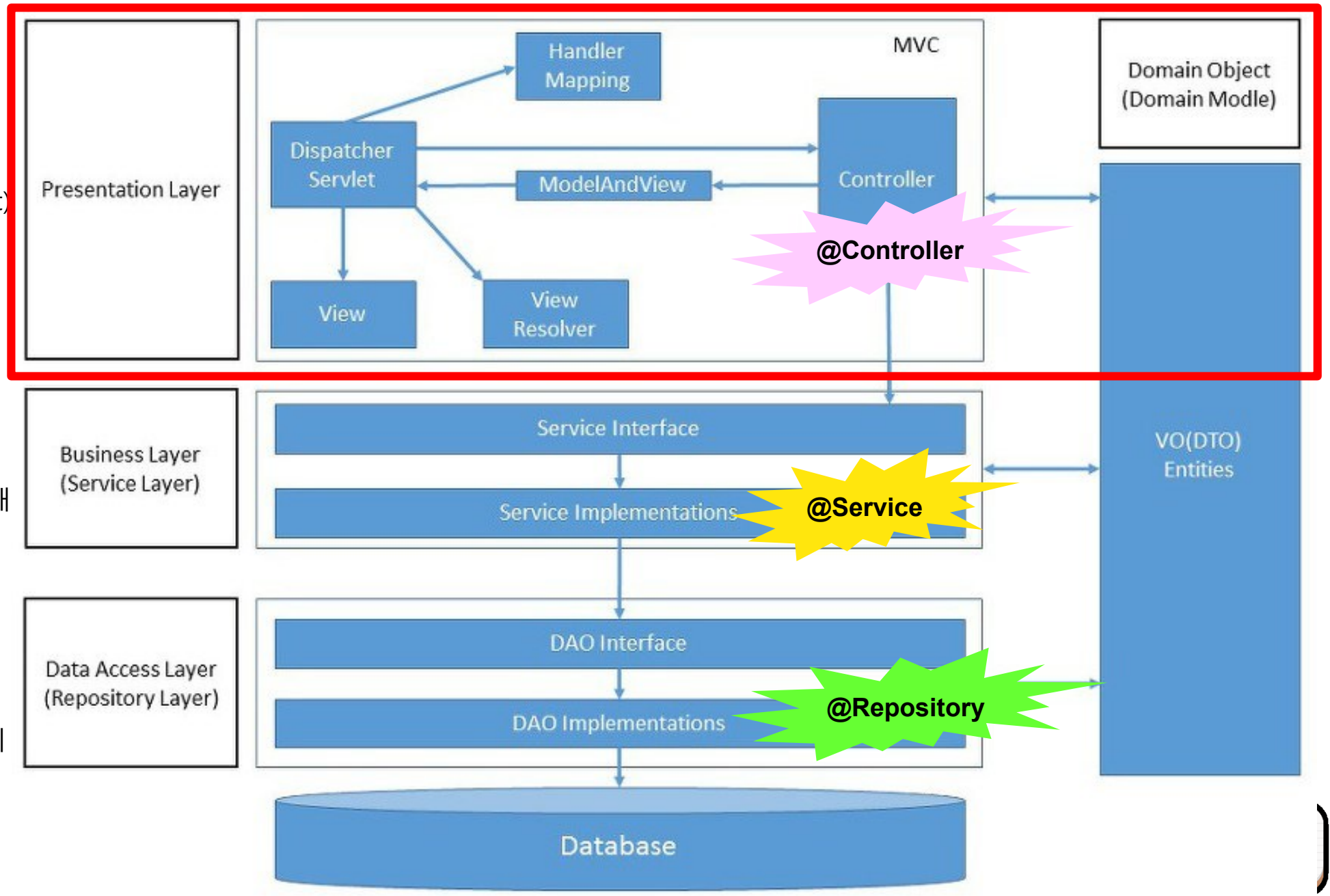
Spring MVC 객체를 말한다.
프론트 컨트롤러(DispatcherServlet)
컨트롤러, 뷰, 모델이 포함

2) Service Layer(Business Layer)

presentation layer에서 요청을 보내면 실제로 비지니스로직 수행

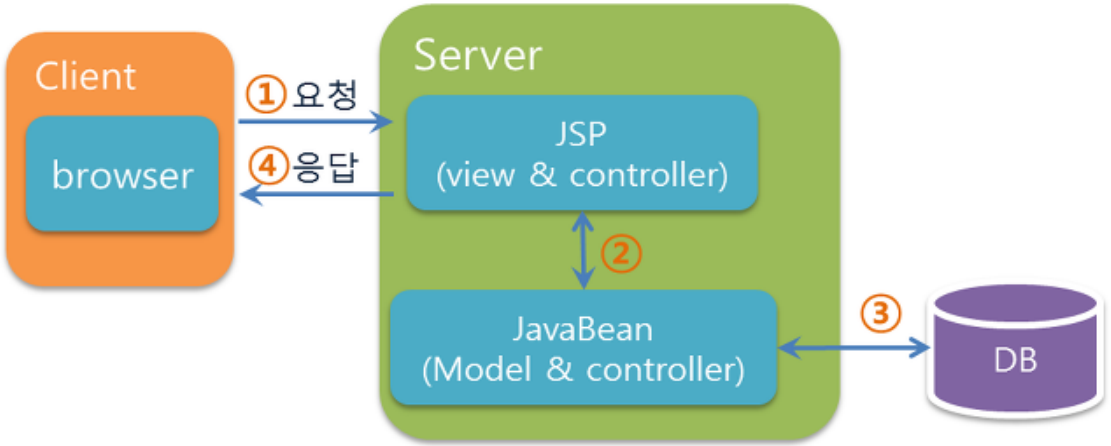
3) Data Access Layer(Repository Layer)

DB에 값을 저장하거나 가져오기 위해 JDBC, Mybatis, JPA 등을 사용해 구현한 DAO

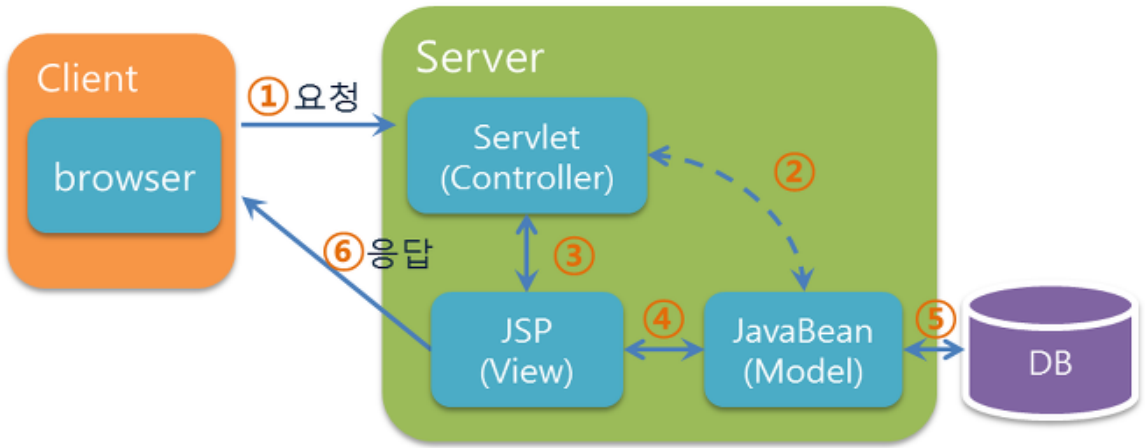


MVC 패턴 : 모델 1과 모델 2의 비교 p241

- **MVC 패턴 :**
  MVC패턴의 최대 장점은 사용자에게 보여지는 프레젠테이션 영역과 비즈니스 로직, 데이터 구조가 서로 완전히 분리되어 있다는 점



<MVC – Model 1>



<MVC – Model 2>

모델1: 비즈니스 로직 영역(Controller)에 프레젠테이션 영역(View)을 같이 구현하는 방식

모델2: 비즈니스 로직 영역과 프레젠테이션 영역이 분리되어 있는 구현 방식

	모델1	모델2
컨트롤러와 뷰의 분리 여부	통합(JSP파일)	분리(JSP, Servlet)
장점	쉽고 빠른 개발	디자이너/개발자 분업 유리 유지보수에 유리
단점	유지보수가 어려움	설계가 어려움 개발 난이도가 높음

MVC 패턴 : Model 1으로 게시판 p241

SpringProject7_MVC_step1 프로젝트

- **모델 1 :** 적은 개발인력으로 간단한 프로젝트 수행시 사용
  - JSP
    - View 역할 : HTML, CSS
    - Controller 역할 : 사용자의 요청 처리와 관련된 자바 코드 의미 (jsp내의 모든 자바코드가 controller는 아님)

기 능	사 용 예
사용자 입력 정보 추출	String id = request.getParameter("userId")
DB 연동 처리	UserVO vo = new UserVO(); vo.setId(id); UserDAO userDAO = new UserDAO(); UserVO user = userDAO.getUser(vo);

화면 내비게이션	if(user != null) { // 로그인 성공
	response.sendRedirect("getBoardList.jsp");
	} else { // 로그인 실패
	response.sendRedirect("login.jsp");
	}

  - JavaBean
    - Model 역할 : DB연동 로직을 제공하며, DB에서 검색한 데이터가 저장되는 자바 객체. VO와 DTO

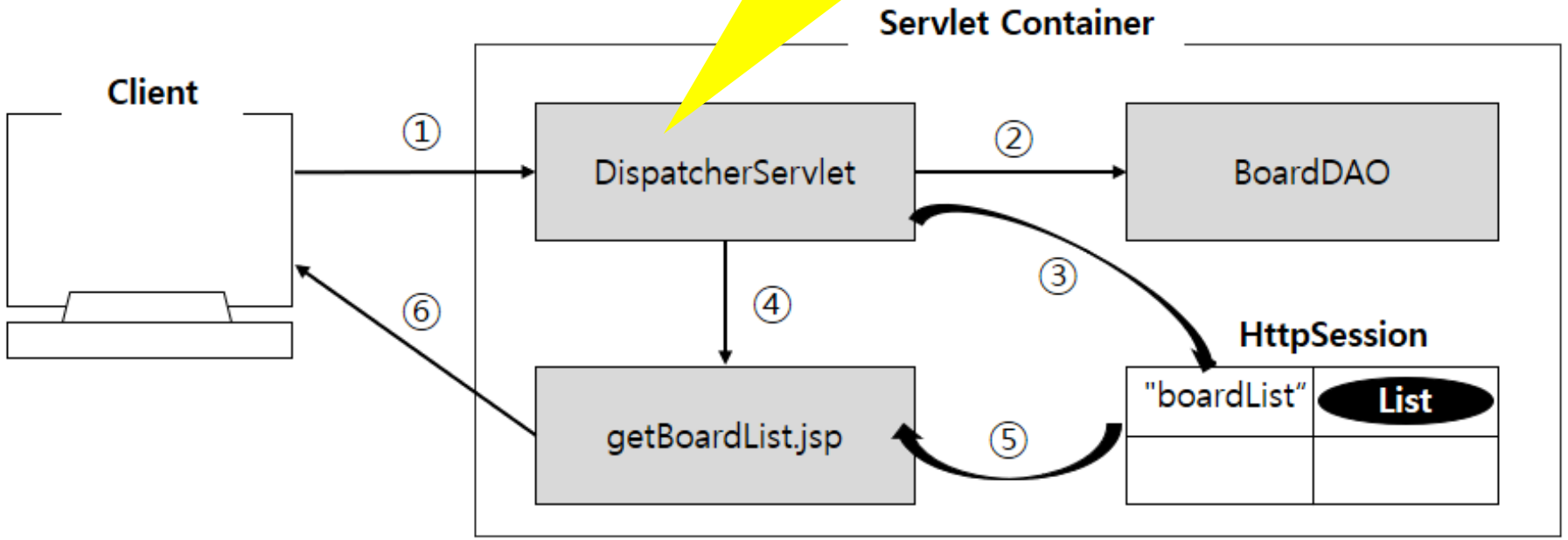
MVC 패턴 : Model 2으로 게시판 p263

SpringProject7_MVC_step2 프로젝트

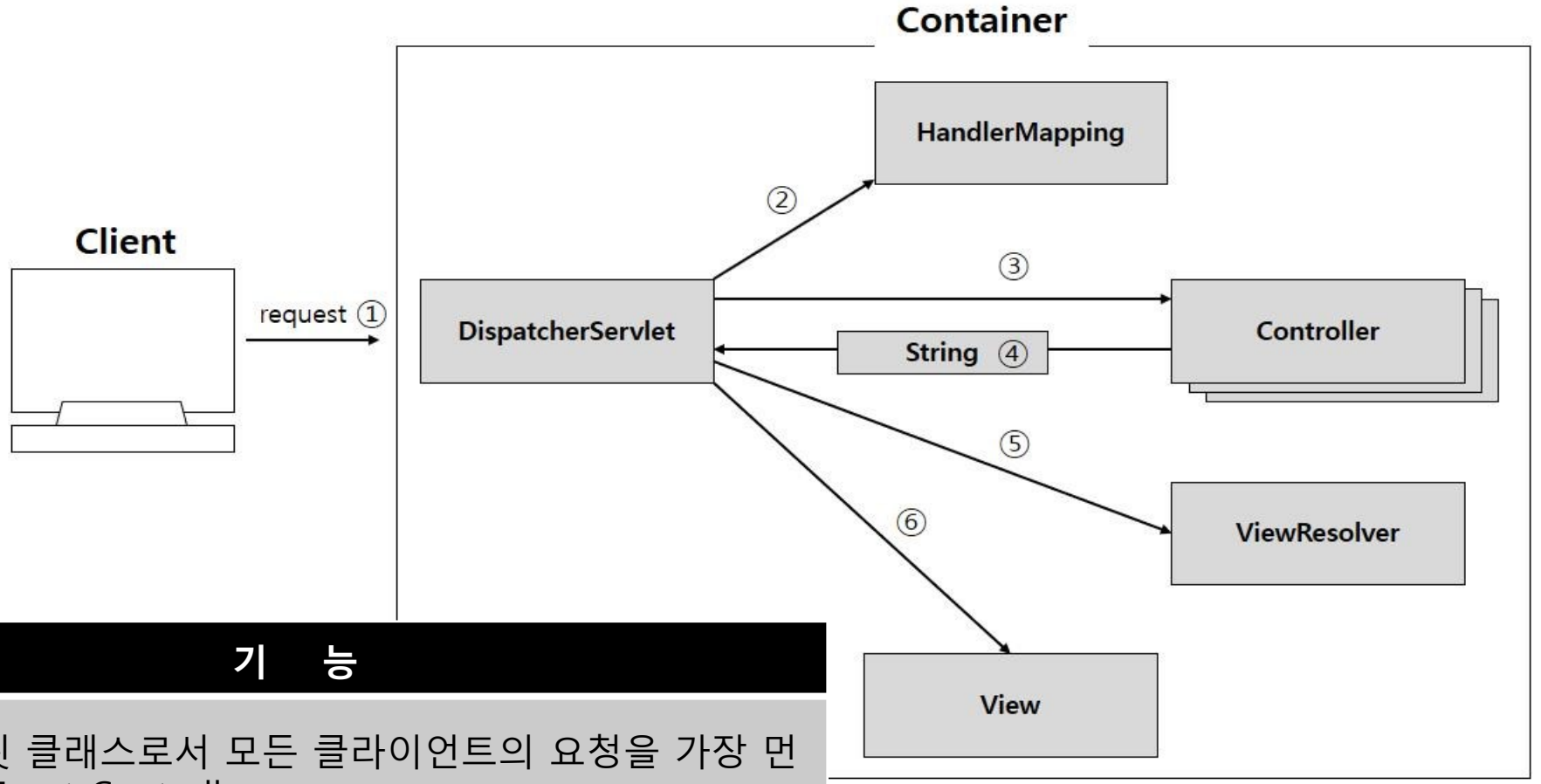
- **모델 2**

기 능	구성 요소	개발 주체
Model	VO, DAO 클래스	자바 개발자
View	JSP 페이지	웹 디자이너
Controller	Servlet 클래스	자바 개발자 또는 MVC 프레임워크

MVC Framework 직접 구현



- MVC Framework 구조



클 래 스	기 능
DispatcherServlet	유일한 서블릿 클래스로서 모든 클라이언트의 요청을 가장 먼저 처리하는 Front Controller
HandlerMapping	클라이언트의 요청을 처리할 Controller 매핑
Controller	실질적인 클라이언트의 요청 처리
ViewResolver	Controller가 반환한 View 이름으로 실행될 JSP 경로 완성