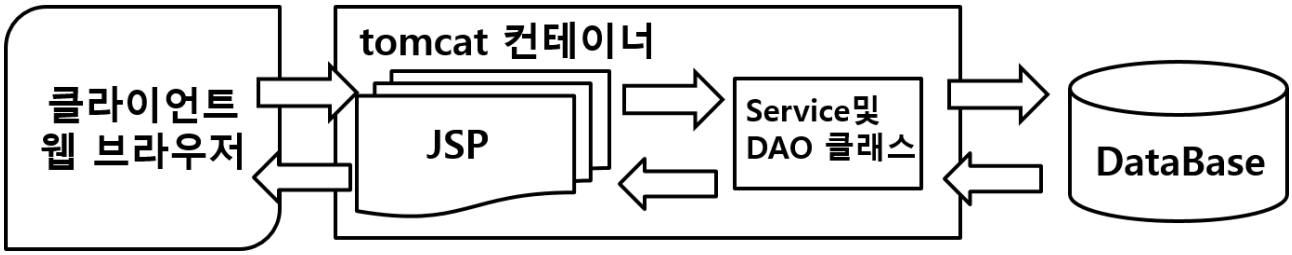
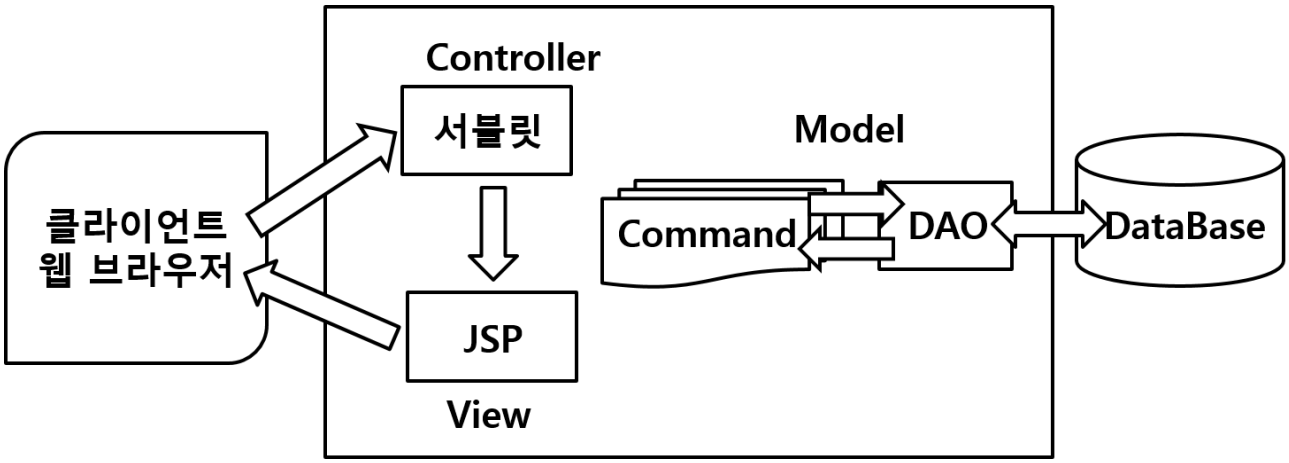
MVC 패턴은 애플리케이션을 세가지 영역, 즉 **모델(Model), 뷰(View), 컨트롤러(Controller)**로 구분하여 작업을 분리함으로써, 서로 간의 결합도를 최소화하고 유지보수도 높이며, 개발자들이 각각 맡은 영역에만 집중할 수 있게 하여 개발의 효율성을 극대화할 수 있는 장점이 있다.

뷰는 클라이언트와 서버 간의 인터페이스 역할을 하는 영역으로써 클라이언트로부터 요청받거나 처리된 결과를 보여주는 기능을 한다. 컨트롤러는 뷰와 모델을 연결하는 중계 역할을 하며, 클라이언트가 전달한 파라미터를 추출하여 모델로 전달하고, 처리 결과 페이지를 보여주는 기능을 한다. 모델은 서비스와 데이터베이스 처리를 담당하는 역할을 하며, 각 로직처리, DB 질의 처리 기능을 한다. 뷰는 HTML, CSS, JSP를 사용하여 구현하고, 컨트롤러는 JSP, 서블릿을 사용하며, 모델은 일반 평범한 자바로 구현한다.

Model 1 아키텍처의 구조



Model 2 아키텍처의 구조



1. MVC 패턴(Model-View-Controller Pattern)

MVC(Model-View-Controller) 구조는 전통적인 GUI(Graphic User Interface)기반의 애플리케이션을 구현하기 위한 디자인 패턴이다. **MVC구조는 사용자의 입력을 받아서, 그 입력 혹은 이벤트에 대한 처리를 하고 그 결과를 다시 사용자에게 표시하기 위한 최적화된 설계를 제시한다.**

뷰는 클라이언트가 보는 화면으로서 클라이언트로부터 요청이 일어나거나 처리된 결과를 보여주는 페이지이다.

컨트롤러는 뷰에서 클라이언트가 서비스를 요청했을 때 실행되는 페이지이다. 컨트롤러 페이지는 서비스를 처리하는 메서드를 호출함으로써 클라이언트 요청과 서비스 처리 객체를 연결해주는 중계 역할을

하면서 서비스 처리 흐름을 제어한다.

컨트롤러는 일반적으로 다음과 같은 기능을 처리한다

1. 뷰에서 들어온 요청을 받는다
2. 클라이언트가 전달한 파라미터를 추출한다.

```
String type = request.getParameter("type");
```

3. 파라미터의 유효성을 검사하여 유효성 검사에 실패하면 다시 뷰로 이동한다.
4. 서비스 객체의 메서드를 호출하며 파라미터를 서비스 객체로 전달한다.
5. 출력 뷰 페이지로 이동한다.

```
RequestDispatcher dispatcher = request.getRequestDispatcher("test.jsp");
dispatcher.forward(request, response);
```

RequestDispatcher 클래스는 javax.servlet 패키지에 존재하며 RequestDispatcher 클래스의 forward(request, response) 메소드는 서블릿에서 다른 리소스(Servlet, JSP page)로 요청을 보낸다. 이때 다른 리소스와 request, response 객체를 공유한다.

모델은 두가지로 구분한다. 하나는 서비스를 처리를 담당하는 Service 객체이며, 다른 하나는 데이터베이스 처리를 담당하는 DAO 객체이다.

Service 객체는 서비스를 전담 처리하기 위해 만든 객체이다. 서비스 처리를 하기 위한 내용으로만 구현된 객체를 Service 또는 Business 객체라고 한다. 모델의 하나이며 일반 자바로 작성한다. 평범한 일반 자바를 'POJO(Plain Old Java Object)'라고 부르기 때문에 Model을 개발할 때 POJO로 개발한다고 한다.

DAO 객체는 데이터베이스 처리에 관한 기능만으로 구성되는 객체를 DAO(Data Access Object) 객체하고 하며 MVC 패턴의 모델 중 하나이다. DAO 객체 역시 POJO로 작성한다.
일반적으로 DAO 클래스는 테이블 당 한 개씩 생성해서 사용한다. DAO 클래스 안에는 특정 테이블에서 수행할 작업을 메서드로 정의해서 구현한다.

※ DB와 관련된 CRUD 작업을 처리하는 클래스 (SQL 쿼리를 실행)

이름	조작	SQL
Create	생성	INSERT
Read(또는 Retrieve)	읽기(또는 인출)	SELECT
Update	갱신	UPDATE
Delete(또는 Destory)	삭제(또는 파괴)	DELETE

DTO(Data Transfer Object)는 데이터를 다른 logic에게 전송 및 반환할 때 효율적으로 데이터를 사용할 수 있게 클래스를 작성한다. 이 클래스를 DTO(Data Transfer Object) 클래스라고 한다.
DTO 클래스는 이름 그대로 데이터를 전송할 때 사용되는 클래스이며 데이터를 전송할 때와 전송된 데이터를 얻어서 사용할 때 효율적으로 사용할 수 있는 장점이 있다. 일반적으로 도메인 객체(Domain Object), VO(Value Object)라고도 한다.

2. 프런트 컨트롤러 디자인 패턴

웹 애플리케이션을 개발할 때 사용하는 MVC 디자인 패턴은 뷰에서 요청이 들어왔을 때 요청을 받아 처리하는 컨트롤러이다. 그런데 하나의 웹 애플케이션에는 많은 뷰와 많은 컨트롤러가 존재해서 각각의 뷰와 컨트롤러가 연결되어 독립적으로 실행되면, 서버 입장에서는 현재 웹 애플리케이션 실행에 대하여 일괄적으로 처리하기가 어렵다.

대표 컨트롤러를 두고 뷰에서 들어오는 모든 요청을 담당하게 하면 웹 애플리케이션을 실행하는 모든 요청을 일괄적으로 처리할 수 있다. 이러한 구조를 '프런트 컨트롤러(**Front Controller**) 디자인 패턴'이라고 한다.

프런트 컨트롤러 설정

프런트 컨트롤러는 웹 애플리케이션의 모든 요청에 우선으로 실행되어야하므로 다음 두 가지 작업이 선행되어야 한다.

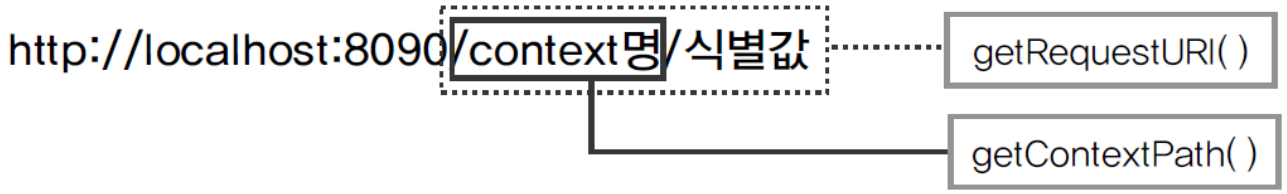
1. 클라이언트의 서비스 요청 시 요청되는 URL 패턴의 규칙을 지정한다
2. 지정된 URL의 패턴과 프론트 컨트롤러를 연결한다.

(1) URL 패턴 지정

웹 애플리케이션 개발 전에 클라이언트의 요청 URL에 요청 패턴을 지정한다. 일반적으로 요청되는 URL의 끝에 특정 단어를 붙이는 방식인데, '~.do', '~.action'과 같은 단어가 공통으로 들어가게 URL을 지정하는 것이다.

FrontController 패턴을 적용한 서블릿에서 고려해야 되는 사항은 사용자가 어떤 동작을 요청했는지를 식별할 수 있어야 된다. 따라서 **사용자가 서블릿에 요청할 때, 다음과 같은 서블릿이 사용자의 요청을 식별할 수 있도록 지원**한다.

```
http://서버IP번호:포트번호/context명/식별값
```



(2) 프런트 컨트롤러 등록

만약 모든 요청 URL 끝에 공통으로 .do가 들어오도록 구현했다면, 이 요청에 대응하여 실행할 프런트 컨트롤러 클래스를 생성한 후에 다음과 같이 web.xml에 등록하면, 프런트 컨트롤러로서 동작한다.

```
<servlet>
        <servlet-name>서블릿명</servlet-name>
        <servlet-class>Front Controller 서블릿 클래스명</servlet-class>
        <init-param><!-- -필요에 의해 설정 -->
                <param-name>초기파라미터명</param-name>
                <param-value>초기파라미터값</param-value>
        </init-param>
  </servlet>
```

```
<servlet-mapping>
        <servlet-name>서블릿명</servlet-name>
        <url-pattern>매핑명</url-pattern>
</servlet-mapping>
```

3. 서브 컨트롤러 연결

web.xml에 프런트 컨트롤러를 설정하였으므로 이후에 .do로 요청이 들어오면 프런트 컨트롤러가 실행된다. 프런트 컨트롤러에서 일괄적으로 처리할 기능을 구현하고, 처리가 완료된 후에는 반드시 실제 요청한 서비스를 처리하는 컨트롤러가 실행되게 해야한다. **프런트 컨트롤러 다음에 실제 서비스를 처리하는 컨트롤러를 서브 컨트롤러라고 표현한다.**

프런트 컨트롤러 실행이 완료된 후 서브 컨트롤러가 실행되게 하려면, 프런트 컨트롤러는 어떤 요청에 대하여 어떤 서브 컨트롤러가 실행되어야 하는지에 대한 정보를 알고 있어야 한다. 이러한 정보는 주로 Map 객체에 저장하며 저장된 정보에서 서브 컨트롤러를 찾아서 실행한다.

※ **Command 패턴**

클라이언트의 각 요청을 처리하는 별도 클래스를 제공하는 구현 패턴이다. 즉 하나의 명령어를 하나의 클래스가 처리하도록 설계하는 것이다.

3계층 아키텍처

웹 애플리케이션을 개발할 때 디자인 패턴은 MVC 디자인 패턴을 사용하며, 구조적인 측면에서 3계층 아키텍처(3-Tier Architecture)를 사용한다. 3계층 아키텍처는 다음 그림과 같이 뷰와 컨트롤러가 있는 영역을 '프레젠테이션 계층(Presentation tier)'으로 분리하고 실제 요청된 서비스를 처리하는 비즈니스 로직이 구현된 영역 '비즈니스 계층(Business tier)', 데이터베이스에 대한 처리를 하는 영역을 '영속 계층(Persistent tier)'으로 구분하여 구현하는 것을 의미한다.

3계층 아키텍처의 장점은 각 영역이 독립적으로 이루어져서 하나의 영역에서 변경이 일어났을 대 다른 영역에 변경된 내용이 영향을 미치지 않아, 서로 간의 의존성은 최소화하고 독립성은 최대화할 수 있다는 것이다.

(1) 프레젠테이션 계층(Presentation tier)

프레젠테이션 계층은 최상위에 위치하는 영역으로서 클라이언트와 애플리케이션 간에 상호 작용을 할 수 있게 하는 인터페이스 역할을 담당한다.

프레젠테이션 계층에는 비즈니스 로직이나 데이터 처리에 관한 기능을 구현하면 안되며, 단순히 웹 브라우저를 통해 클라이언트로부터 서비스를 요청받거나 비즈니스 계층의 메서드를 호출해주거나, 클라이언트가 보낸 데이터를 비즈니스 계층으로 전달하는 기능만 구현한다. 비즈니스 계층의 작업이 끝난 다음 처리 결과 뷰 페이지로 이동하는 기능도 구현한다.

프레젠테이션 계층은 '프론트 엔드(front-end)'라고도 불리며, HTML, CSS, JavaScript 등이 사용된다.

(2) 비즈니스 계층(Business tier)

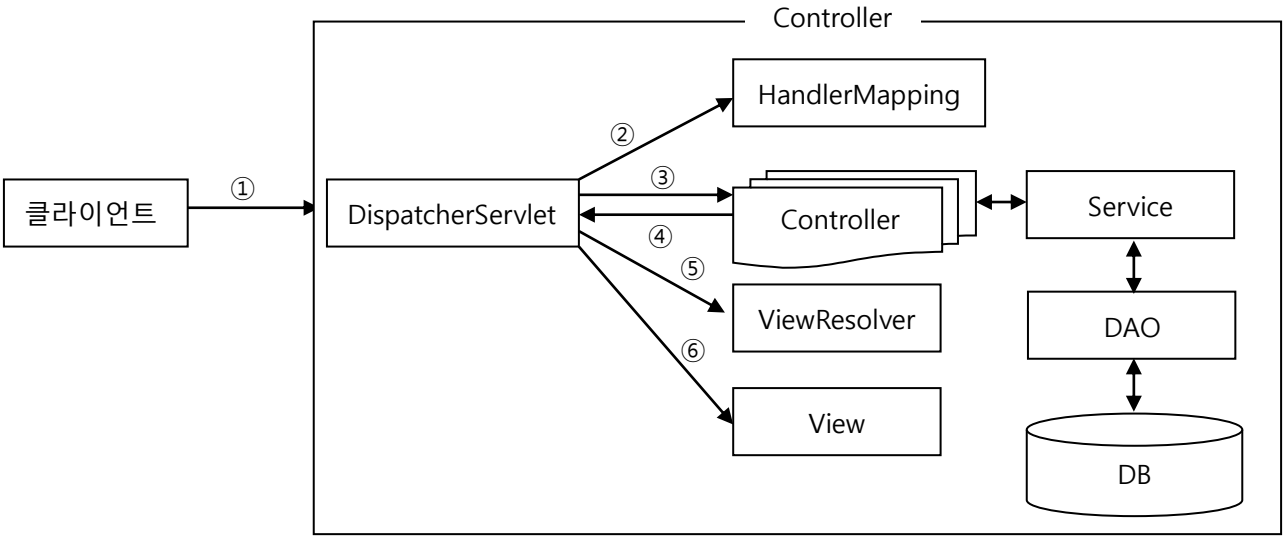
비즈니스 계층은 실제 클라이언트가 요청한 서비스를 처리하는 비즈니스 로직이 구현되는 영역이다. 서비스를 요청하는 클라이언트와 직접적으로 연결되지 않으며, 단지 프레젠테이션 계층과 연결되어 실행되는 영역이다. 전적으로 서비스 처리에 관한 기능만 구현한다. 서비스 처리를 위한 비즈니스 로직을 구현할 때 사용하는 기술은 특별한 기술을 사용하는 것이 아니고, 일반 자바 코드로 구현한다. 비즈니스 계층은 '백엔드(back-end)'라고도 부른다

(3) 영속 계층(Persistent tier)

영속 계층은 여러 가지 종류의 데이터베이스 서버나 파일 시스템에 접근하여 데이터를 생성, 관리하는 기능을 담당하는 영역이다. 서비스를 처리하면서 영구적으로 데이터를 추출, 수정, 삭제 또는 생성하는 작업이 필요하다면 영속 계층에서 처리해야 한다. 영속 계층 역시 '백엔드(back-end)'라고도 부른다. 데이터 조작에 관한 처리를 할 때 사용하는 기술은 일반 자바 코드로 구현한다.

각 클래스들의 역할을 간단하게 정리하면 다음과 같다.

클래스	기능
DispatcherServlet	유일한 서블릿 클래스로서 모든 클라이언트의 요청을 가장 먼저 처리하는 Front Controller
HandlerMapping	클라이언트의 요청을 처리할 Controller 매핑
Controller	실질적인 클라이언트 요청을 처리
ViewResolver	Controller가 리턴한 View 이름으로 실행될 JSP 경로 완성



① 클라이언트가 메뉴에서 게시판 메뉴를 클릭하면 /getBoardList.do를 요청하면 DispatcherServlet이 요청을 받는다.

② DispatcherServlet은 HandlerMapping 객체를 통해 게시판 목록 요청을 처리할 GetBoardListController를 검색하고

③ 검색된 GetBoardListController의 execute() 메서드를 호출하면 게시판 목록을 보여주기 위한 로직이 처리된다.

④ 로직 처리 후에 이동할 화면 정보를 리턴하면

⑤ DispatcherServlet은 ViewResolver를 통해 접두사와 접미사가 붙은 JSP 파일의 이름과 경로를 리턴받는다.

⑥ 그리고 최종적으로 JSP를 실행하고 실행 결과가 브라우저에 응답된다.

매핑정보(key)	서브컨트롤러(value)	뷰(화면에 보여줄 JSP 문서 및 요청정보)
/board/getBoardList.do	GetBoardListController	/WEB-INF/board/getBoardList.jsp
/board/insertForm.do	InsertFormController	/WEB-INF/board/insertForm.jsp