[MVC(Model View Controller) 패턴]

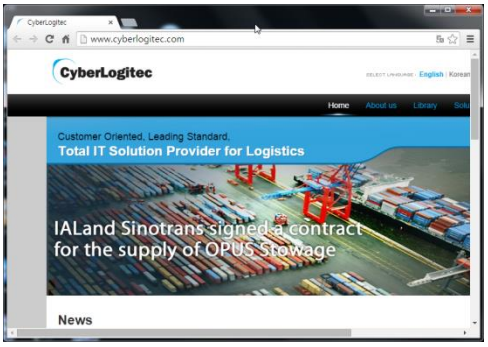
모델 : 비즈니스 영역의 상태 정보를 처리 ex) 자바빈, VO, DTO, DAO

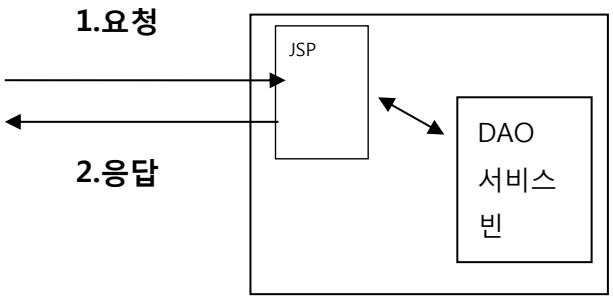
뷰 : 비즈니스 영역에 대한 presentation view를 담당 ex) jsp, html

컨트롤러 : 사용자의 입력 및 흐름 제어 담당 ex) servlet

[웹 어플리케이션의 모델1 구조]

웹브라우저(웹클라이언트) 웹서버, 웹 어플리케이션 서버(WAS)

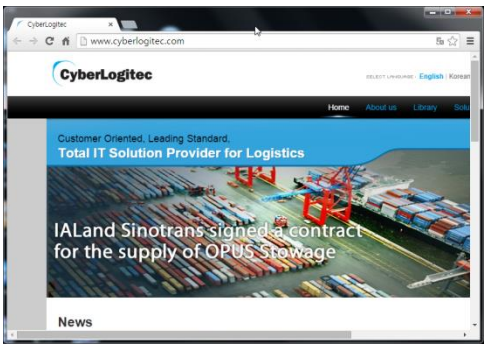


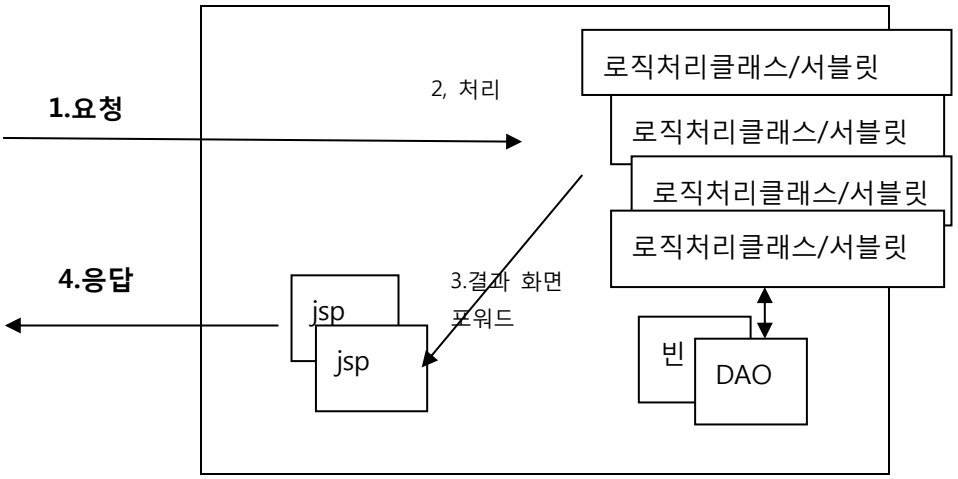


웹 클라이언트의 요청이 JSP로 전달, JSP에서 요청 작업 처리 후, 그 결과 클라이언트에 전달.(jsp에서 모든 로직 처리) 예) lab12

[웹 어플리케이션의 모델2 구조]

웹브라우저(웹클라이언트)





요청을 Controller(서블릿)가 받고 요청 처리한 후 그 결과를 보여 줄 JSP 페이지로 포워딩 한다. 즉, 비즈니스 로직을 JSP에서 처리하지 않는다. (jsp에서 결과 화면(출력)만 처리) 예) lab13

[FRONT Controller 패턴]

FrontController 패턴이란 쉽게 말해서 모든 요청을 하나의 서블릿이 받아서 요청을 처리하도록 하는 방식을 말합니다. 다른 말로는 단일 진입점 패턴이라고도 합니다.
프로젝트 이름을 fc 로 하여 프로젝트를 생성한다.

web.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
	<servlet>
		<servlet-name>action</servlet-name>
		<servlet-class>test.FrontController</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>action</servlet-name>
		<url-pattern>*.test</url-pattern>
	</servlet-mapping>
</web-app>
```

또는 다음과 같이 @WebServlet("*.test") 작성

```
package test;
import java.io.IOException; import javax.servlet.*;
import javax.servlet.http.*;

@WebServlet("*.test")
public class FrontController extends HttpServlet {

 protected void service(HttpServletRequest req,
 HttpServletResponse res throws ServletException,IOException{
		System.out.println("client 의 요청");
		System.out.println(req.getRequestURI());
		System.out.println(req.getRequestURL());
		System.out.println(req.getContextPath());
	}
}
```

http://localhost:8088/컨텍스트패스/test.test 와

http://localhost:8088/컨텍스트패스/list.test 를 실행

```
Console
Tomcat v8.5 Server at localhost [Apache Tomcat] C:₩Program Files₩
client 의 요청
/fc/test.test
http://localhost:8090/fc/test.test
/fc
client 의 요청
/fc/list.test
http://localhost:8090/fc/list.test
/fc
```

[Command 패턴]

Command 패턴이란 명령을 객체화 시켜서 처리하는 방식을 말합니다. 혹은 웹의 경우 사용자의 요청을 하나의 명령으로 보고 이를 처리하는 클래스를 별도로 작성하는 방법을 의미하기도 합니다.

```
public abstract class Command {
      public abstract void run();
}
```

```
public class ACommand extends Command {
      public void run() {
            System.out.println("A 업무를 수행합니다.");
      }
}
```

```
public class BCommand extends Command {
      public void run() {
            System.out.println("B 업무를 수행합니다.");
      }
}
```

```
public class CommandFactory {
      public static Command getCommand(String cmd){
            if("A".equals(cmd)){
                  return new ACommand();
            }else if("B".equals(cmd))
                  return new BCommand();
            else return null;
      }
   }
```

```
public class Test {
      public static void main(String args[]){
            Command c = CommandFactory.getCommand("A") ;
            c.run();
            Command c2 = CommandFactory.getCommand("B") ;
            c2.run();
      }
   }
```