디자인 패턴

디자인 패턴 (Design Pattern)

프로그램 설계 시 발생햇던 문제점들을 객체 간의 상호 관계 등을 이용하여 해결 할 수 있도록 하나의 규약 형태로 만들어 놓은 것

싱글톤 패턴 (Singleton Pattern)

하나의 클래스에 오직 하나의 인스턴스만 가지는 패턴

보통 DB 연결 모듈에 많이 사용

장단점

- 장점

  하나의 인스턴스를 만들어 놓고 해당 인스턴스를 다른 모듈들이 공유하며 사용하기 때문에 인스턴스 생성 시의 비용이 감소함

- 단점

  의존성이 높아짐

TDD (Test Driven Development) 시 걸림돌이 됨

TDD 시 단위 테스트 (Unit Test) 를 주로 수행

TDD : 테스트가 서로 독립적이어야 하며 테스트를 어떤 순서로든 실행할 수 있어야함

싱글톤 패턴 : 미리 생성된 하나의 인스턴스를 기반으로 구현하는 패턴이므로 각 테스트마다 독립적인 인스턴스를 만들기가 어려움

의존성 주입

사용하기가 쉽고 굉장히 실용적이나, 모듈 간의 결합을 강하게 만들 수 있다는 단점이 존재함

의존성 주입 (Dependency Injection, DI) 을 통해 모듈 간의 결합을 느슨하게 하여 해결 할 수 있음

종속성이라고도하며, A가 B에 의존성이 있다는 것 → B의 변경 사항에 대해 A 또한 변해야 된다는 것을 의미함

메인 모듈이 직접 다른 하위 모듈에 대한 의존성을 주기보다는 중간에 의존성 주입자가 이 부분을 가로채 메인 모듈이 간접적으로 의존성을 주입하는 방식

⇒ 메인 모듈: 하위 모듈에 대한 의존성이 떨어짐 (디커플링)

- **장점**

  모듈들을 쉽게 교체할 수 있는 구조가 되어 테스팅하기 쉽고 마이그레이션하기도 수월함

  구현 시 추상화 레이어를 넣고 이를 기반으로 구현체를 넣어줌

  → 애플리케이션을 쉽게 추론할 수 있음

  → 모듈 간 관계들이 조금 더 명확해짐

- **단점**

  모듈들이 더욱 더 분리되므로 클래스 수가 늘어나 복잡성이 증가될 수 있음

  약간의 런타임 패널티가 생기기도함

- **의존성 주입 원칙**

  상위 모듈은 하위 모듈에서 어떠한 것도 가져오지 않아야함

  두 모듈 다 추상화에 의존해야함

  추상화는 세부 사항에 의존하지 말아야함

팩토리 패턴 (Factory Pattern)

객체를 사용하는 코드에서 객체 생성 부분을 떼어내 추상화한 패턴

상속 관계에 있는 두 클래스에서 상위 클래스가 중요한 뼈대를 결정하고 하위 클래스에서 객체 생성에 관한 구체적인 내용을 결정하는 패턴

상위 클래스와 하위 클래스가 분리 → 느슨한 결합을 가짐

상위 클래스는 인스턴스 생성 방식에 대해 전혀 알 필요가 없으므로 더 많은 유연성을 갖게 됨

객체 생성 로직이 따로 분리되어 있기 때문에 한 곳만 고칠 수 있으므로 유지 보수성이 증가함

전략 패턴 (Strategy Pattern)

정책 패턴 (Policy Pattern) 이라고도함

객체의 행위를 바꾸고 싶은 경우 직접 수정하지 않고 전략이라고 부르는 캡슐화한 알고리즘을 컨텍스트 안에서 바꿔주면서 상호 교체가 가능하게 만드는 패턴

옵저버 패턴 (Observer Pattern)

주체가 어떤 객체(subject)의 상태 변화를 관찰하다가 상태 변화가 있을 때마다 메소드 등을 통해 옵저버 목록에 있는 옵저버들에게 변화를 알려주는 디자인 패턴

주체와 객체를 분리하지 않고 상태가 변경되는 객체를 기반으로 구축하기도 함

대표적인 서비스 : 트위터

주체

객체의 상태 변화를 보고 있는 관찰자

옵저버

객체의 상태 변화에 따라 전달되는 메소드 등을 기반으로 추가 변화 사항이 생기는 객체들을 의미함

주로 이벤트 기반 시스템에 사용하며 MVC패턴에도 사용됨

주체라고 볼 수 있는 모델에서 변경 사항이 생겨 update() 메소드로 옵저버인 뷰에 알려주고 이를 기반으로 컨트롤러 등이 작동

프록시 패턴 (Proxy Pattern)

대상 객체(subject)에 접근하기 전 그 접근에 대한 흐름을 가로채 대상 객체 앞단의 인터페이스 역할을 하는 디자인 패턴

객체의 속성, 변환 등을 보완하며 보안, 데이터 검증, 캐싱, 로깅에 사용

프록시 객체로 쓰이기도 하나, 서버로도 활용됨

프록시 서버에서의 캐싱

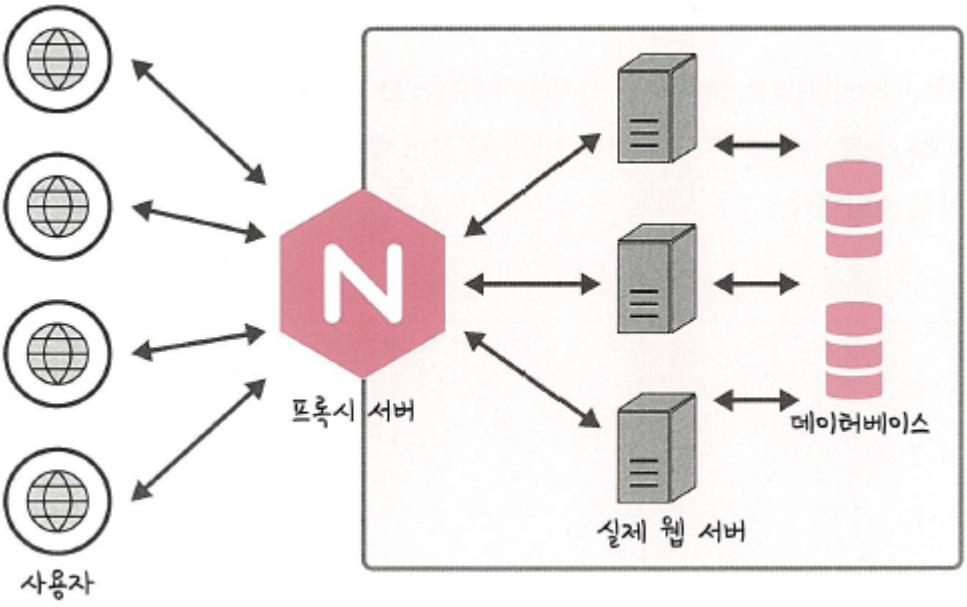
캐시 안에 정보를 담아주고, 캐시 안에 있는 정보를 요구하는 요청에 대해 원격 서버에 요청하지 않고 캐시 안에 있는 데이터를 활용하는 것

불필요하게 외부와 연결하지 않아 트래픽을 줄일 수 있다는 장점이 있음

프록시 서버 (Proxy Server)

서버와 클라이언트 사이에서 클라이언트가 자신을 통해 다른 네트워크 서비스에 간접적으로 접속할 수 있게 해주는 컴퓨터 시스템이나 응용 프로그램

Nginx

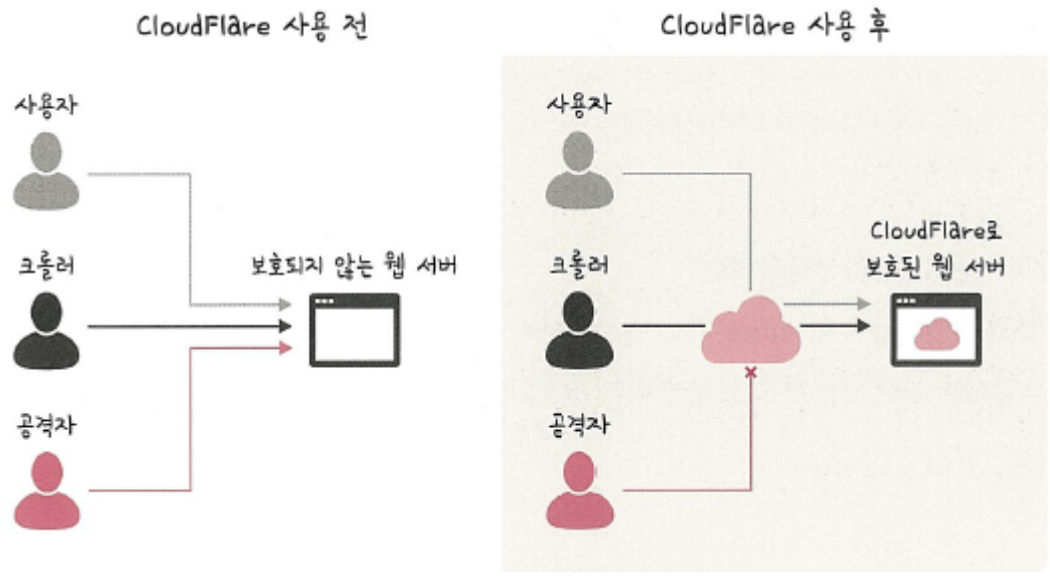


비동기 이벤트 기반의 구조와 다수의 연결을 효과적으로 처리 가능한 웹서버

익명 사용자의 직접적인 서버로의 접근을 차단하고 간접적으로 한단계를 거침으로써 보안성을 더욱 강화 할 수 있음

nginx 를 프록시 서버로 둬서 실제 포트를 숨실 수 잇꼬 정적 자원은 gzip 압축하거나 메인 서버 앞단에서의 로깅을 할 수도 있음

CloudFlare



전 세계적으로 분산된 서버가 있고 이를 통해 어떠한 시스템의 콘텐츠 전달을 빠르게 할 수 있는 CDN 서비스

- CDN (Content Devlivery Network)

  각 사용자가 인터넷에 접속하는 곳과 가까운 곳에서 콘텐츠를 캐싱하거나 배포하는 서버 네트워크. 사용자가 웹 서버로부터 콘텐츠를 다운로드하는 시간을 줄일 수 있음

DDOS 공격 방어, HTTPS 구축 등의 이점을 얻을 수 있음

웹 서버 앞단에 두어 프록시 서버로 사용하기 때문에 얻을 수 있는 이점

- DDOS 공격 방어

  DDOS : 짧은 기간 동안 네트워크에 많은 요청을 보내 네트워크를 마비시켜 웹 사이트의 가용성을 방해하는 공격 유형

  - CloudFlare

        - 의심스러운 트래픽. 특히 사용자가 아니라 시스템을 통해 오는 트래픽을 자동으로 차단함
        - 거대한 네트워크 용량과 캐싱 전략을 통해 소규모 DDOS 공격은 쉽게 막아낼 수 있으며, 이러한 공격에 대한 방화벽 대시보드도 제공함
- HTTPS 구축

  별도의 인증서 설치 없이 좀 더 손쉽게 HTTPS 구축을 할 수 있음

CORS 와 프론트엔드의 프록시 서버

CORS(Cross-Origin Resource Sharing) : 서버가 웹 브라우저에서 리소스를 로드할 때 다른 오리진을 통해 로드하지 못하게하는 HTTP 헤더 기반 메커니즘

프론트엔드 개발 시 프론트엔드 서버와 백엔드 서버 통신 시 CORS 에러가 빈번하게 발생하며, 이를 해결하기 위해 프론트엔드에서 프록시 서버를 만들기도함

이터레이터 패턴 (Iterator Pattern)

이터레이터 (Iterator) 를 사용하여 컬렉션의 요소들에 접근하는 디자인 패턴

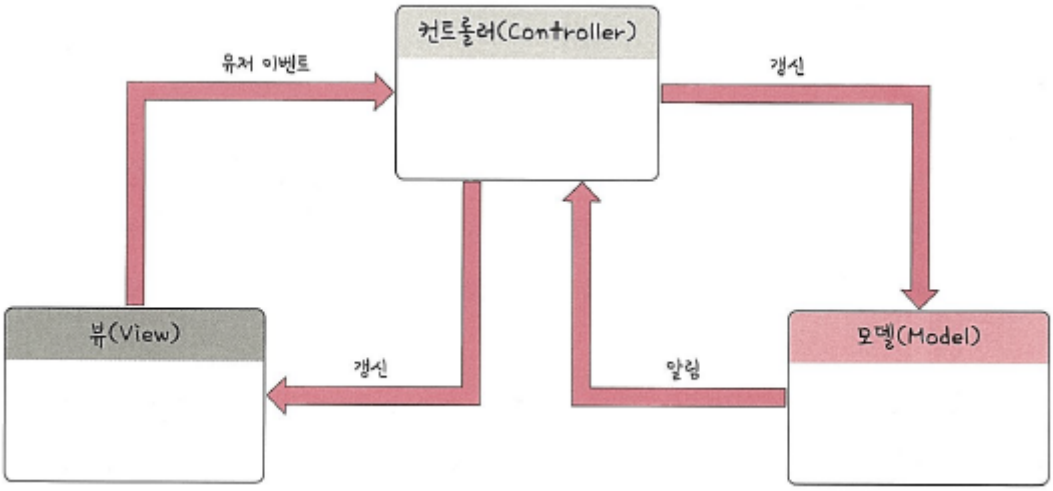
여러 가지 자료형의 구조와는 상관없이 이터레이터라는 하나의 인터페이스로 순회가 가능함

노출모듈 패턴 (Revealing Module Pattern)

즉시 실행 함수를 통해 private, public 같은 접근 제어자를 만드는 패턴

JS 에서 사용하는 것 같음

MVC 패턴



모델(Model), 뷰(View), 컨트롤러(Controller) 로 이루어진 디자인 패턴

애플리케이션의 구성 요소를 세 가지 역할로 구분하여 개발 프로세스에서 각각의 구성 요소에만 집중해서 개발할 수 있음

특징

재사용성과 확장성이 용이함

애플리케이션이 복잡해질수록 모델과 뷰의 관계가 복잡해짐

모델 (Model)

애플리케이션의 데이터인 DB, 상수, 변수 등을 뜻함

뷰에서 데이터를 생성하거나 수정하면 컨트롤러를 통해 모델을 생성하거나 갱신함

뷰 (View)

사용자 인터페이스 요소. 즉, 모델을 기반으로 사용자가 볼 수 있는 화면

모델이 가지고 있는 정보를 따로 저장하지 않아야함

화면에 표시하는 정보만 가지고 있어야함

변경이 일어나면 컨트롤러에 이를 전달해야함

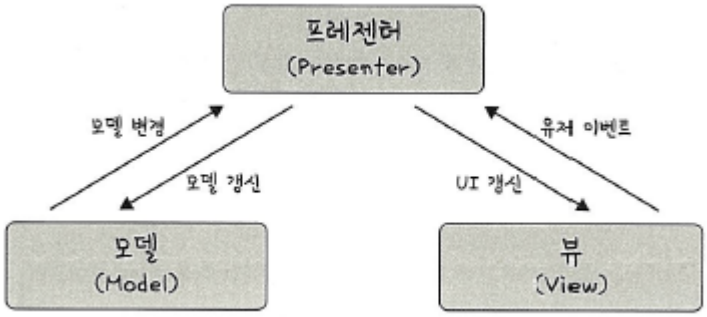
컨트롤러 (Controller)

하나 이상의 모델과 하나 이상의 뷰를 잇는 다리 역할을 하며 이벤트 등 메인 로직을 담당

모델과 뷰의 생명주기도 관리함

모델이나 뷰의 변경 통지를 받으면 이를 해석하여 각각의 구성요소에 해당 내용에 대해 알려줌

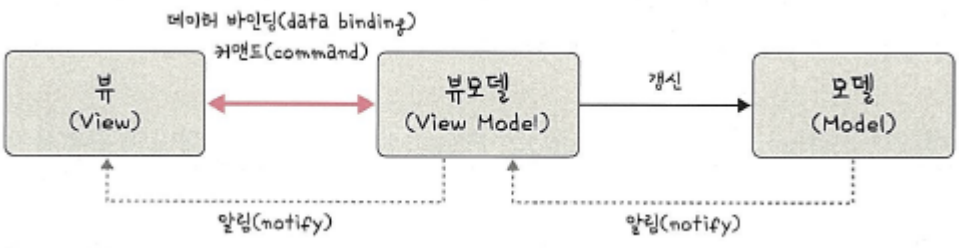
MVP 패턴



MVC 로부터 파생 되었으며 MVC 에서 C 에 해당하는 컨트롤러가 프레젠터 (Presenter)로 교체된 패턴

뷰와 프레젠터가 일대일 관계이므로 MVC 패턴보다 더 강한 결합을 지닌 디자인 패턴이라고 볼 수 있음

MVVM 패턴



MVC 에서 C 에 해당하는 컨트롤러가 뷰 모델(View Model) 로 바뀐 패턴

뷰 모델은 뷰를 더 추상화한 계층

특징

MVC 패턴과는 다르게 커맨드와 데이터 바인딩을 가짐

뷰와 뷰모델 사이의 양방향 데이터 바인딩을 지원함

UI를 별도의 코드 수정 없이 재사용할 수 있음

단위 테스트하기 쉬움

대표적인 프레임워크 : Vue.js