

여러개의 서블릿을 운영하는데 생기는 단점을 극복한 1개의 서블릿 운영에서  
그1개의 서블릿을 프론트 컨트롤러라고함. 이에 기능들이 더 추가된것이  
스프링 프레임워크이다.

KB금융그룹 | 국민의 평생  
금융파트너

스프링 구조와 같음.

스프링의 숨겨진 내부 매커니즘을 맛볼수있다.

2025년 상반기 K-디지털 트레이닝

## FrontController

보통servlet에서는 클라이언트한테 넘어온 데이터를 다루기쉽게  
모델데이터로 캡슐화해야한다. 비즈니스로직을 실행하기전에  
데이터를준비하는 단계가 있다.  
그다음엔 비즈니스로직을 처리. 이때 데이터베이스연동이 일어난다.  
그다음엔 비즈니스로직에 대한 결과를저장하고(스코프에)  
결과물이 어느 범위에 유지에 되어야하는지에 따라 저장스코프가 달라짐.

## [KB] IT's Your Life

이 단계들을 자동화하고 싶은데

- 1) 넘어온 데이터를 모델객체로 맵핑  
하는것을 자동화->원리만알면 가능
- 2)비즈니스로직은 자동화X 그때 그때 다름
- 3),4) 결과저장과 포워딩,리다이렉트는 자동화  
가능

=>

아주여러개의 서블릿을 한개의 서블릿으로 운용할수있겠다. => 비즈니스로직처리하는부분만 우리가 지정할수만 있다면,나머지는자동화  
예외처리,중앙제어처리가 쉬워짐.

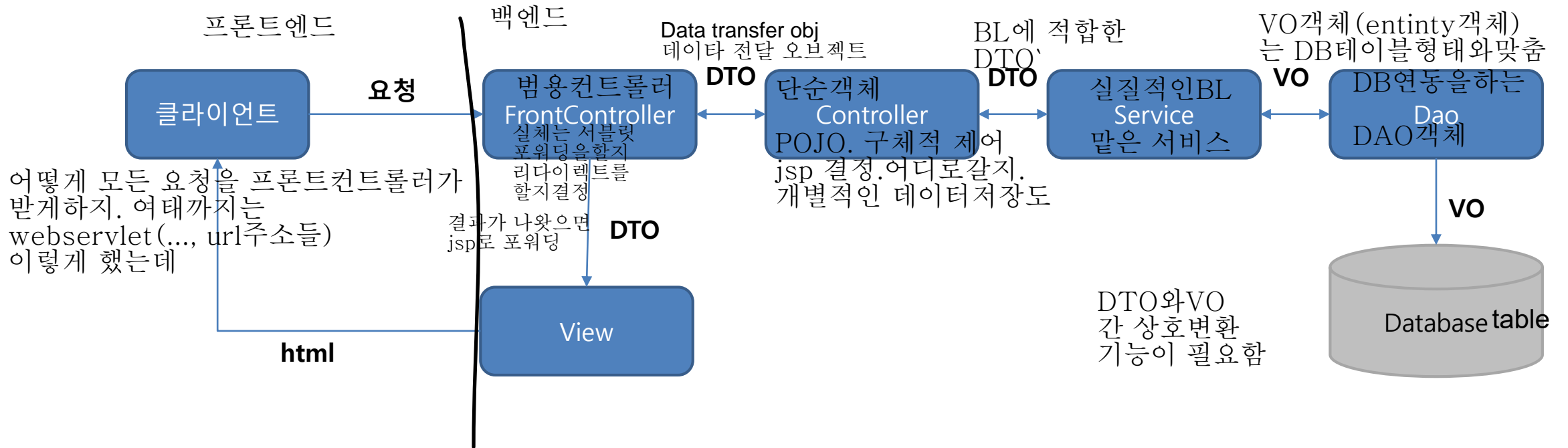
=> frontcontroller pattern==모든요청처리를 앞에서 다 하겠다. 뒷단에는 비즈니스로직처리.  
중요한 패턴 == 대부분의 웹앱 프레임워크가 채택한 패턴. mvc패턴은 유지되면서 프론트컨트롤러 개입.  
이시간에는 프론트컨트롤러 미니버전을 만들어보자. 프레임워크를 이해하기 좋음

 KB국민은행

## ✓ FrontController

- 모든 요청을 받고, 요청의 종류에 따라 작업을 분기하는 **Dispatcher Servlet**
- MVC 패턴에서 제일 앞 단에 위치

VO와 DTO를 엄연하게 구분하는 프레임워크와 아닌게있다.  
DB프레임워크를 뭘쓰냐에 따라.  
JPA (hibernate) 같은경우 전자.  
xml기반(mybatis)는 덜구분? 우리는이거사용  
엄격하게 구분지어서할예정.

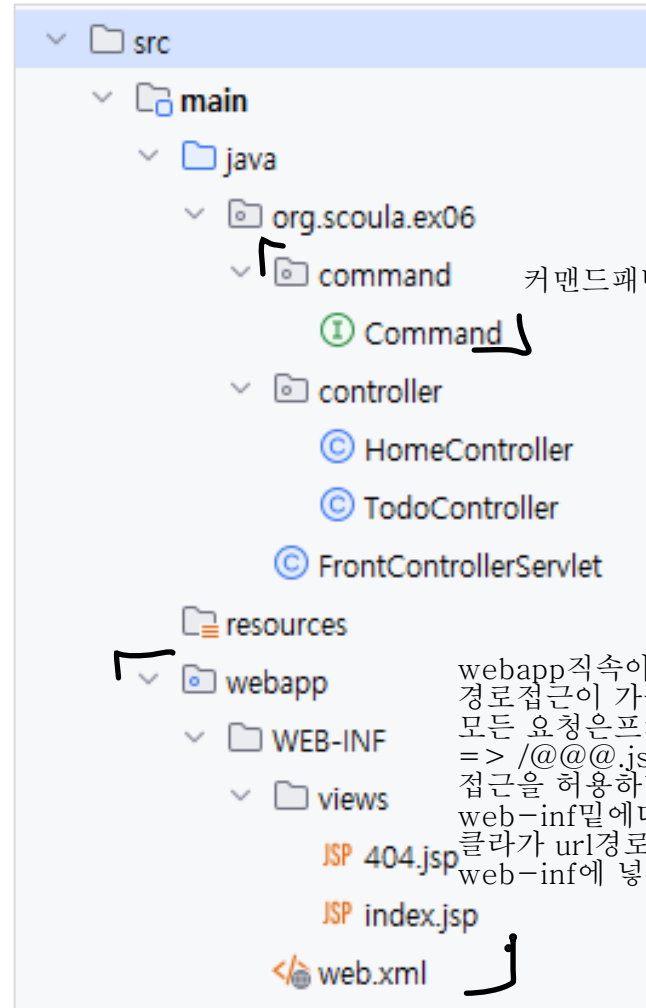


frontcontroller == dispatch servlet. spring에서는 이렇게 지칭함

## ✓ 프로젝트 만들기

### ○ 프로젝트 명: ex06

최종적으로 만들 전체 아키텍처



webapp직속아래에 jsp파일은  
경로접근이 가능 /@@@.jsp처럼.  
모든 요청은프컨이 받음  
=> /@@@.jsp처럼 직접적인  
접근을 허용하면 안돼. 프컨패턴에 위배됨.  
web-inf밑에다 넣으면 백엔드만 인식하는 공간이기에  
클라가 url경로로 접근 불가함. jsp파일들을  
web-inf에 넣을거임. lib web.xml등등을 넣을거임

클라는 /web-inf/... 접근 불가 하지만 백엔드는 가능  
즉 백엔드에서는 저 경로로 포워딩가능  
요청경로를 제한하므로써 모든 요청이 프컨에 가게끔 하는 것.

## FrontControllerServlet.java

전에 했던 url매핑은 개별적으로 한것  
디렉토리 기반으로 한것 == 지정 디렉토리 하위들은 해당서블릿이  
처리한다는뜻. 디렉토리 경로를 주면돼  
확장명 기반 == 지정 확장명은 해당 서블릿이 처리한다는뜻. 확장명을 주면돼

```
@WebServlet(name = "frontControllerServlet", value = "/")  
public class FrontControllerServlet extends HttpServlet {
```

url 매핑 (디렉토리기반) 루트경로니 모든 요청을 다 해당서블릿이 처리하겠다.

```
    @Override  
    public void init() {  
        어느요청이 왔는지 식별행해  
    }  
  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
    }  
  
    @Override  
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
        doGet(req, resp);  
    }  
}
```

## ✓ FrontController

### ○ 사용자 요청 식별 방법

#### ▪ url의 구성

- http://서버IP번호:포트번호/context명/경로(식별값)  
경로(식별값): 수행해야할 명령에 해당

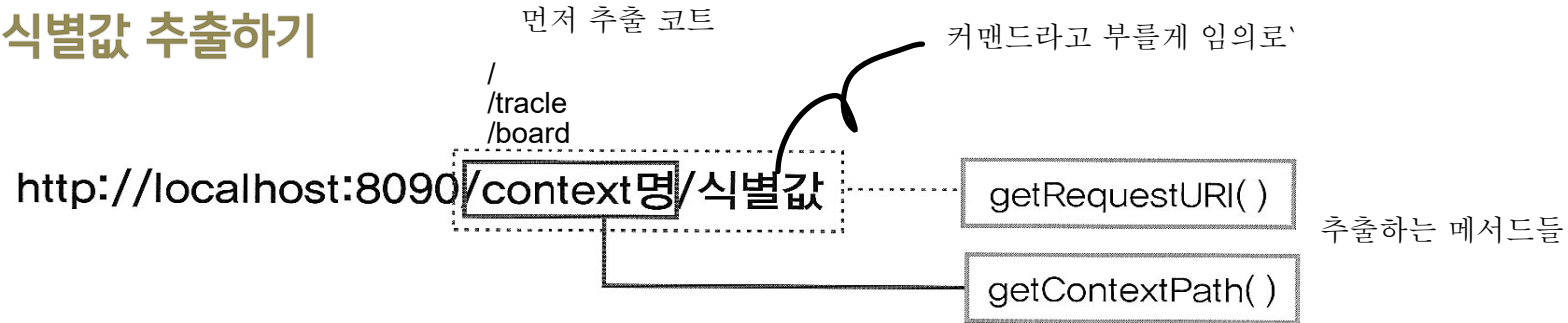
먼저 요청을 식별해야해

컨텍스트를 제외한 나머지이 파트를 추출해야함  
해당 파트에따라서 어느 비즈니스로직으로 넘어갈지 결정.

경로 파트를 해석하는 코드

## ✓ FrontController

### ○ 식별값 추출하기



```
private String getCommandName(HttpServletRequest request) {  
    String requestURI = req.getRequestURI();  
    String contextPath = req.getContextPath();  
    return requestURI.substring(contextPath.length());  
}
```

요청이 왔을때 제일 먼저 할일  
뭘 요구하는지 추출

요청을 담당하는 애한테 넘겨야함.

/getlist

/create

/update

=> 커맨드 패턴 사용.

매핑

키가 되는게 문자열임. => 어떤 컬렉션이 좋을까 인덱스도 없으니

맵으로 관리하는게 좋아. map<string, 커맨드인터페이스> 형태가 좋아

원래 이 단계에서는 클라가 보낸 데이터 추출도 해야하긴함.  
현재는 안돼. 리플렉션 기법을 사용해야함. Class객체를 이용하는것.

## ✓ FrontController

### ○ 요청별 처리 코드 찾기

- Command 패턴 적용
- Map<String, Command> ✓
  - 키: 요청 식별값
  - 값: Command 인터페이스 구현체(메서드 참조)

(  
  /create → CreateCommand  
  /get → GetCommand  
  ...

- Map에 없다면 404 에러

## Command.java

```
package org.scoula.ex06.command;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

    하나밖에 없으니 functional interface == 람다, 메서드참조 가능
public interface Command {
    String execute(HttpServletRequest req, HttpServletResponse resp) throws IOException;
}
```

커맨드클래스를 통해서 비즈니스로직 처리되니까  
다음단계인 jsp로 포워딩할거냐 리다이렉트할거냐를 정하는  
어디로

"경로" = 포워딩  
"redirect:경로" = 리다이렉트  
로 임의로 우리의 규칙



## FrontControllerServlet.java

```
@WebServlet(name = "frontControllerServlet", value = "/")
public class FrontControllerServlet extends HttpServlet {
    Map<String, Command> getMap;   맵 2개 운영. 하나는겟요청처리. 하나는 포스트요청처리.
    Map<String, Command> postMap;  ajax요청도 처리할경우 맵이 더 필요 putmap deletemap 등등.

    public void init() {
        getMap = new HashMap<>();   각 맵 초기화
        postMap = new HashMap<>();
    }
    ...
}
```

## FrontControllerServlet.java

```
@WebServlet(name = "frontControllerServlet", value = "/")
public class FrontControllerServlet extends HttpServlet {
    ...
    private String getCommandName(HttpServletRequest req) {
        String requestURI = req.getRequestURI();
        String contextPath = req.getContextPath();
        return requestURI.substring(contextPath.length());
    }

    private Command getCommand(HttpServletRequest req) {
        String commandName = getCommandName(req);

        Command command;
        if(req.getMethod().equalsIgnoreCase("GET")) {
            command = getMap.get(commandName);
        } else {
            command = postMap.get(commandName);
        }

        return command; 반환
    }
}
```

쿼리스트링이 있다면  
쿼리스트링도 추출

쿼리스트링은 없다고 가정

요청온 http메서드를 추출하는 메서드  
GET일 경우 GET맵에서 꺼내기  
아닐 경우 POST맵에서 꺼내기

없으면 null이 켜져요

## FrontControllerServlet.java

```
@WebServlet(name = "frontControllerServlet", value = "/")
public class FrontControllerServlet extends HttpServlet {
    ...

    public void execute(Command command, HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {

        String viewName = command.execute(req, resp); //b1처리하고 다음단계로 가야하는 정보를 반환
    }                                     뷰이름이 어떻게 구성됐는지에 따라 분기

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        Command command = getCommand(req);
        if(command != null) {
            execute(command, req, resp);
        } else { // 404 에러 처리
        }
    }
}
```

## ✓ 컨트롤러

- 실제 요청을 처리하고 흐름을 제어하는 역할
- 각 메서드는 Command 인터페이스와 일치하게 작성

보편적으로

### ○ GET 요청

- 리턴값은 뷰의 이름  
"/WEB-INF/views/" + 뷰이름 + ".jsp"  
prefix                      suffix → FrontController에서 처리
- 예: todo/list
  - /WEB-INF/views/todo/list.jsp
- FrontController에서 forward 처리

### ○ POST 요청

- 리턴값은 리다이렉트할 url  
"redirect:redirect url" 형식
- FrontController에서 redirect 처리

개별 컨트롤러 만들자!

## HomeController.java

```
package org.scoula.ex06.controller;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
public class HomeController {  
    public String getIndex(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
        return "index";  
    }  
}
```

POJO로 구현

커맨드 인터페이스와 맞춘 메소드

이걸 getMap 컬렉션에 추가해줘야함

view 이름: index;

→ /WEB-INF/views/index.jsp

## WEB-INF/views/index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
  <title>JSP - Hello World</title>
</head>
<body>
  <h1>FrontController</h1>
  <br/>
  <a href="/todo/list">Todo 목록보기</a>
</body>
</html>
```

변의 인텔리제이 기능.

디렉티브 입력하는거 귀찮. == 템플릿.사용.  
jsp에서 이미 템플릿을제공함.

파일>설정>데이터>파일및코드템플릿>기타탭>jsp파일>jsp\_file.jsp템플릿 수정.

jstl을 사용하기위한 디렉티브코드 넣기

gradle.bulid에 jstl임포트코드도 넣기

## FrontControllerServlet.java

```
@WebServlet(name = "frontControllerServlet", value = "/")
public class FrontControllerServlet extends HttpServlet {
    ...
    { String prefix = "/WEB-INF/views/";
      String suffix = ".jsp";

      HomeController homeController = new HomeController();

      public void init() {
          ...
          getMap.put("/", homeController::getIndex);
      }
      public void execute(Command command, HttpServletRequest req, HttpServletResponse resp)
          throws IOException, ServletException {

          String viewName = command.execute(req, resp;
          { if(viewName.startsWith("redirect:")) { // redirect 처리
            resp.sendRedirect(viewName.substring("redirect:".length()));
          } else { // forward 처리
            String view = prefix + viewName + suffix;
            RequestDispatcher dis = req.getRequestDispatcher(view);
            dis.forward(req, resp);
          }
      }
  }
```

- http://localhost:8080/

## FrontController

[Todo 목록보기](#)



## ✓ 404에러 처리

- URL 맵핑 맵에 없는 경우
  - WEB-INF/views/404.jsp로 리다이렉트

## WEB-INF/views/404.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

    <h1>404</h1>
    <h3>요청하신 페이지가 존재하지 않습니다.</h3>

</body>
</html>
```

## FrontControllerServlet.java

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {  
    Command command = getCommand(request);  
    if(command != null) {  
        execute(command, req, resp;  
    } else { // 404 에러 처리  
        String view = prefix + "404" + suffix;  
        RequestDispatcher dis = req.getRequestDispatcher(view);  
        dis.forward(req, resp);  
    }  
}
```

에러처리하고 싶다면 try문으로 묶어서 error.jsp로 가기

doPosst메서드에도 만들어

- `http://localhost:8080/abc`

**404**

요청하신 페이지가 존재하지 않습니다.

## ✓ Todo 만들기

METHOD	URL	컨트롤러 메서드	뷰 이름
GET	/todo/list	getList	todo/list 이걸 상대경로
GET	/todo/view	getView	todo/view
GET	/todo/create	getCreate	create전용페이지 { todo/create
POST	/todo/create	postCreate	create전용페이지에서 create동작요청 (폼) { redirect:_todo/list
GET	/todo/update	getUpdate	update전용페이지 { todo/update
POST	/todo/update	postUpdate	update전용페이지에서 update요청 (폼) { redirect:_todo/list
POST	/todo/delete	postdDelete	redirect:/todo/list

:/는 절대경로라는 뜻

url이 같으면 좋은점  
관리할게 url이 줄어듬  
폼에서?생략가능

## TodoController.java

```
package org.scoula.ex06.controller;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.util.Arrays;  
import java.util.List;  
  
import static sun.jvm.hotspot.code.CompressedStream.L;  
  
public class TodoController {  
    public String getList(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
        List<String> list = Arrays.asList("Todo 1", "Todo 2", "Todo 3");  
        req.setAttribute("todoList", list); 요청 스코프에 todoList라는 키로 list컬렉션 데이터 저장  
        System.out.println("GET /todo/list");  
        return "todo/list";  
    }  
  
    public String getView(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
        System.out.println("GET /todo/view");  
        return "todo/view";  
    }  
}
```

## TodoController.java

```
public String getCreate(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
    System.out.println("GET /todo/create");  
    return "todo/create";  
}  
  
public String postCreate(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
    System.out.println("POST /todo/create");  
    return "redirect:/todo/list";  
}  
  
public String getUpdate(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
    System.out.println("GET /todo/update");  
    return "todo/update";  
}  
  
public String postUpdate(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
    System.out.println("POST /todo/update");  
    return "redirect:/todo/list";  
}  
  
public String postDelete(HttpServletRequest req, HttpServletResponse resp) throws IOException {  
    System.out.println("POST /todo/delete");  
    return "redirect:/todo/list";  
}  
}
```

a 텔그는 get.

삭제 전용페이지는  
따로 없는게 보편적이죠?

목록 보여주는 페이지에서  
삭제기능을 제공하면 되죠?

## WEB-INF/views/todo/list.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Title</title>
</head>
<body>
  <h1>Todo 목록 보기</h1>
  <div>
    ${todoList} ✓ 아까 요청스코프에 저장한 거 찾아서 출력  

    <a href="view"> >상세보기</a> 벨류가 객체니까 toString메소드 호출되어 출력됨
  </div>
  <div> get
    <a href="create"> >새 Todo</a>
  </div>
</body>
</html>
```

http://localhost:8080/todo/list

## Todo 목록 보기

[Todo 1, Todo 2, Todo 3] 상세보기  
새 Todo



## WEB-INF/views/todo/view.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Title</title>
</head>
<body>
  <h1>Todo 보기</h1>

  <div>
    <a href="list">목록보기</a> |
    <a href="update">수정하기</a>
  </div>

  <form action="delete" method="POST">
    <input type="submit" value="삭제">
  </form>

</body>
</html>
```

## Todo 보기

[목록보기](#) | [수정하기](#)

삭제

## WEB-INF/views/todo/create.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Title</title>
</head>
<body>
  <h1>새 Todo 생성</h1>
  <form method="POST">
    <input type="submit">
  </form>

</body>
</html>
```

액션속성은?? 이때는 생략가능! 디폴트액션은 현재 url로 지정되

새 Todo 생성

제출


## WEB-INF/views/todo/update.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Title</title>
</head>
<body>
  <h1>Todo 수정</h1>
  <form method="POST">
    <input type="submit">
  </form>
</body>
</html>
```


Todo 수정

제출

## FrontControllerServlet.java

```
@WebServlet(name = "frontControllerServlet", value = "/")
public class FrontControllerServlet extends HttpServlet {
    ...
     TodoController todoController = new TodoController();
    ...
    public void init() {
        getMap = new HashMap<>();
        postMap = new HashMap<>();

        getMap.put("/", homeController::getIndex);

        
        getMap.put("/todo/list", todoController::getList);
        getMap.put("/todo/view", todoController::getView);
        getMap.put("/todo/create", todoController::getCreate);
        getMap.put("/todo/update", todoController::getUpdate);

        postMap.put("/todo/create", todoController::postCreate);
        postMap.put("/todo/update", todoController::postUpdate);
        postMap.put("/todo/delete", todoController::postDelete);
    }
    ...
}
```

## ✓ 또 다른 애플리케이션 제작 시 FrontController에서 수정되어야 하는 부분은 어디인가?

```
@WebServlet(name = "frontControllerServlet", value = "/")
public class FrontControllerServlet extends HttpServlet {
    ...
    HomeController homeController = new HomeController();
    TodoController todoController = new TodoController();

    String prefix = "/views/";
    String suffix = ".jsp";

    public void init() {
        getMap = new HashMap<>();
        postMap = new HashMap<>();

        getMap.put("/", homeController::getIndex);

        getMap.put("/todo/list", todoController::getList);
        getMap.put("/todo/view", todoController::getView);
        getMap.put("/todo/create", todoController::getCreate);
        getMap.put("/todo/update", todoController::getUpdate);

        postMap.put("/todo/create", todoController::postCreate);
        postMap.put("/todo/update", todoController::postUpdate);
        postMap.put("/todo/delete", todoController::postDelete);
    }
    ...
}
```

- ✓ 공통부분은 부모 클래스에서 정의
- ✓ 애플리케이션 특화 부분은 자식 클래스에서 정의

## ✓ DispatcherServlet.java (부모 클래스)

```
// @WebServlet 애너테이션 붙이지 않음
public class DispatcherServlet extends HttpServlet {
    Map<String, Command> getMap;
    Map<String, Command> postMap;

    String prefix = "/views/";
    String suffix = ".jsp";

    public void init() {
        getMap = new HashMap<>();
        postMap = new HashMap<>();
        createMap(getMap, postMap);
    }

    protected void createMap(Map<String, Command> getMap, Map<String, Command> postMap) {

    }

    ...
}
```

## ✓ FrontControllerServlet.java

상속을 안하는 방법 ?

```
@WebServlet(name = "frontControllerServlet", value = "/")
public class FrontControllerServlet extends DispatcherServlet {

    HomeController homeController = new HomeController();
    TodoController todoController = new TodoController();

    @Override
    protected void createMap(Map<String, Command> getMap, Map<String, Command> postMap) {

        getMap.put("/", homeController::getIndex);

        getMap.put("/todo/list", todoController::getList);
        getMap.put("/todo/view", todoController::getView);
        getMap.put("/todo/create", todoController::getCreate);
        getMap.put("/todo/update", todoController::getUpdate);

        postMap.put("/todo/create", todoController::postCreate);
        postMap.put("/todo/update", todoController::postUpdate);
        postMap.put("/todo/delete", todoController::postDelete);
    }
}
```