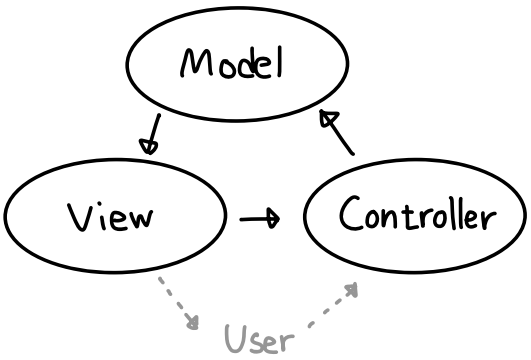
## 초기 MVC - BackEnd 위주




- MVC는 최초 BE를 위해 만들어진 패턴
- 단방향 플로우임

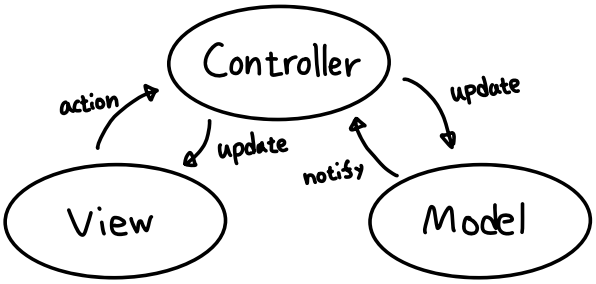
model : 데이터, 비지니스 로직

view : 유저가 보게되는 화면 UI

Controller : 입력처리. 모델 업데이트

ex) 스프링부트

## 후기 MVC - FrontEnd 위주 (이것이 MVP 이기도 하다!)



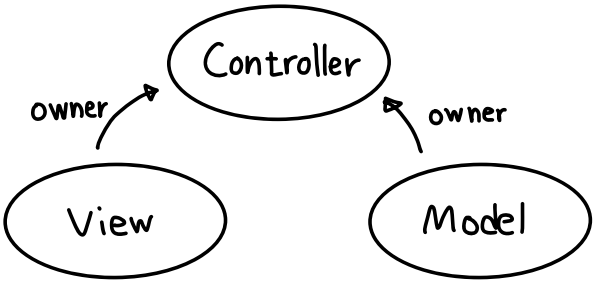

- Controller가 View와 model 사이에서 중계

model : 데이터. 데이터 관련 로직

View : UI 유저인터페이스

* Controller : 핵심 비지니스 로직.
  - View에서 온 입력 처리
  - model update
  - view update

## 실제 MVC



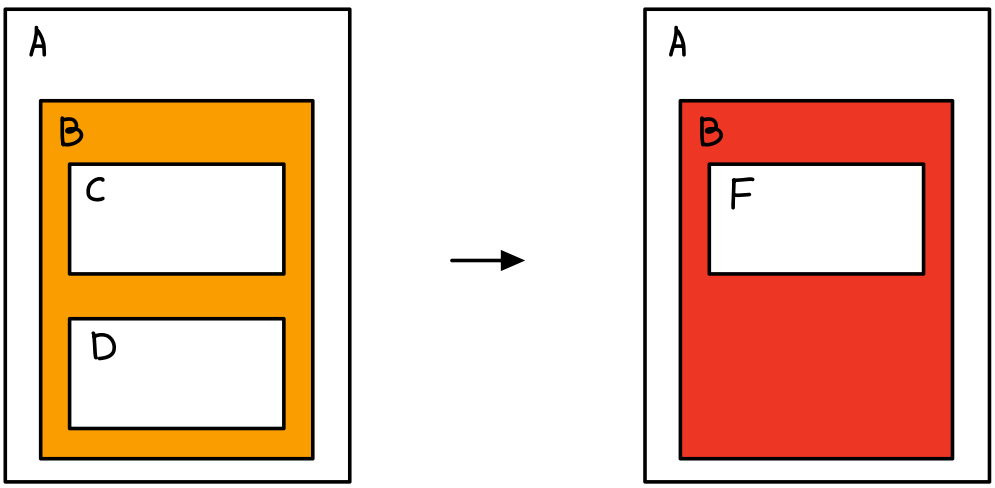

실제론 Controller가 View와 model를 모두 소유하고 컨트롤함

Controller의 코드가 엄청나게 길어진다.

* 명령적 UI 특성 때문.

(아래 그림)

ex) jQuery, Django




명령적 UI

· 화면을 직접조작

B를 선택
↓
B의 색상 변경
↓
B의 Child를 Clear (C, D를 제거)
↓
F를 만들어서 B의 자식으로 삽입

선언적 UI

· 화면을 데이터 기반으로 '새로' 그림

B의 데이터를 변경.
↓
다시 그림

```
// Imperative style
b.setColor(red)
b.clearChildren()
ViewC c3 = new ViewC(...)
b.add(c3)
```

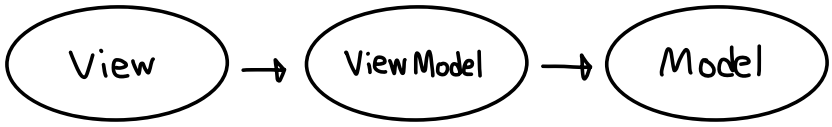
```
// Declarative style
return ViewB(
  color: red,
  child: ViewC(...),
)
```

통상적으로 MVC는 명령적 UI에서, MVVM은 선언적 UI에서 쓰임.

ex) jQuery

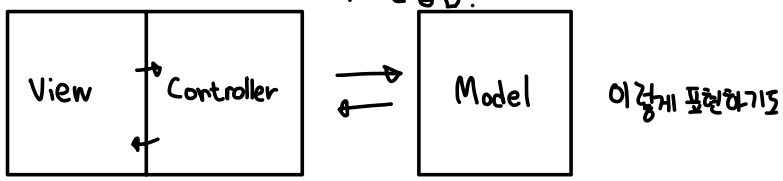
ex) Vue, React

MVVM




등장이유 1. MVC는 Controller에 대부분의 기능이 몰려 있어 코드 가독성이 떨어지고 View와 Controller가 서로 너무 밀접함.

그래서.




이렇게 표현하기도

2. 선언적 UI의 등장

- MVC와 가장 큰 차이는 View가 ViewModel을 바인딩해서 알아서 그린다는 것.
  - 이런 차이는 명령적 UI / 선언적 UI의 차이에서 기인 한다.
- 바인딩으로 인해 View는 ViewModel을 몰라도 된다. 그냥 들어오는데로 그림

Model : 데이터, (+데이터관련 로직)

View : UI (사용자 인터페이스)

ViewModel : 비지니스로직

실제 MVVM