MVC 패턴

MVC 패턴

디자인 패턴

소프트웨어를 설계할 때 특정 맥락에서 자주 발생하는 고질적인 문제들이 또 발생했을 때 재사용할 할 수있는 훌륭한 해결책
“바퀴를 다시 발명하지 마라(Don’t reinvent the wheel)”이미 만들어져서 잘 되는 것을 처음부터 다시 만들 필요가 없다는 의미이다.

패턴이란 각기 다른 소프트웨어 모듈이나 기능을 가진 다양한 응용 소프트웨어 시스템들을 개발할 때도 서로 간에 공통되는 설계 문제가 존재하며 이를 처리하는 해결책 사이에도 공통점이 있다.

이러한 유사점을 패턴이라 한다.패턴은 공통의 언어를 만들어주며 팀원 사이의 의사 소통을 원활하게 해주는 아주 중요한 역할을 한다

MVC 패턴

디자인 패턴 종류

GoF 디자인 패턴GoF(Gang of Fout)라 불리는 사람들에리히 감마(Erich Gamma), 리차드 헬름(Richard Helm), 랄프 존슨(Ralph Johnson), 존 블리시디스(John Vissides)소프트웨어 개발 영역에서 디자인 패턴을 구체화하고 체계화한 사람들23가지의 디자인 패턴을 정리하고 각각의 디자인 패턴을 생성(Creational), 구조(Structural), 행위(Behavioral) 3가지로 분류했다.

GoF 디자인 패턴 구분

생성(Creational) 패턴	구조(Structural) 패턴	행위(Behavioral) 패턴
• 추상 팩토리(Abstract Factory) • 빌더(Builder) • 팩토리 메서드(Factory Methos) • 프로토타입(Prototype) • 싱글턴(Singleton)	• 어댑터(Adapter) • 브리지(Bridge) • 컴퍼지트(Composite) • 데커레이터(Decorator) • 퍼사드(Facade) • 플라이웨이트(Flyweight) • 프록시(Proxy)	• 책임 연쇄(Chain of Responsibility) • 커맨드(Command) • 인터프리터(Interpreter) • 이터레이터(Iterator) • 미디에이터(Mediator) • 메멘토(Memento) • 옵서버(Observer) • 테이트(State) • 스트래티지(Strategy) • 템플릿 메서드(Template Method) • 비지터(Visitor)

MVC 패턴

GoF 디자인 패턴 구분

생성(Creational) 패턴

객체 생성에 관련된 패턴
객체의 생성과 조합을 캡슐화해 특정 객체가 생성되거나 변경되어도 프로그램 구조에 영향을 크게 받지 않도록 유연성을 제공한다

구조(Structural) 패턴

클래스나 객체를 조합해 더 큰 구조를 만드는 패턴
예를 들어 서로 다른 인터페이스를 지닌 2개의 객체를 묶어 단일 인터페이스를 제공하거나 객체들을 서로 묶어 새로운 기능을 제공하는 패턴이다.

행위(Behavioral) 패턴

객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴
한 객체가 혼자 수행할 수 없는 작업을 여러 개의 객체로 어떻게 분배하는지, 또 그렇게 하면서도 객체 사이의 결합도를 최소화하는 것에 중점을 둔다.

MVC 패턴

싱글톤 패턴

전역 변수를 사용하지 않고 객체를 하나만 생성하도록 하며, 생성된 객체를 어디에서든지 참조할 수 있도록 하는 패턴

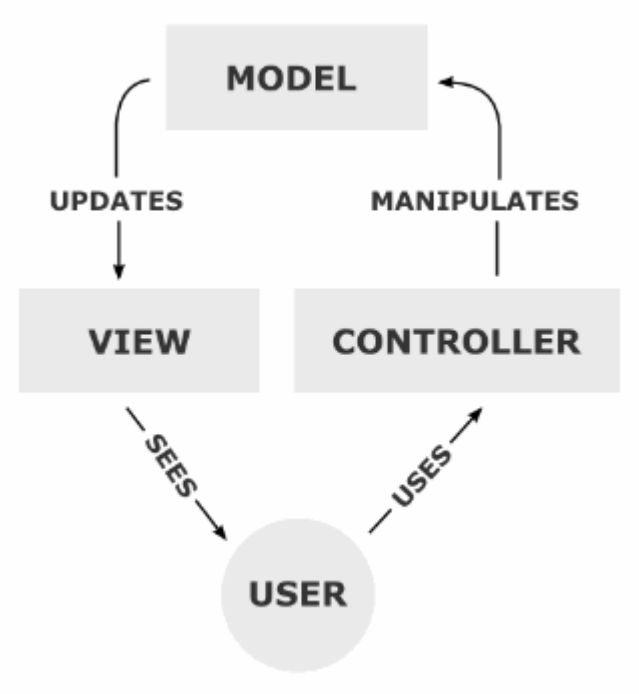
예시

```
package Ch44MVC2.Service;

import Ch44MVC2.dao.memberDAO;

public class memberservice {

        // 싱글톤 패턴
        private static memberservice instance = new memberservice();
        private memberservice() {}
        public static memberservice getInstance() {
            if(instance==null) {
                instance=new memberservice();
            }
            return instance;
        }

}
```

MVC 패턴

MVC 패턴

MVC란 **M**odel **V**iew **C**ontroller의 약자로 에플리케이션을 세가지의 역할로 구분한 개발 방법론이다. 아래의 그림처럼 사용자가 Controller를 조작하면 Controller는 Model을 통해서 데이터를 가져오고 그 정보를 바탕으로 시각적인 표현을 담당하는 View를 제어해서 사용자에게 전달하게 된다.



MVC 패턴

세부 항목

Controller
사용자가 접근 한 URL에 따라서 사용자의 요청사항을 파악한 후에 그 요청에 맞는 데이터를 Model에 의뢰하고, 데이터를 View에 반영해서 사용자에게 알려준다.

Model
일반적으로 CI의 모델은 데이터베이스 테이블에 대응된다. 이를테면 Topic이라는 테이블은 topic_model이라는 Model을 만든다. 그런데 이 관계가 강제적이지 않기 때문에 규칙을 일관성 있게 정의하는 것이 필요하다.

View
View는 클라이언트 측 기술인 html/css/javascript들을 모아둔 컨테이너이다.

MVC 패턴

WEB 과 MVC

1.사용자가 웹사이트에 접속한다. (Uses)

2.Controller는 사용자가 요청한 웹페이지를 서비스 하기 위해서 모델을 호출한다. (Manipulates)

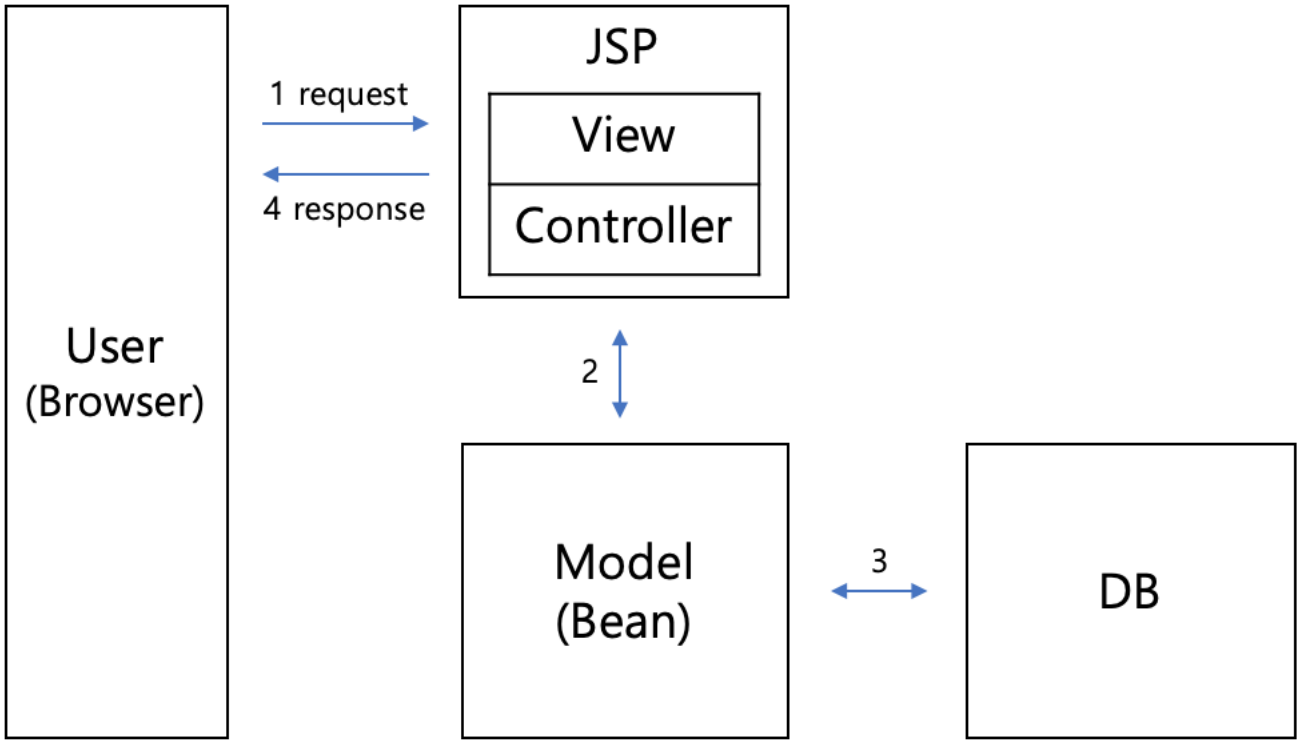
3.모델은 데이터베이스나 파일과 같은 데이터 소스를 제어한 후에 그 결과를 리턴한다.

4.Controller는 Model이 리턴한 결과를 View에 반영한다. (Updates)

5.데이터가 반영된 VIew는 사용자에게 보여진다. (Sees)

MVC 패턴

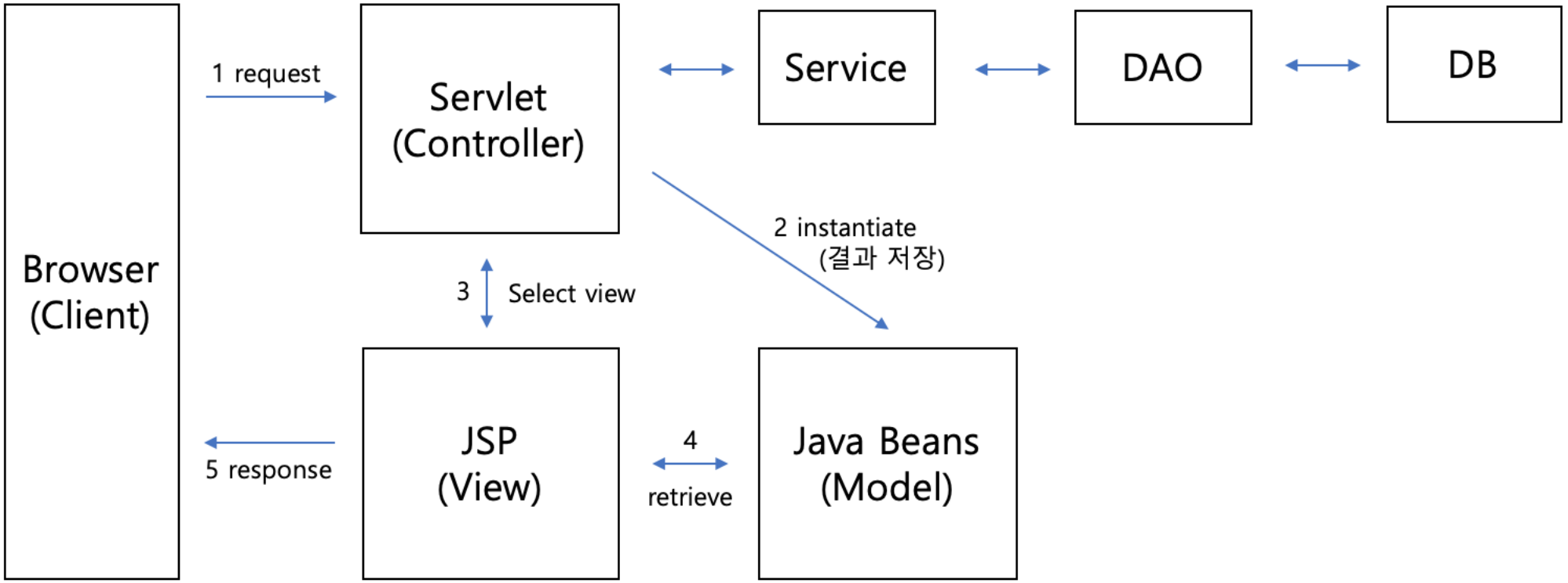
MVC1



•요청이 JSP단에서 처음 다뤄지고, 다뤄지면서 Bean하고 상호작용을 한다. DB와 상호작용하는 Bean으로 대량의 프로세싱 로직이 다뤄진다하더라도 JSP 페이지는 부분적인 프로세싱 로직을 가지고 있을 수 있다.
•JSP단에서 View와 Controller의 역할을 같이 수행, Bean은 Model의 역할을 가지고 있다.

MVC 패턴

MVC2



•들어오는 요청을 다루는 서블릿 단이 있고, 이 서블릿 단이 Controller의 역할을 수행한다.
•Controller는 다음에 어떤 작업이 될지, Model이 어떤 작업을 할지 결정하고 모든 작업 후 JSP단에 어떻게 뿌려질 지를 결정하는 것이다.
•서블릿은 요청 결과가 들어있는 Beans를 인스턴스와 한다.

MVC 패턴

MVC1

•모든 클라이언트 요청과 응답을 JSP가 담당

•장점
단순한 페이지 작성으로 쉽게 구현 가능하다.(개발이 쉬움)
구조가 간단하다. (중소형 프로젝트에 적합)

•단점
애플리케이션이 복잡해지면 개발과 유지보수가 어려워진다.
개발자와 디자이너간 역할 분담이 어려워진다.

MVC2

•클라이언트의 요청을 Servlet이 받아 Controller 역할을 수행하고 JSP가 View 역할을 수행하고 Response를 처리한다.

•장점
어플리케이션이 복잡하여도 controller와 view의 분리로 개발과 유지보수, 확장이 용이하다.

•단점
개발이 어렵다. (구조 설계를 위한 충분한 시간이 필요하며 높은 수준의 이해가 필요하다)