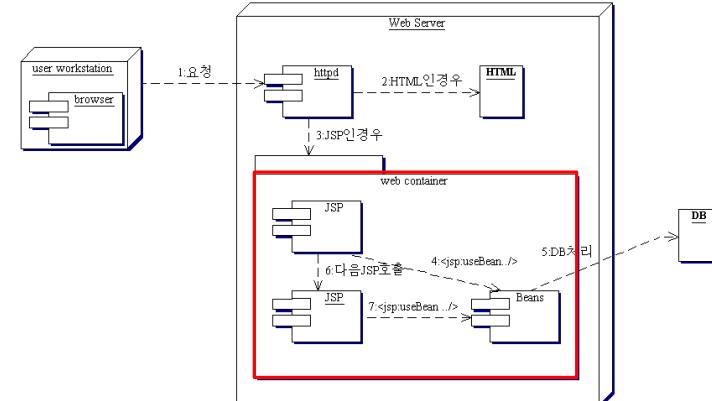


Web Application의 구조

4. MVC Architecture

◆ Model 1

- JSP page에서 presentation logic과 business logic, 입출력 데이터 처리, 실행 흐름 제어 등을 모두 구현
- 작은 규모, 짧은 기간의 프로젝트 수행 시 사용 가능
- 복잡하고 변경이 많은 응용 프로그램의 경우 부적합
 - 개발, 유지보수, 재사용에 어려움이 큼

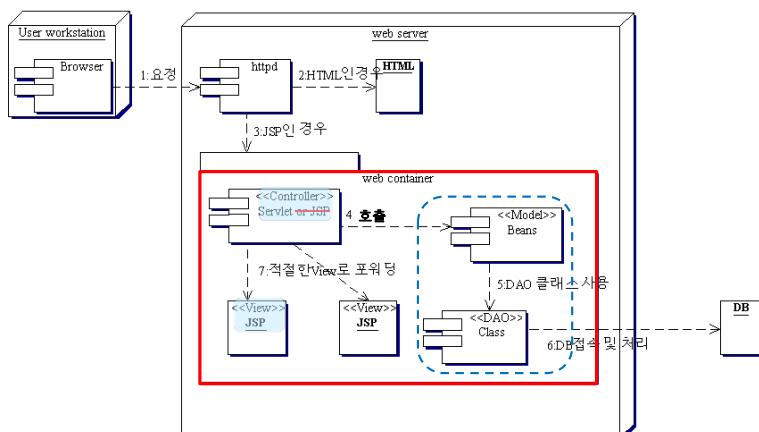


2

Web Application의 구조

◆ MVC Architecture/Pattern (Model 2)

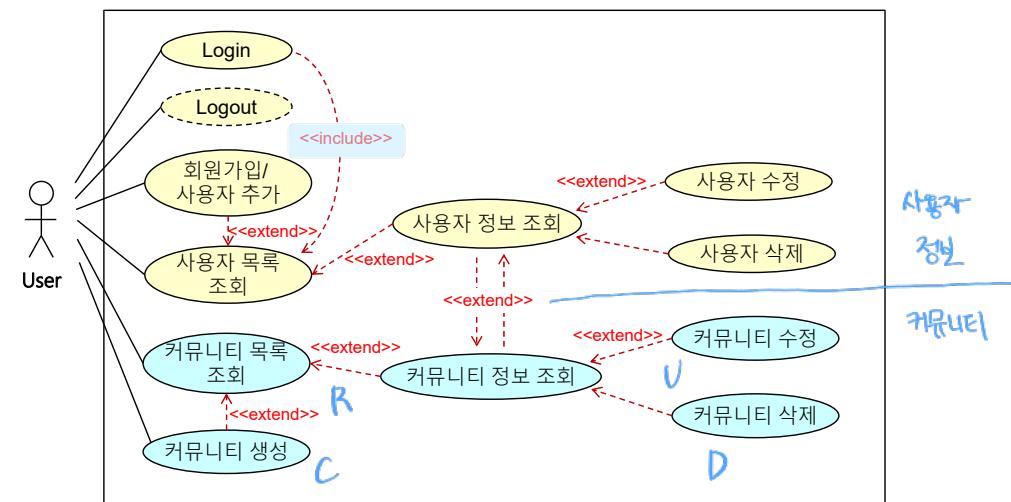
- **Model:** business logic 및 data 처리(저장, 관리, 검색) 수행 Java
- **View:** UI 및 presentation logic 구현 → 입출력 화면 생성 JSP
- **Controller:** 사용자 입력 처리, Model과 View 사이의 실행 흐름 제어 Servlet



3

Example: 사용자 관리

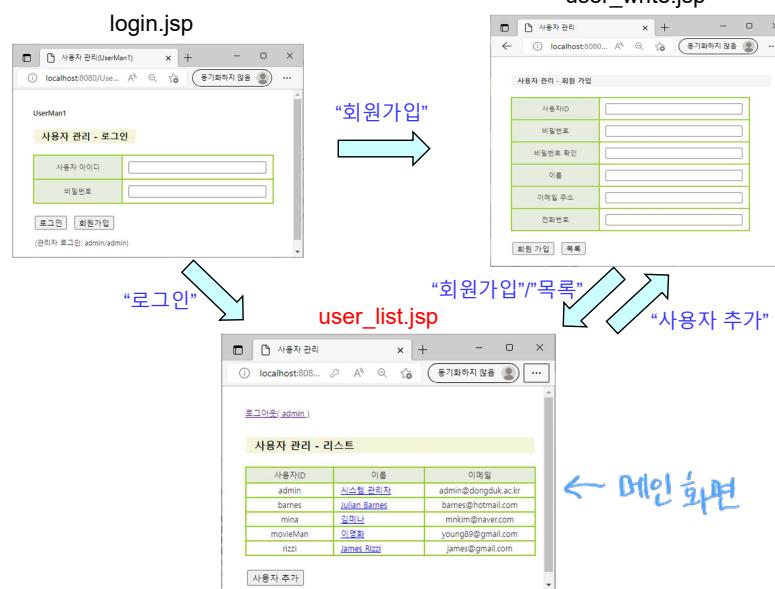
◆ Use Cases



4

Example: 사용자 관리

◆ UI 화면 설계



Example: 사용자 관리

user_list.jsp

사용자ID	이름	이메일
admin	시스템 관리자	admin@dongduk.ac.kr
barnes	Julian Barnes	barnes@hotmail.com
mina	길민나	mnmin@naver.com
movieMan	영화관	young8@gmail.com
rizzi	James Rizzi	james@gmail.com

이름 선택
“삭제”/“목록”

사용자ID	이름	이메일
barnes	Julian Barnes	barnes@hotmail.com
mina	길민나	mnmin@naver.com
young	영화관	young8@gmail.com
rizzi	James Rizzi	james@gmail.com

user_view.jsp

사용자ID	비밀번호	이름	이메일 주소	전화번호
barnes	*****	Julian Barnes	barnes@hotmail.com	778-443-1532

수정 “수정”

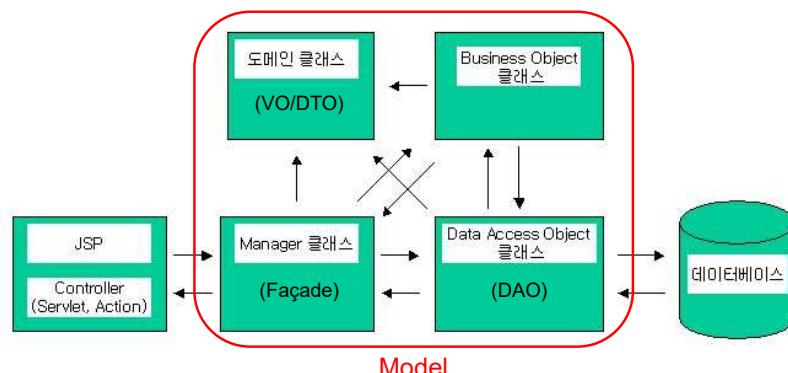
user_modify.jsp

“저장”/“삭제”
반드시 포함 X

모델(Model)

◆ 모델의 역할

- business logic 구현
- 데이터 처리
 - Database, file system, legacy system 등과의 연동 수행



모델(Model)

◆ Domain class

- Value Object(VO) 및 Data Transfer Object(DTO)를 정의
- Application에서 사용되는 데이터의 표현 및 전달을 위한 객체
- Database에 저장되는 객체는 테이블과 유사한 구조를 가짐
- 속성에 대한 setter & getter methods 포함

◆ Business class

- Business logic을 구현하는 클래스
- Business logic이 간단할 경우 Manager class에서 구현 가능

모델(Model)

◆ Data Access Object(DAO) class

- Database나 기존 legacy system과 연동하여 데이터 처리 및 관리 수행
- JDBC, MyBatis 등을 이용하여 구현

◆ Manager(Facade) class = Service Class

- JSP(in Model 1) 또는 Controller(in Model 2)에서 모델에 접근하기 위해 사용하는 인터페이스를 제공하는 Facade class
 - 외부에서 모델에 접근할 때 반드시 manager 객체를 이용하도록 함
- Manager 객체는 Business 객체와 DAO를 호출하여 business logic 및 데이터 처리를 실행시키고 그 결과를 Controller에 전달하는 역할을 수행
 - domain 객체를 통해 데이터 및 결과 전달
- Manager class에서 간단한 business logic을 직접 구현 가능

9

모델(Model): Example

◆ Database Schema

```
CREATE TABLE UserInfo (
    userId      varchar2(12)  PRIMARY KEY,
    password    varchar2(12)  NOT NULL,
    name        varchar2(20)  NOT NULL,
    email       varchar2(50),
    phone       varchar2(20)
    commId      NUMBER(4); ← Community를 참조하는 외래키
```

```
CREATE TABLE Community (
    cId         NUMBER(4)    PRIMARY KEY,
    cName       varchar2(22)  NOT NULL,
    descr       varchar2(50),
    startdate   Date,
    chairId    varchar2(12));
```

상당히

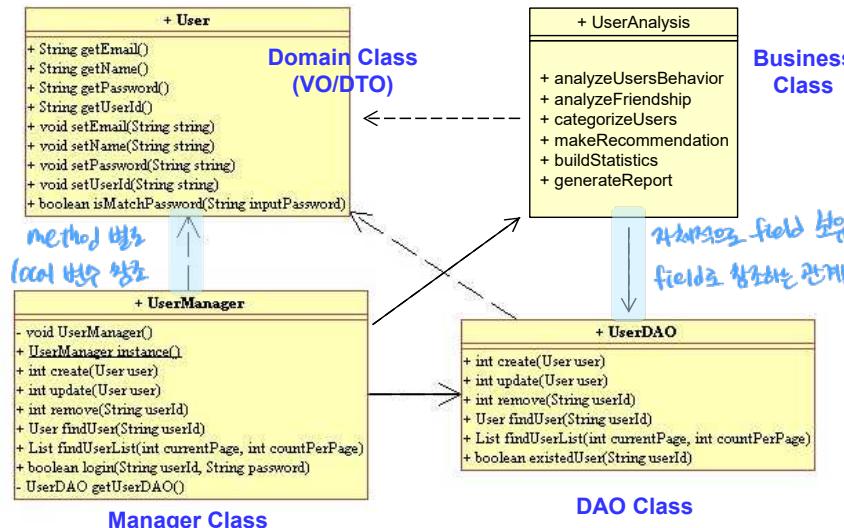
```
ALTER TABLE UserInfo ADD FOREIGN KEY (commId) REFERENCES Community (cId);
ALTER TABLE Community ADD FOREIGN KEY (chairId) REFERENCES UserInfo (userId);
```

```
CREATE SEQUENCE commId_seq START WITH 10 INCREMENT BY 10;
```

10

모델(Model): Example

◆ Class Diagram



11

■ User class

```

package user;
/* 사용자 관리를 위해 필요한 도메인 클래스.
 * UserInfo 테이블과 대응됨 */
public class User {
    private String userId;
    private String password;
    private String name;
    private String email;
    private String phone;
    // private int commId; // in UserMan3
    // private String commName;

    public User() {}
    public User(String userId, String pw, String name, String em, String ph) {
        this.userId = userId;
        this.password = password;
        this.name = name;
        this.email = email;
        this.phone = phone;
    }

    public void setId(String s) {
        userId = s;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String s) {
        password = s;
    }
    ...
    /* 비밀번호 검사 */
    public boolean matchPassword(String pword) {
        if (pword == null) {
            return false;
        }
        return this.password.equals(pword);
    }

    public boolean isSameUser(User user) {
        return userId.equals(user.userId);
    }
    ...
}
```

■ UserDAO class

```

package user;
/* DB 연동 담당: JDBC API를 이용하여 UserInfo 테이블에 사용자 정보 추가/수정/삭제/검색 수행 */
public class UserDAO {
    private JDBCUtil jdbcUtil = new JDBCUtil(); // JDBCUtil 활용

    /* UserInfo 테이블에 새로운 사용자 레코드 생성. */
    public int create(User user) throws SQLException {
        private static String query = "INSERT INTO UserInfo VALUES (?, ?, ?, ?, ?)";
        Object[] param = new Object[] {user.getUserId(), user.getPassword(),
            user.getName(), user.getEmail(), user.getPhone()};

        jdbcUtil.setSql(query); // insert 문 설정
        jdbcUtil.setParameters(param); // insert 문에 대한 매개변수 설정
        try {
            int result = jdbcUtil.executeUpdate(); // insert 문 실행
            return result; // insert 된 행의 개수 반환
        } catch (Exception ex) {
            jdbcUtil.rollback();
            ex.printStackTrace();
        } finally {
            jdbcUtil.commit();
            jdbcUtil.close(); // insert 완료
        }
        return 0;
    }
}

```

■ UserDAO class (cont'd)

```

/* 사용자들의 정보를 검색한 후 페이지 번호와 페이지당 출력할 사용자 수를 이용하여 해당되는
 * 사용자 정보를 List에 저장하여 반환. */
public List<User> findUserList(int currentPage, int countPerPage) throws SQLException {
    private static String query = "SELECT userId, password, name, email, phone " +
        "FROM UserInfo";
    jdbcUtil.setSql(query); // query 문 설정
    try {
        ResultSet rs = jdbcUtil.executeQuery(); // query 문 실행
        int start = ((currentPage-1) * countPerPage) + 1;
        List<User> userList = null;
        if (start >= 0) {
            userList = new ArrayList<User>();
            do {
                User user = new User();
                user.setUserId(rs.getString("userId"));
                user.setPassword(rs.getString("password"));
                user.setName(rs.getString("name"));
                user.setEmail(rs.getString("email"));
                user.setPhone(rs.getString("phone"));
                userList.add(user);
            } while (rs.next() && (--countPerPage > 0));
        }
        return userList;
    } catch (Exception ex) {
    } finally {
        ... jdbcUtil.close(); // resource 반환
    }
    return null;
}

```

■ UserDAO class (cont'd)

```

/* 사용자 정보를 DB에서 검색하여 User 도메인 클래스에 저장 및 반환 */
public User findUser(String userId) throws SQLException {
    private static String query = "SELECT password, name, email, phone " +
        "FROM UserInfo WHERE userId=?";
    jdbcUtil.setSql(query); // query 문 설정
    Object[] param = new Object[] {userId};
    jdbcUtil.setParameters(param); // query 문에 userId를 매개변수로 설정
    try {
        ResultSet rs = jdbcUtil.executeQuery(); // query 실행
        User user = null;
        if (rs.next()) {
            user = new User();
            user.setUserId(userId);
            user.setPassword(rs.getString("password"));
            user.setName(rs.getString("name"));
            user.setEmail(rs.getString("email"));
            user.setPhone(rs.getString("phone"));
        }
        return user;
    } catch (Exception ex) {
    } finally {
        ... jdbcUtil.close(); // resource 반환
    }
    return null;
}

```

■ UserDAO class (cont'd)

```

/* 기존 사용자 정보를 수정 */
public int updateUser(User user) throws SQLException {
    private static String query = "UPDATE UserInfo " +
        "SET password=?, name=?, email=?, phone=? WHERE userId=?";
    Object[] param = new Object[] {user.getPassword(), user.getName(),
        user.getEmail(), user.getPhone(), user.getUserId()};
    jdbcUtil.setSql(query); // update 문 설정
    jdbcUtil.setParameters(param); // update 문에 대한 매개변수 설정
    try {
        int result = jdbcUtil.executeUpdate(); // update 문 실행
        return result; // update 된 행의 개수 반환
    } catch (Exception ex) {
    } finally {
        ... jdbcUtil.close(); // resource 반환
    }
    return 0;
}

/* 사용자 아이디에 해당하는 사용자를 삭제 */
public int remove(String userId) throws SQLException {
    ... (소스 파일 참조) ...
}

/* 인자로 전달되는 ID를 갖는 사용자 존재 여부 판별 */ → 3.1인증API
public boolean existedUser(String userId) throws SQLException {
    ... (소스 파일 참조) ...
}

```

■ UserManager class

```

package user;
import java.sql.SQLException; ...
/*
 * 사용자 관리 API를 제공하는 클래스:
 * UserDao를 이용하여 DB와 연동하고 business object를 호출하거나 직접 business logic을 수행함
 */
public class UserManager {
    private static UserManager userMan = new UserManager();
    private UserDao userDao;

    private UserManager() {
        try {
            userDao = new UserDao(); → 외부에서 new 할 수 없도록 static으로 선언
        } catch (Exception e) { e.printStackTrace(); }
    }

    public static UserManager getInstance() { return userMan; }

    public int create(User user) throws SQLException, ExistedUserException {
        if (userDao.existedUser(user.getUserId())) {
            throw new ExistedUserException(user.getUserId() + "는 존재하는 아이디입니다.");
        }
        return userDao.create(user);
    }
}

```

Singleton Pattern:
UserManager
객체(instance)를 하나만
생성 및 사용(공유)

■ UserManager class (cont'd)

```

public int update(User user) throws SQLException {
    return userDao.update(user);
}

public int remove(String userId) throws SQLException {
    return userDao.remove(userId);
}

public User findUser(String userId) throws SQLException, UserNotFoundException {
    User user = userDao.findUser(userId);
    if (user == null) {
        throw new UserNotFoundException(userId + "는 존재하지 않는 아이디입니다.");
    }
    return user;
}

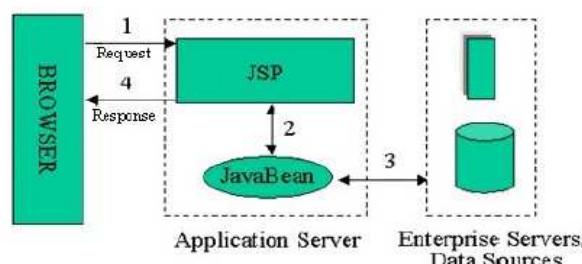
public List findUserList(int currentPage, int countPerPage) throws SQLException {
    return userDao.findUserList(currentPage, countPerPage);
}

public boolean login(String userId, String password)
    throws SQLException, UserNotFoundException, PasswordMismatchException {
    User user = userDao.findUser(userId);
    if (user.matchPassword(password) == false) {
        throw new PasswordMismatchException("비밀번호가 일치하지 않습니다.");
    }
    return true;
}
}

```

Model 1 구조 참고~

◆ 시스템 구조

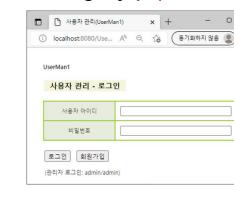


- 초기 Web Application 개발 방식
- 각 JSP page가 클라이언트로부터 HTTP request를 받아 parameter 추출, 모델 호출, 결과 화면 생성(결과 출력) 또는 forward/redirection 등을 모두 수행
- Database를 이용한 데이터 처리나 legacy system과의 연동도 JSP page에서 직접 수행하거나 JavaBeans 객체(Model)를 이용하여 수행

Model 1 구조: Example

◆ Request 처리 흐름

login.jsp (입력 Form 생성)



“로그인” → login_action.jsp (DB 검색)

user_list.jsp (DB 검색, 결과 출력)

사용자 ID	이름	이메일
admin	스失调	admin@donguk.ac.kr
benes	Julian Benes	benes@donguk.ac.kr
minseok	민석	minseok@donguk.ac.kr
youngjin	정진	youngjin@gmail.com
nci	장재우	jangleu@gmail.com

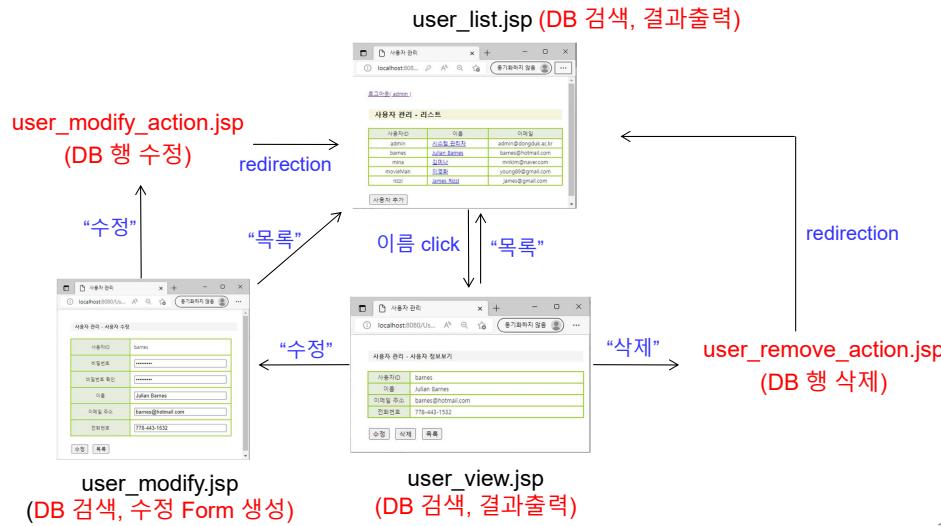
“회원가입” → user_write.jsp (입력 Form 생성)

“회원가입” → user_write_action.jsp (DB 행 삽입)



Model 1 구조: Example

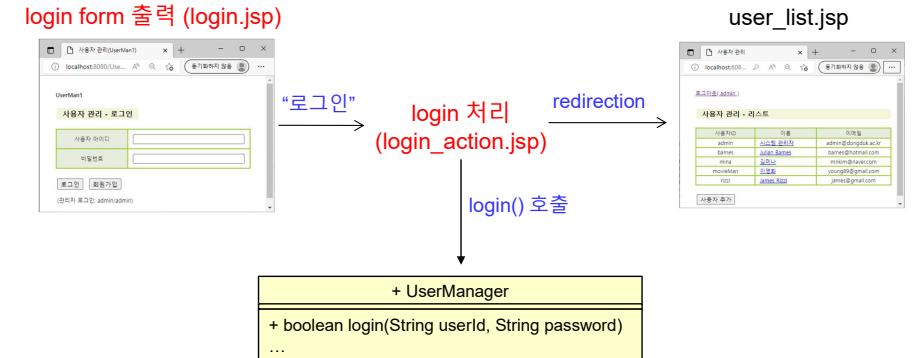
◆ Request 처리 흐름 (cont'd)



21

Model 1 구조: Example

◆ 예 1: 로그인



22

■ login form 출력 (login.jsp)

```
<%@page contentType="text/html; charset=utf-8" %>
<html>
<head>
<title>사용자 관리</title>
<script language="JavaScript">
function userCreate() {
    form.action = "user_write.jsp";
    form.submit();
}
function login() {
    form.action = "login_action.jsp";           // 요청이 전달될 JSP page (URL)
    form.submit();
}
</script></head>
<body>
<b>사용자 관리 - 로그인</b>
<form name="form" method="POST">
    사용자 아이디: <input type="text" style="width:150" name="userId"><br>
    비밀번호: <input type="password" style="width:150" name="password"><br>
    <input type="button" value="로그인" onClick="login()"><br>
    <input type="button" value="회원가입" onClick="userCreate()"><br>
</form>
</body>
```

■ login 처리 (login_action.jsp)

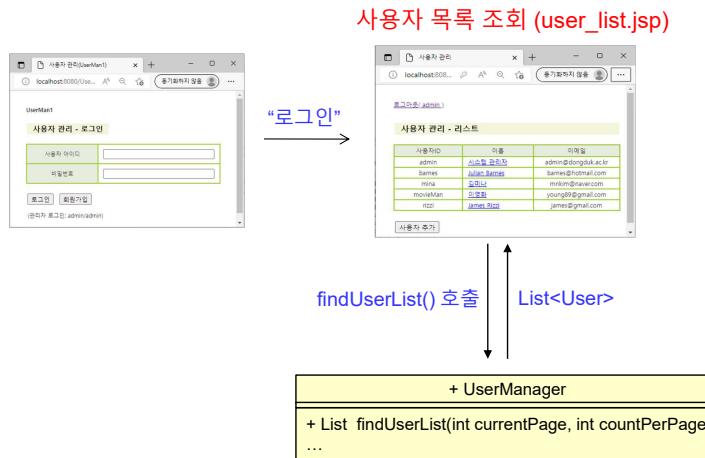
```
<%@page contentType="text/html; charset=utf-8" %>
<%@page import="user.*" %>
<%
try {
    String userId = request.getParameter("userId");      // parameter 추출
    String password = request.getParameter("password");

    UserManager manager = UserManager.getInstance();
    manager.login(userId, password);                  // 모델 호출: login 처리를 위임

    session.setAttribute("userId", userId);            // 정상적으로 login된 경우 세션에 userId 저장
    response.sendRedirect("user_list.jsp");             // 사용자 목록 조회 요청으로 redirection
    // (결과 화면 없음)
} catch (Exception e) {
%
<!-- 에러가 발생할 경우 이전 페이지로 이동 -->
<script language="javascript">
    alert("<%= e.getMessage() %>");
    history.back();
</script>
<%
}
%>
```

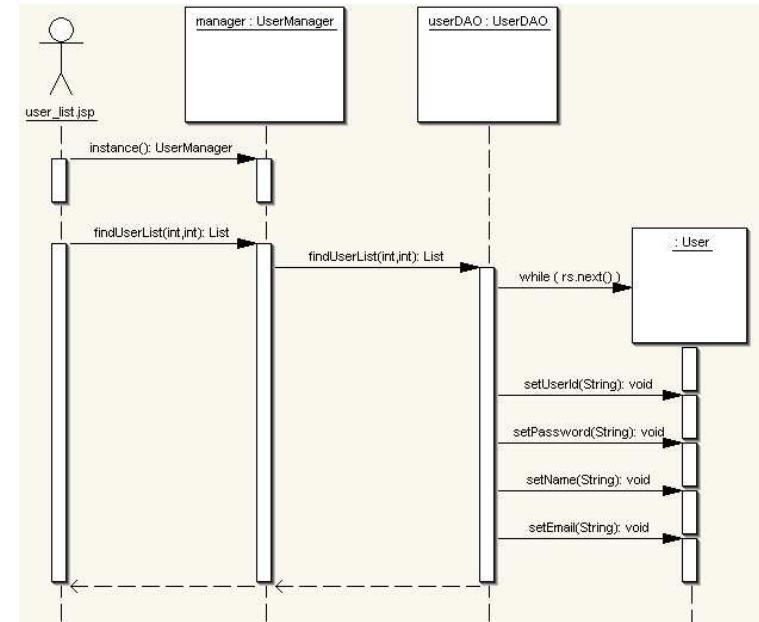
Model 1 구조: Example

◆ 예 2: 사용자 목록 조회



25

■ Sequence Diagram (Model 1)



26

■ 사용자 목록 조회 (user_list.jsp)

```

<%@page contentType="text/html; charset=utf-8" %>
<%@page import="java.util.*; user.*" %>
<%@include file="loginCheck.jsp" %>
<%
int currentPage = 1, countPerPage = 10;
String currentPageStr = request.getParameter("currentPage");           // parameter 추출
if (currentPageStr != null) && (!currentPageStr.equals("")) )
    currentPage = Integer.parseInt(currentPageStr);

// 모델을 이용하여 사용자 목록을 가져옴
UserManager manager = UserManager.getInstance();
List<User> userList = manager.findUserList(currentPage, countPerPage); // Model 호출
%>
<html> <!-- 결과 화면 생성 -->
<head><title>사용자 관리</title></head>
<body>
<form name="f" method="POST" action="user_write.jsp">
<table>
<tr><td width="20"></td>
<td>사용자 관리 - 리스트</td><br>
<table>
<tr>
<td>사용자 아이디</td>
<td>이름</td>
<td>이메일</td>

```

■ 사용자 목록 조회 (user_list.jsp) (cont'd)

```

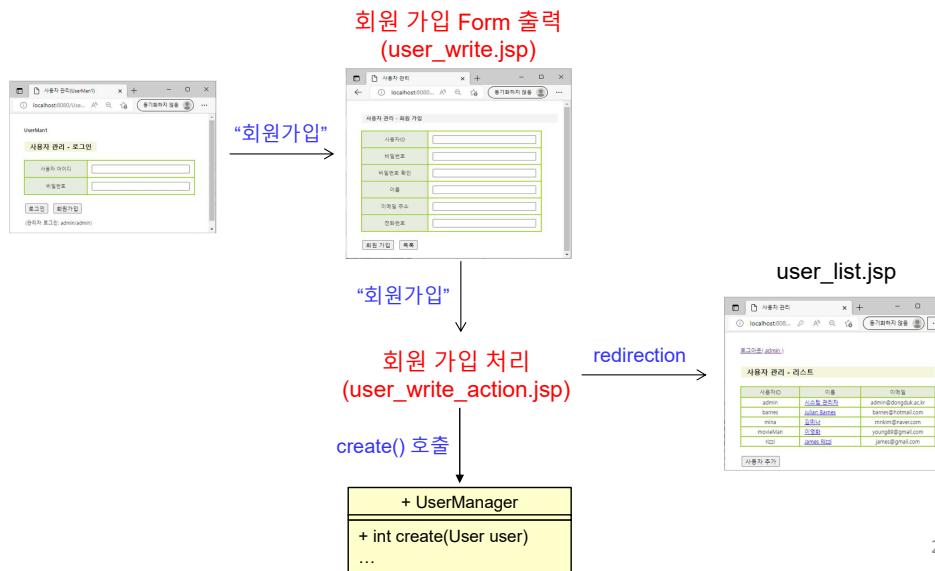
<%
// 검색된 사용자 목록을 출력
Iterator<User> userIter = userList.iterator();
while ( userIter.hasNext() ) {
    User user = (User)userIter.next();

    <tr>
        <td> <%=user.getUserId() %> </td>
        <td> <a href="user_view.jsp?userId=<%=user.getUserId()%>" class="user">
            <%=user.getName()%> </a>
        </td>
        <td> <%=user.getEmail() %> </td>
    </tr>
}
</table><br>
<table>
    <tr><td><input type="submit" value="사용자 추가"/></td></tr>
</table>
</td>
</tr>
</table>
</body>
</html>

```

Model 1 구조: Example

◆ 예 3: 회원가입



29

■ 회원 가입 Form 출력 (user_write.jsp)

```
<%@page contentType="text/html; charset=utf-8"%>
<html>
<head><title>사용자 관리</title>
<script language="JavaScript">
function userCreate() {
    if (form.userId.value == "") {
        alert("사용자 아이디를 입력하십시오.");
        form.userId.focus();
        return false;
    }
    form.submit();
}
</script></head>
<body>
<b>사용자 관리 – 회원 가입</b>
<!-- 입력 Form 생성 -->
<form name="form" method="POST" action="user_write_action.jsp">
    사용자 아이디: <input type="text" style="width:150" name="userId"><br>
    비밀번호: <input type="password" style="width:150" name="password"><br>
    이름: <input type="text" style="width:240" name="name">
    이메일주소: <input type="text" style="width:240" name="email">
    전화번호: <input type="text" style="width:240" name="phone">
    <input type="button" value="회원가입" onClick="userCreate()"><br>
</form>
</body></html>
```

■ 회원 가입 처리 (user_write_action.jsp) *서버의 Servlet으로 핸들링된다*

```
<%@page contentType="text/html; charset=utf-8"%>
<%@page import="user.*"%>

<%
String userId = request.getParameter("userId"); // parameter 추출
String password = request.getParameter("password");
String name = request.getParameter("name");
String email = request.getParameter("email");

User user = new User(userId, password, name, email, phone); // User 객체(VO) 생성

UserManager.getInstance().create(user);
response.sendRedirect("user_list.jsp");
%>
```

action tag를 이용하여 재작성 가능

```
<jsp:useBean id="user" class="user.User"/> // User 태입의 JavaBeans 객체 생성
<jsp:setProperty name="user" property="*"/> // parameter 추출 후 User 객체에 저장
```

결과화면(X)

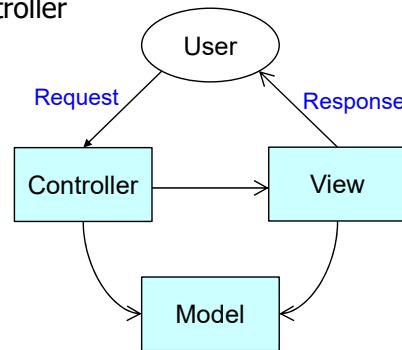
Model 1 구조

◆ Model 1 구조의 단점

- 복잡한 application의 경우 개발 및 유지보수의 어려움 발생
 - Presentation logic(HTML code)과 business logic(Java code)이 혼재될 경우 JSP page의 구조와 내용이 매우 복잡해짐
- 프로그램 개발자와 웹 디자이너 사이의 작업의 분리가 어려움
- 효율적인 개발 및 유지 보수가 어렵고, 확장성이나 재사용성이 크게 떨어짐

MVC 구조

◆ Model-View-Controller

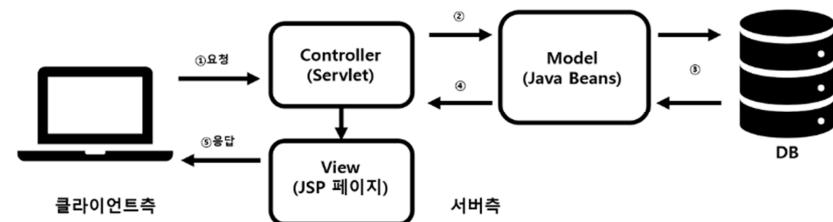


- Model: 입력 값 검증, business logic 실행, database 연동 등을 담당
- View: 사용자 요청 처리 결과에 대한 출력(presentation) 및 UI 담당
- Controller: 사용자와 Model, View 사이의 실행 흐름 제어
 - ✓ 사용자의 요청 수신, Model의 기능 선택 및 호출, View 선택 및 결과 전송 등

33

MVC 구조

◆ Request 처리 과정

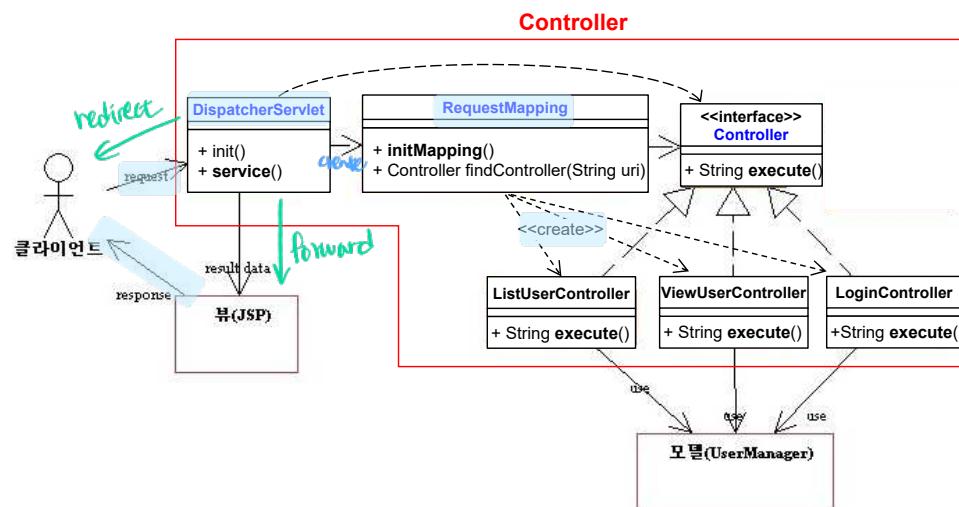


- 모든 사용자 요청은 Controller(Servlet)로 전달됨
- Controller는 사용자 요청에 대한 처리를 Model에 위임함
 - ✓ 사용자의 요청 데이터(parameter)를 전달
- Model은 요청 처리를 완료한 후 Controller에게 결과를 반환함
- Controller는 사용자에게 보여줄 View를 호출함
 - ✓ 사용자의 요청과 Model로부터 반환된 결과에 따라 적절한 view page를 선택
 - ✓ Model로부터 반환된 요청 처리 결과를 view page에 전달
- View는 결과 화면을 생성하여 사용자에게 전송함(response)

34

MVC 구조

◆ Controller 구성 요소



35

◆ DispatcherServlet class ("front controller")

```

// @WebServlet(name="dispatcherServlet", urlPatterns="/", loadOnStartup=1)
public class DispatcherServlet extends HttpServlet {
    private RequestMapping rm;
    public void init() throws ServletException {
        rm = new RequestMapping();
        rm.initMapping(); // controller 객체들 생성, request URI와 controller 간의 mapping 정의
    }
    public void service(HttpServletRequest request, HttpServletResponse response) throws ... {
        String contextPath = request.getContextPath(); // application name
        String servletPath = request.getServletPath(); // request URI

        // servletPath에 대응되는 controller를 구함
        Controller controller = rm.findController(servletPath);
        try {
            // controller를 실행하여 request를 처리한 후, 이동할 URI를 반환 받음
            String uri = controller.execute(request, response);

            // 반환된 uri의 형식에 따라 forwarding 또는 redirection 여부를 결정하고 이동
            if (uri.startsWith("redirect:")) { // redirect로 이동하면
                String targetUri = contextPath + uri.substring("redirect:".length());
                response.sendRedirect(targetUri); // redirection 응답 생성 Uri
            } else { // jsp로 처리
                RequestDispatcher rd = request.getRequestDispatcher(uri);
                rd.forward(request, response); // forwarding 실행
            }
        } catch (Exception e) { throw new ServletException(e.getMessage()); }
    }
}
  
```

DispatcherServlet은 웹 서비스로 등록되어 초기화 시 RequestMapping을 초기화합니다. service 메서드는 HttpServletRequest와 HttpServletResponse를 처리합니다. servletPath에 해당하는 controller를 찾습니다. controller는 execute() 메서드를 호출하여 request를 처리합니다. execute()는 결과 URI를 반환합니다. 결과 URI가 redirect 형식인 경우, targetUri를 설정하고 response.sendRedirect()를 호출하여 redirection 응답을 생성합니다. 결과 URI가 jsp 형식인 경우, request.getRequestDispatcher()를 사용하여 RequestDispatcher를 생성하고 forward()를 호출하여 forwarding 실행합니다.

MVC 구조

DispatcherServlet을 애플리케이션의 초기 진입점으로 설정

- web.xml 또는 @WebServlet을 통해 요청 URL과 servlet 간의 mapping 설정

```
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>controller.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

```
@WebServlet(name="dispatcherServlet", urlPatterns="/", loadOnStartup=1)
public class DispatcherServlet extends HttpServlet { ... }
```

- ✓ url-pattern "/" : (JSP 요청을 제외한) 모든 request를 이 servlet이 받음
- ✓ 예: http://localhost:8080 /UserMan2 /user/login
http://localhost:8080 /UserMan2 /user/view ?userId=scott
(context path) (servlet path) (query string/parameters)

37

■ RequestMapping class

```
public class RequestMapping {
    // 각 요청 URI에 대한 controller 객체를 저장할 HashMap 생성
    private Map<String, Controller> mappings = new HashMap<String, Controller>();

    public void initMapping() {
        // 각 URI에 대응되는 controller 객체를 생성 및 저장
        mappings.put("/", new ForwardController("index.jsp"));
        mappings.put("/user/login/form", new ForwardController("/user/loginForm.jsp"));
        mappings.put("/user/login", new LoginController());
        mappings.put("/user/logout", new LogoutController());
        mappings.put("/user/list", new ListUserController());
        mappings.put("/user/view", new ViewUserController());
        mappings.put("/user/register/form", new ForwardController("/user/registerForm.jsp"));
        mappings.put("/user/register", new RegisterUserController());
        mappings.put("/user/update/form", new UpdateUserFormController());
        mappings.put("/user/update", new UpdateUserController());
        mappings.put("/user/delete", new DeleteUserController());
    }

    public Controller findController(String uri) {
        // 주어진 URI에 대응되는 controller 객체를 찾아 반환
        return mappings.get(uri);
    }
}
```

@RequestMapping(urlPatterns="/")
public Controller findController(String uri) {
 // 주어진 URI에 대응되는 controller 객체를 찾아 반환
 return mappings.get(uri);
}

MVC 구조

Controller interface

- 모든 Controller들이 구현해야 할 execute() method를 선언한 인터페이스

```
package controller;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public interface Controller {
    public String execute(HttpServletRequest request,
                         HttpServletResponse response) throws Exception;
}
```

▪ 반환 문자열

- 사용자의 요청을 처리한 후 결과를 출력할 view(JSP)의 경로나 redirection 주소
 - "redirect:" 로 시작할 경우 → redirection 응답 생성
 - 그 다음 sub-string이 redirection URL의 일부를 나타냄
 - 예: "redirect:/user/list" → "(context path)/user/list" 로 redirection
 - 그렇지 않으면 → view page(jsp)로 forwarding 실행
 - 예: "/user/list.jsp" → "/user/list.jsp" 로 forwarding

39

MVC 구조

ForwardController

- 미리 정해진 URI로 forwarding을 실행하기 위한 controller

- 예: 로그인 화면과 같이 정적인 form을 정의한 페이지로 바로 이동

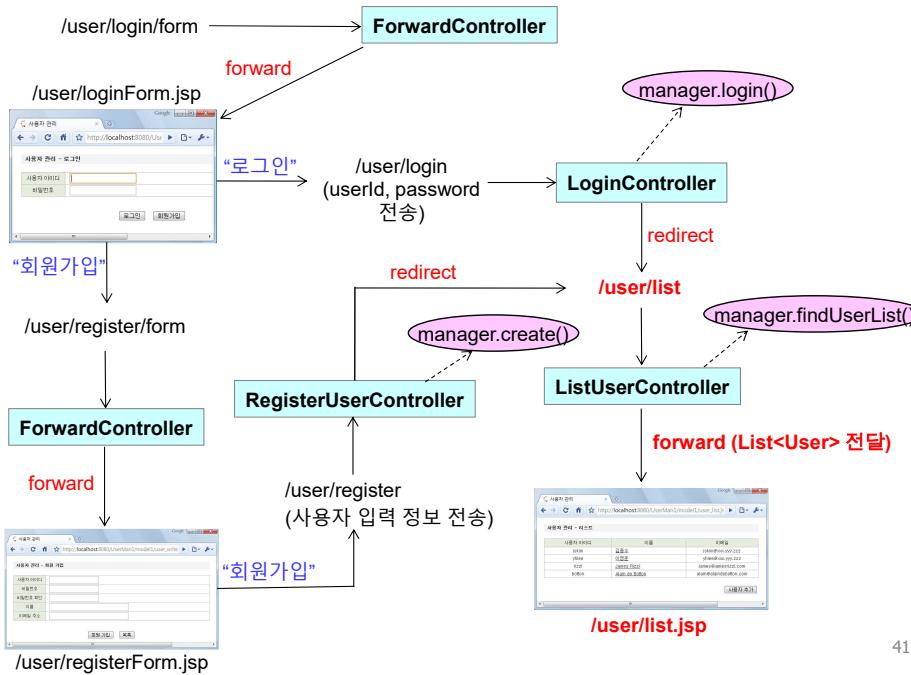
```
public class ForwardController implements Controller {
    private String forwardUrl;

    public ForwardController(String forwardUrl) {
        if (forwardUrl == null)
            throw new NullPointerException("forwardUrl is null.");
        이동할 URL을 입력하세요.";
        this.forwardUrl = forwardUrl;
    }

    @Override
    public String execute(HttpServletRequest req, HttpServletResponse resp)
        throws Exception {
        return forwardUrl;
    }
}
```

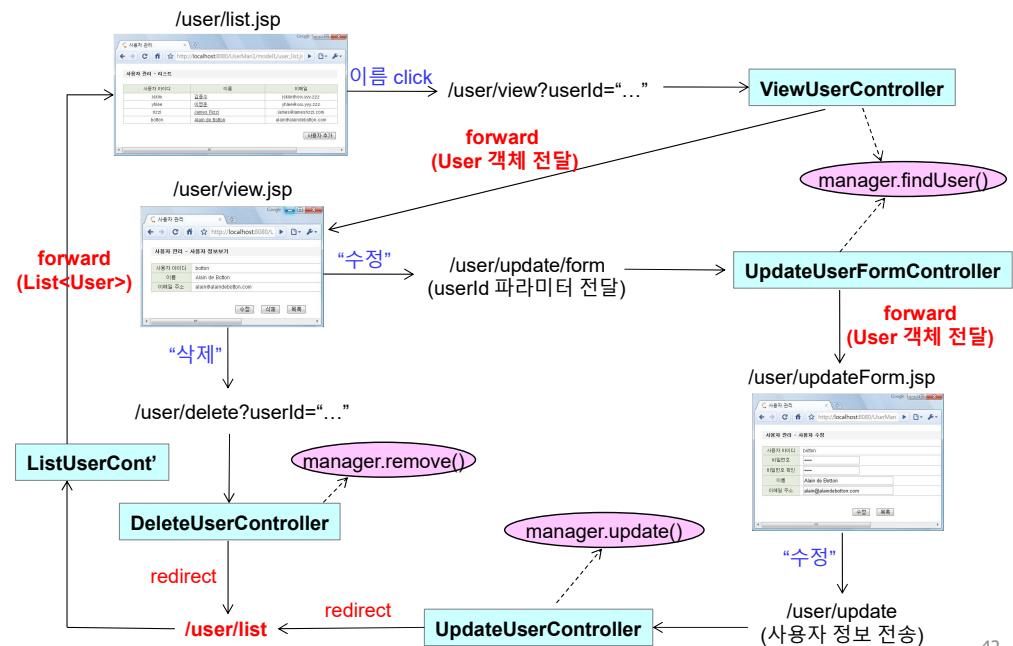
40

◆ MVC 구조 기반 Application 구현 예: Request 처리 흐름도



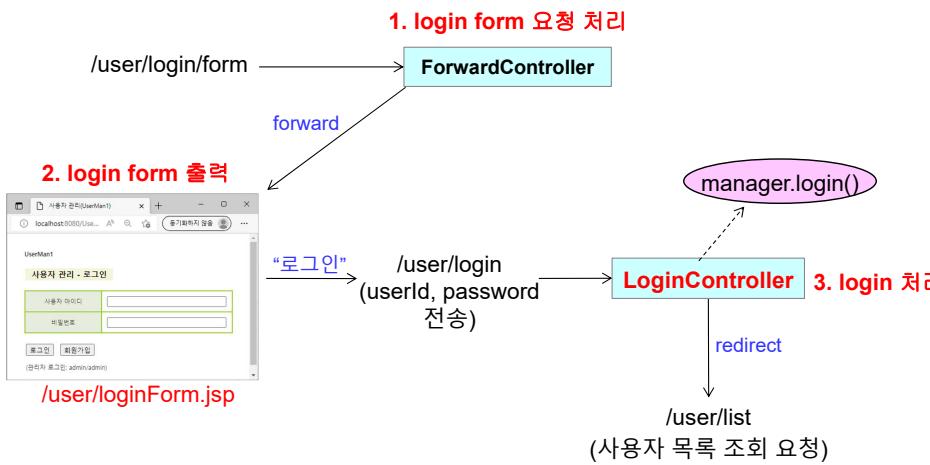
41

◆ Request 처리 흐름도(cont'd)



42

◆ 예 1: 로그인



43

■ login form 출력 (`/user/loginForm.jsp`)

```

<%@page contentType="text/html; charset=utf-8" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>사용자 관리</title>
<script language="JavaScript">
function login() {
    if (form.userId.value == "") { ... }
    if (form.password.value == "") { ... }
    form.submit();
}
function userCreate(targetUri) {
    form.action = targetUri;
    form.submit();
}
</script></head>
<body>
<b>사용자 관리 - 로그인</b>
<form name="form" method="POST" action="

```

44

■ login form 출력 (/user/loginForm.jsp)

사용자 관리 - 로그인

사용자 아이디	<input type="text"/>
비밀번호	<input type="password"/>
<input type="button" value="로그인"/>	<input type="button" value="회원가입"/>

(관리자 로그인: admin/admin)

/user/login
(userId, password 전송)

/user/register/form

■ login 처리 (LoginController class)

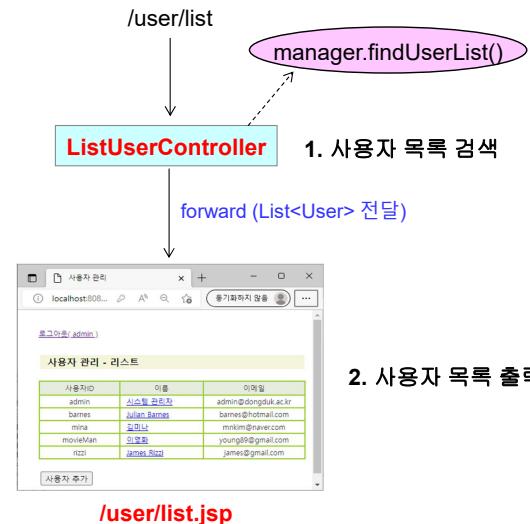
```
public class LoginController implements Controller {
    public String execute(HttpServletRequest request,
    HttpServletResponse response) throws Exception {
        String userId = request.getParameter("userId");
        String password = request.getParameter("password");
        try {
            // 모델에 login 처리를 위임
            UserManager.getInstance().login(userId, password);
            // 세션에 userId 저장
            HttpSession session = request.getSession();
            session.setAttribute("userId", userId);
        } catch (Exception e) {
            /* UserNotFoundException이나 PasswordMismatchException 발생 시
            다시 login form을 사용자에게 전송하고 오류 메세지 출력 */
            request.setAttribute("loginFailed", true); // 로그인 실패
            request.setAttribute("exception", e);
            return "/user/loginForm.jsp"; // loginForm 뷰로 forward
        }
    }
}
```

Model1:
login_action.jsp의
코드와 유사

request.setAttribute("key", value) 를 저장

45 46

◆ 예 2: 사용자 목록 조회



45

■ 사용자 목록 조회 (ListUserController class)

```
public class ListUserController implements Controller {
    public String execute(HttpServletRequest request,
    HttpServletResponse response) throws Exception {
        // 로그인 여부 확인
        if (!UserSessionUtils.isLoggedIn(request.getSession())) {
            return "redirect:/user/login/form"; // login form 요청으로 redirect
        }

        UserManager manager = UserManager.getInstance();
        List<User> userList = manager.findUserList();

        // userList 객체를 request 객체에 저장하여 뷰에 전달
        request.setAttribute("userList", userList);
        request.setAttribute("curUserId", UserSessionUtils.getLoginUserId(request.getSession())); // 현재 login한 사용자 ID

        // 사용자 목록 출력 view로 이동 (forwarding)
        return "/user/list.jsp";
    }
}
```

Model1:
user_list.jsp의
코드와 동일

47 48

■ 사용자 목록 view (/user/list.jsp)

```

<%@page import="java.util.* , user.*" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    List<User> userList = (List<User>)request.getAttribute("userList");
%>
<html>
<head><title>사용자 관리</title></head>
<body>
    <table>
        <%
            Iterator<User> userIter = userList.iterator();
            while ( userIter.hasNext() ) {
                User user = (User) userIter.next();
        %>
            (Model 1: user_list.jsp 의 내용과 유사)
            <td>
                <a href=<c:url value="/user/view">
                    <c:param name='userId' value='<%=user.getUserId()%>' />
                </c:url>><%=user.getName()%></a>
            </td>
            ...
        <% } %>
    </table>
    <a href=<c:url value='/user/register/form' />>사용자 추가</a>
</body></html>

```

49

Model1
user_list.jsp의
코드를 대체

사용자 이름에
hyperlink 생성
(URL 지정)

■ 사용자 목록 view (/user/list.jsp) – JSTL & EL 이용

```

<%@page import="java.util.* , user.*" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    List<User> userList = (List<User>)request.getAttribute("userList");
%>
<html>
<head><title>사용자 관리</title></head>
<body>
    <table>
        <%-
            Iterator<User> userIter = userList.iterator();
            while ( userIter.hasNext() ) {
                User user = (User) userIter.next();
        %>
            <c:forEach var="user" items="${userList}">
                <td>
                    <a href=<c:url value="/user/view">
                        <c:param name='userId' value='${user.userId}' />
                    </c:url>>${user.name}</a>
                </td>
                ...
            </c:forEach>
        </table>

```

생략! (불필요)

아래의 <c:forEach>로 대체

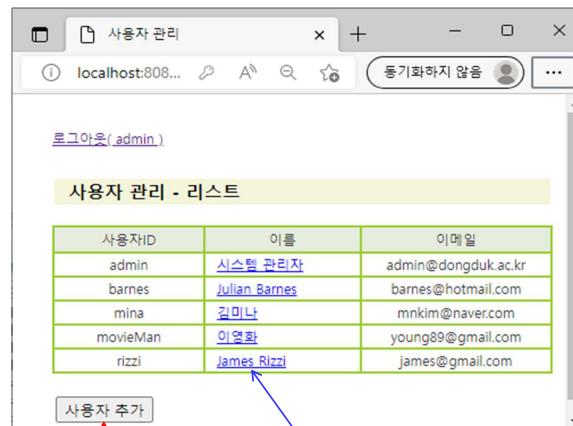
```

<td>
    <a href=<c:url value="/user/view">
        <c:param name='userId' value='${user.userId}' />
    </c:url>>${user.name}</a>
</td>
...
</table>
...

```

50

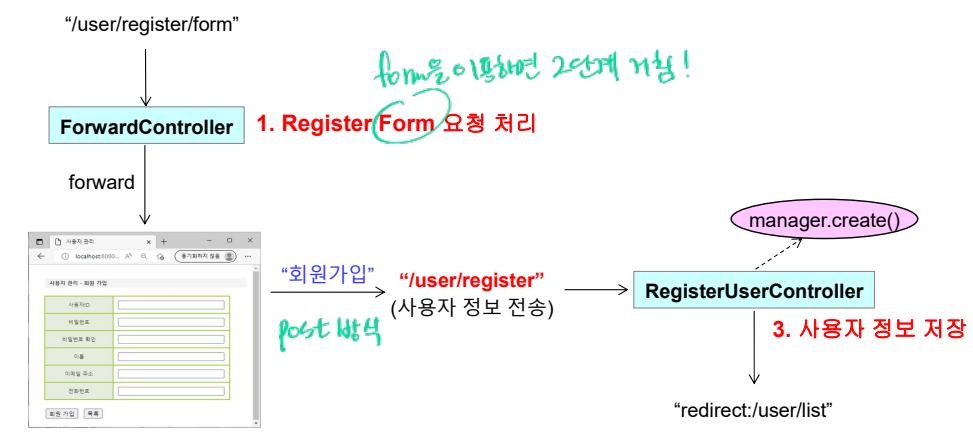
■ 사용자 목록 view (/user/list.jsp)



/user/register/form

/user/view?userId=rizzi

◆ 예 3: 회원 가입



■ 회원 가입 (RegisterUserController class)

```
public class RegisterUserController implements Controller {
    public String execute(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        User user = new User(
            request.getParameter("userId"), request.getParameter("password"),
            request.getParameter("name"), request.getParameter("email"),
            request.getParameter("phone"));
        try {
            UserManager manager = UserManager.getInstance();
            manager.create(user);
            // 성공 시 사용자 목록 요청으로 redirection
            return "redirect:/user/list";
        } catch (ExistingUserException e) {
            // 예외 발생 시 회원가입 form으로 forwarding
            request.setAttribute("registerFailed", true);
            request.setAttribute("exception", e);
            request.setAttribute("user", user);
            return "/user/registerForm.jsp";
        }
    }
}
```

53

Model1:
user_write_action.jsp
의 코드와 동일

Model1:
user_write_action.jsp
의 코드와 동일

◆ 예 4: 사용자 정보 조회



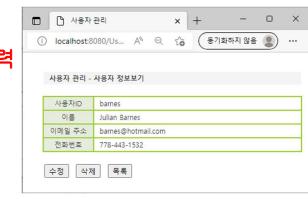
이름 click

/user/view?userId=... → ViewUserController
(userId 전송)

manager.findUser()

1. 사용자 정보 검색
forward
(User 객체 전달)

2. 사용자 정보 출력



/user/view.jsp

54

■ 사용자 정보 조회 (ViewUserController class)

```
public class ViewUserController implements Controller {
    public String execute(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        ... // 로그인 여부 확인

        User user = null;
        String userId = request.getParameter("userId");
        UserManager manager = UserManager.getInstance();
        try {
            user = manager.findUser(userId); // 사용자 정보 검색
        } catch (UserNotFoundException e) {
            return "redirect:/user/list"; // 사용자 목록 요청으로 redirection
        }

        // user 객체를 request에 저장하여 view로 전달
        request.setAttribute("user", user); // 사용자 정보 저장
        return "/user/view.jsp"; // 사용자 조회 view로 forwarding
    }
}
```

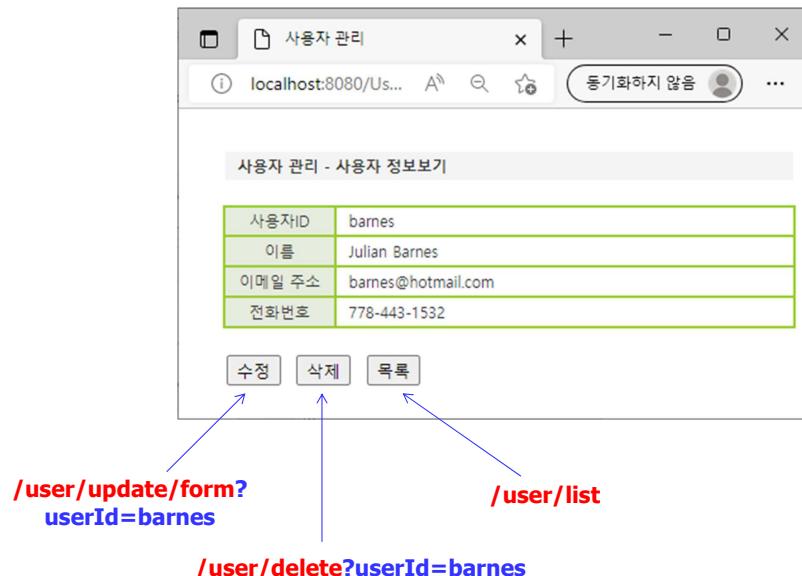
55

■ 사용자 정보 조회 view page (/user/view.jsp)

```
<%@page contentType="text/html; charset=utf-8" %> ...
<% User user = (User)request.getAttribute("user"); %> → 별로 ...
<html>
<head><title>사용자 관리</title></head>
<body>
<table>
<tr><td>사용자 아이디</td>
    <td><%= user.getUserId() %></td></tr>
<tr><td>이름</td>
    <td><%= user.getName() %></td></tr>
<tr><td>이메일 주소</td>
    <td><%= user.getEmail() %></td></tr>
<tr><td>전화번호</td>
    <td><%= user.getPhone() %></td></tr>
</table>
<br>
<a href=<c:url value='/user/update/form'>
    <c:param name='userId' value='<%=user.getUserId()%>' />
    </c:url>>수정 </a> &nbsp;
<a href=<c:url value='/user/delete'>
    <c:param name='userId' value='<%=user.getUserId()%>' />
    </c:url>>삭제 </a> &nbsp;
<a href=<c:url value='/user/list' />>목록</a>
</body></html>
```

56

- 사용자 정보 조회 view page ([/user/view.jsp](#))



57

MVC 구조

◆ 장점

- MVC(Model 2) 구조는 Model 1 구조에서 JSP가 처리해야 했던 많은 일들을 분리하여 개발 가능하도록 구성
- **개발 및 유지보수의 효율성 향상**
 - 웹 디자이너와 프로그램 개발자가 각각 UI 화면과 business logic을 개발 및 수정하는 것이 용이

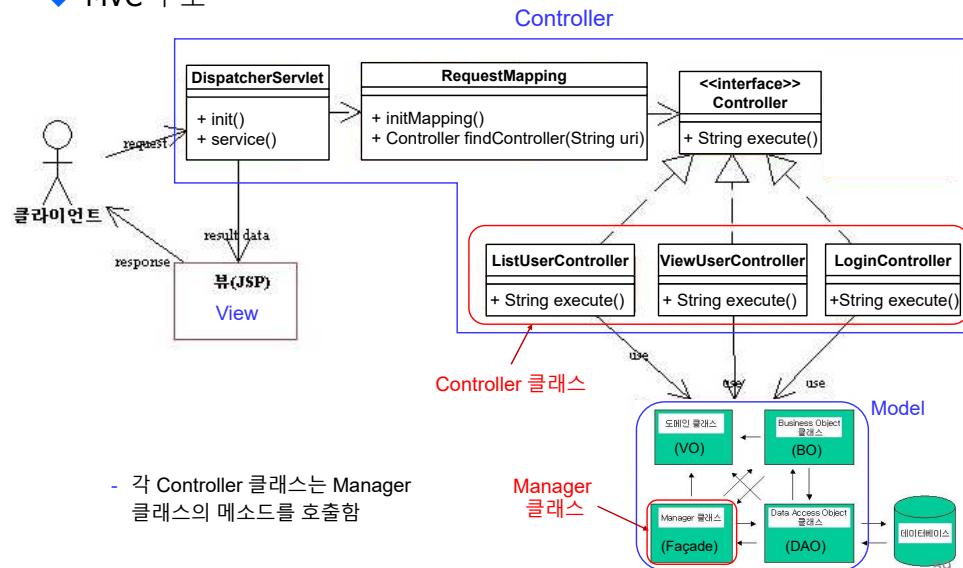
◆ 단점

- MVC(Model 2) 구조에서는 Controller 부분에 속하는 클래스들을 별도로 정의해야 함
 - Controller interface/classes, DispatcherServlet, RequestMapping 등
- Request에 대한 mapping은 애플리케이션에 종속적이므로 재사용 어려움
- 해결 방안: Spring MVC, Struts 등과 같은 Framework를 활용
 - 기본적인 Controller 구조와 DI(dependency injection) 컨테이너를 제공
 - XML/Annotation/Java code 방식의 설정을 통해 request mapping 정의 가능

58

참고: Controller와 Manager의 통합

◆ MVC 구조



■ UserManager class

```

/*
 * 사용자 관리 API를 제공하는 클래스:
 * * UserDao를 이용하여 DB와 연동하고 business object를 호출하거나 직접 business logic을 수행함
 */
public class UserManager {
    ...
    public boolean login(String userId, String password)
        throws SQLException, UserNotFoundException, PasswordMismatchException {
        User user = findUser(userId);
        if (user.matchPassword(password) == false) {
            throw new PasswordMismatchException("비밀번호가 일치하지 않습니다.");
        }
        return true;
    }

    public List findUserList() throws SQLException {
        return userDao.findUserList();
    }

    public int create(User user) throws SQLException, ExistedUserException {
        if (userDAO.existedUser(user.getUserId())) {
            throw new ExistedUserException(user.getUserId() + "는 존재하는 아이디입니다.");
        }
        return userDao.create(user);
    }
}

```

59

■ UserManager class (cont'd)

```

public int update(User user) throws SQLException {
    return userDAO.update(user);
}

public int remove(String userId) throws SQLException {
    return userDAO.remove(userId);
}

public User findUser(String userId) throws SQLException, UserNotFoundException {
    User user = userDAO.findUser(userId);
    if (user == null) {
        throw new UserNotFoundException(userId + "는 존재하지 않는 아이디입니다.");
    }
    return user;
}
...
}

```

- 위와 같이 Manager 클래스의 메소드들의 로직이 단순하고 Controller들이 같은 메소드에 대한 호출을 공유하지 않는 경우, Manager 클래스를 생략하고 Controller에서 DAO를 직접 이용 가능
- 일반적으로는 Manager 클래스를 포함한 3계층 구조를 갖는 것이 바람직함

61

■ LoginController class (기존)

```

public class LoginController implements Controller {

    public String execute(HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        String userId = request.getParameter("userId");
        String password = request.getParameter("password");
        try {
            // 모델에 login 처리를 위임: manager 호출
            UserManager.getInstance().login(userId, password);

            // 세션에 userId 저장
            HttpSession session = request.getSession();
            session.setAttribute("userId", userId);

            return "redirect:/user/list"; // 사용자 목록 요청으로 redirect
        } catch (Exception e) {
            /* UserNotFoundException이나 PasswordMismatchException 발생 시
            다시 login form을 사용자에게 전송하고 오류 메세지 출력 */
            request.setAttribute("loginFailed", true);
            request.setAttribute("exception", e);
            return "/user/loginForm.jsp"; // loginForm 뷰로 forward
        }
    }
}

```

62

■ LoginController class (변경)

```

public class LoginController implements Controller {

    public String execute(HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        String userId = request.getParameter("userId");
        String password = request.getParameter("password");
        try {
            // UserDAO를 직접 호출하여 로그인 작업 수행
            User user = UserDAO.getInstance().findUser(userId);
            if (user == null) {
                throw new UserNotFoundException(userId + "는 존재하지 않는 아이디입니다.");
            }
            if (!user.matchPassword(password)) {
                throw new PasswordMismatchException("비밀번호가 일치하지 않습니다.");
            }
            ...
        } catch (Exception e) {
            ...
        }
    }
}

```

UserDAO 클래스를 singleton pattern으로 구현

// UserDAO를 직접 호출하여 로그인 작업 수행
 User user = UserDAO.getInstance().findUser(userId);

63

■ ListUserController class (변경)

```

public class ListUserController implements Controller {

    public String execute(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        // 로그인 여부 확인
        ...

        /* UserManager manager = UserManager.getInstance();
        List userList = manager.findUserList(); */
        //↓

        // UserDAO를 직접 호출하여 사용자 목록 검색 수행
        List userList = UserDAO.getInstance().findUserList();

        // userList 객체를 request 객체에 저장하여 뷰에 전달
        ...
    }
}

```

64

Form 입력 처리 개선

- ◆ 입력 form을 이용한 데이터 입력/수정
 - 두 번의 HTTP request를 처리해야 함
 - 1st request: 입력 form 화면과 초기값을 사용자에게 출력
 - 2nd request: form을 통해 입력된 데이터를 전달받아 필요한 작업 수행
 - 예: 사용자 정보 수정
 - 기존 프로그램에서는 두 request를 각각 별도의 URI와 controller를 정의하여 처리함
 - ✓ 수정 form에 대한 request는 "/user/update/form" URI를 통해 UpdateUserController가 처리 (초기값 전송 및 form 화면 출력)
 - ✓ 수정 form에 입력된 데이터를 전송하는 두 번째 request는 "/user/update" URI를 통해 UserController가 처리 (DB 갱신)

65

Form 입력 처리 개선

- UpdateUserController class (기존)

```
public class UpdateUserController implements Controller {  
  
    public String execute(HttpServletRequest request,  
                         HttpServletResponse response) throws Exception {  
  
        String userId = request.getParameter("userId");  
        ...  
  
        UserManager manager = UserManager.getInstance();  
        User user = manager.findUser(updatedId); // DB에서 사용자 정보 검색  
        request.setAttribute("user", user); // 검색된 정보를 request에 저장  
        return "/user/updateForm.jsp"; // update form으로 전송  
    }  
}
```

66

Form 입력 처리 개선

- UpdateUserController class (기존)

```
public class UpdateUserController implements Controller {  
  
    public String execute(HttpServletRequest request,  
                         HttpServletResponse response) throws Exception {  
  
        User updateUser = new User(  
            request.getParameter("userId"),  
            request.getParameter("password"),  
            request.getParameter("name"),  
            request.getParameter("email"),  
            request.getParameter("phone"));  
  
        UserManager manager = UserManager.getInstance();  
        manager.update(updateUser); // DB에서 사용자 정보 수정  
        return "redirect:/user/list"; // 사용자 목록 요청으로 redirect  
    }  
}
```

forwarding 하면 안 되는데: 아무것도 안 보임!

67

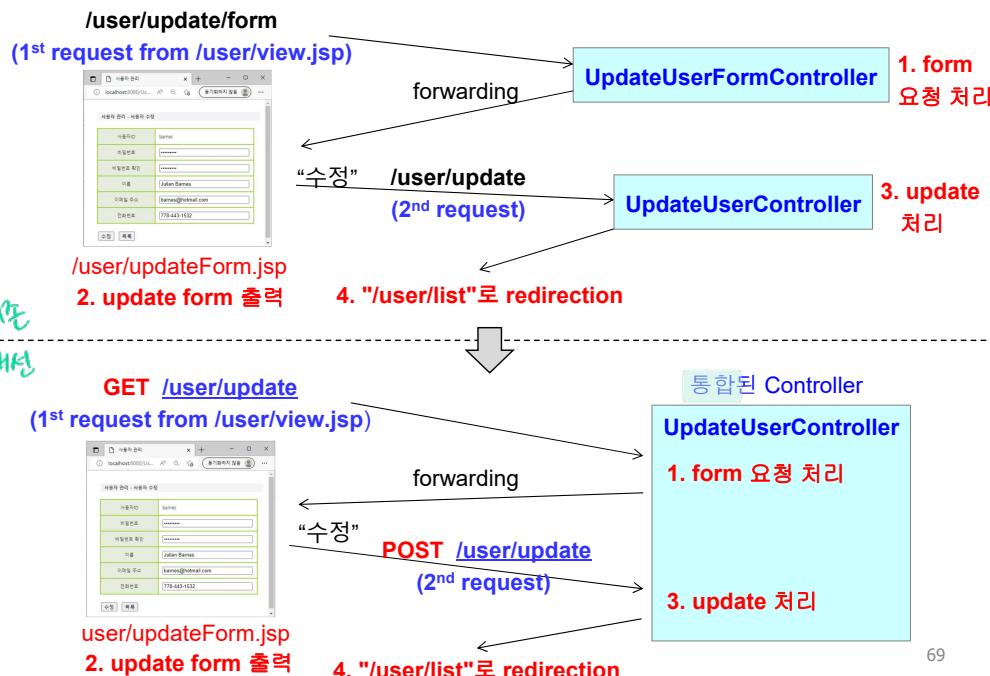
Form 입력 처리 개선

- ◆ Form Controller의 통합

- 입력 form 생성을 위한 controller와 입력 데이터 처리 controller를 하나로 통합
- 두 request가 같은 URI를 공유하되 서로 다른 HTTP Method를 이용
 - 입력 form 요청 request → GET method를 이용하도록 정의
 - Form 입력 값 전송 request → POST method로 정의
- 하나의 controller에서 request method에 따라 구별하여 처리 가능
- GET/POST request 정의 (사용자 수정 예)
 - 사용자 보기 화면(/user/view.jsp)에서 "수정" link는 GET 요청임 (만일 form을 정의하는 경우 명시적으로 method="GET"으로 지정)
 - Update form에서는 method="POST"로 설정
- 두 request가 모두 하나의 controller로 들어오도록 URI mapping 설정
- 주의: <a> 태그로 정의되는 hyperlink나 redirection을 통해 발생하는 request는 모두 GET request임

68

◆ 요청 처리 과정



■ RequestMapping class

```
public class RequestMapping {
    // 각 요청 uri에 대한 controller 객체를 저장할 HashMap 생성
    private Map<String, Controller> mappings = new HashMap<String, Controller>();

    public void initMapping() {
        // 각 uri에 대응되는 controller 객체를 생성 및 저장
        ...
        // mappings.put("/user/update/form", new UpdateUserFormController());
        // mappings.put("/user/update", new UpdateUserController());
        mappings.put("/user/update", new UpdateUserController());
        ...
    }
}
```

- 두 request를 GET/POST로 구별 가능하므로 같은 URI 사용 가능
 - Form 요청 URI도 "/user/update/form" 대신 "/user/update" 사용

70

■ UpdateUserController class (통합)

```
public class UpdateUserController implements Controller {
    public String execute(HttpServletRequest request, ... ) ... {
        UserManager manager = UserManager.getInstance();
        if(request.getMethod().equals("GET")) {
            // GET request 처리: 수정 form 생성 (UpdateUserFormController의 작업)
            String userId = request.getParameter("userId");
            ...
            User user = manager.findUser(userId); // 사용자 정보 검색
            request.setAttribute("user", user); // 검색된 정보를 request에 저장
            return "/user/updateForm.jsp"; // 검색한 사용자 정보를 form으로 전송
        }
        else // POST request 처리: 입력된 parameter 값으로 회원정보를 수정
        {
            User updateUser = new User(request.getParameter("userId"),
                request.getParameter("password"), request.getParameter("name"), ... );
            manager.update(updateUser);
            return "redirect:/user/list"; // 사용자 목록 요청으로 redirect
        }
    }
}
```

71

- /user/view.jsp: link를 통해 update form에 대한 GET 요청 정의

```
<%@page contentType="text/html; charset=utf-8" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>사용자 관리</title></head>
<body>
<table>
...
</table>
<a href=<c:url value='/user/update'>
    <c:param name='userId' value='${user.userId}'/>
</c:url>>수정</a>
<br>
</body>
</html>
```

[link 정의 → GET request 발생!](#)

72

- /user/updateForm.jsp: update form에서 **POST** 요청 생성

```

<%@page contentType="text/html; charset=utf-8" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>사용자 관리</title></head>
<body>
<form name="form" method="POST" action="
<input type="hidden" name="userId" value="${user.userId}" />
<table style="...">
    <tr height="40">
        <td>사용자 ID</td>
        <td>${user.userId}</td>
    </tr>
    <tr height="40">
        <td>비밀번호</td>
        <td>
            <input type="password" style="width: 240" name="password"
                   value="${user.password}" />
        </td>
    </tr>
    ...
</table>
<input type="submit" value="수정" />
...
</form>
</body></html>

```

POST 지정 → POST request 발생!

hidden은 사용하지
ستخدم을 이용해서

References

- 박재성, 자바 웹 프로그래밍 Next Step, 로드북, 2016.
 - 4~6장
- 조행남, JSP/Servlet 프로그래밍, 컴원미디어, 2014.
 - 16장 MVC 설계 패턴
- 김승현, 서블릿/JSP 웹 프로그래밍, 프리렉, 2012.
 - 10장 MVC 패턴
- 에릭 프리먼, 헤드 퍼스트 디자인 패턴(Head First Design Patterns), 한빛미디어, 2022.

