**reference** : <https://beomy.tistory.com/43>

<https://blog.naver.com/backsajang420/221368106022>

**MVC**

1. MVC 패턴은 Model + View + Controller를 합친 용어
2. 구조

Model : 어플리케이션에서 사용되는 데이터와 그 데이터를 처리하는 부분

View : 사용자에게 보여지는 UI부분

Controller : 사용자의 입력을 받고 처리하는 부분

1. 동작
2. 사용자의 Action들은 Controller에 들어옴
3. controller는 사용자의 Action을 확인하고, Model을 업데이트함
4. Controller는 Model을 나타내줄 View를 선택함.
5. View는 Model을 이용하여 화면을 나타냄.

* MVC에서 View가 업데이트 되는 방법

view가 Model을 이용하여 직접 업데이트하는 방법

Model에서 View에게 Notify하여 업데이트를 하는 방법

View가 Polling으로 주기적으로 Model의 변경을 감지하여 업데이트하는 방법

1. 특징
   1. Controller는 여러 개의 View를 선택할 수 있는 1: n 구조이다
   2. Controller는 View를 선택할 뿐 직접업데이트하진 않음
2. 장점 : MVC 패턴의 장점은 널리사용되고 있는 패턴이라는 점에서 단순함.
3. 단점 : View와 Model 사이의 의존성이 높아서, 어플리케이션이 커지면 유지보수가 어려워진다.

**MVP**

1. Model + View + Presenter. MVC에서 Controller 대신 Presenter가 존재
2. 구조
   1. Model : 어플리케이션에서 사용되는 데이터와 그 데이터를 처리하는 부분
   2. View : 사용자에게 보여지는 UI부분
   3. Presenter : View에서 요청한 정보로 Model을 가공하여 View에 전달해줌. View와 Model을 붙여주는 접착제 역할을 함
3. 동작
   1. 사용자의 Action들은 View를 통해 들어오게 됨
   2. View는 데이터를 Presenter에 요청함
   3. Presenter는 Model에게 데이터를 요청함
   4. Model은 Presenter에서 요청받은 데이터를 응답함
   5. Presenter도 View에게 응답함
   6. View는 Presenter가 응답한 데이터를 이용하여 화면에 나타냄
4. 특징 : Presenter는 View와 Model의 인스턴스를 가지고 있어 둘을 이어줌  
    Presenter와 View는 1:1 관계이다
5. 장점 : View와 Model의 의존성이 없음.
6. 단점 : View와 Presenter 사이의 의존성이 높아짐.

**MVVM**

1. Model + View + View Model
2. 구조
   1. Model : 어플리케이션에서 사용되는 데이터와 그 데이터를 처리하는 부분
   2. View : 사용자에게 보여지는 UI부분
   3. View Model : View를 표현하기 위해 만든 View를 위한 Model.  
      View를 나타내기 위한 데이터를 처리해주는 부분.
3. 동작
   1. 사용자의 Action들은 View를 통해 들어오게 됩니다.
   2. View에 Action이 들어오면, Command 패턴으로 View Model에 Action을 전달합니다.
   3. View Model은 Model에게 데이터를 요청합니다.
   4. Model은 View Model에게 요청받은 데이터를 응답합니다.
   5. View Model은 응답 받은 데이터를 가공하여 저장합니다.
   6. View는 View Model과 Data Binding하여 화면을 나타냅니다.
4. 특징 : Command 패턴과 Data Binding 두가지 패턴을 사용. Command과 Data Binding을 통해 View와 view Model사이의 의존성을 없앰. View Model과 View는 1:n 관계이다
5. 장점 : View와 Model 사이의 의존성이 없음. View와 View Model 사이의 의존성도 없다.  
    각각이 독립적이기에 모듈활하여 개발할 수 있다
6. 단점 : 설계가 힘들다

**Flux**

1. 리덕스를 시작하기 전에 기본적으로 알아야할 디자인패턴
2. MVC를 잘못쓰면 Model과 View 사이의 양방향 데이터 흐름이 발생하여 많은 문제를 일으킬 수 있다

* 이를 극복하기 위해 단방향 데이터 흐름을 가지는 Flux를 도입함

1. 데이터 흐름 : Dispatcher -> store -> view -> action -> dispatcher
2. 구조

## Dispatcher

Dispatcher는 Flux의 모든 데이터 흐름을 관리하는 허브 역할을 하는 부분입니다. Action이 발생되면 Dispatcher로 전달되는데, Dispatcher는 전달된 Action을 보고, 등록된 콜백 함수를 실행하여 Store에 데이터를 전달합니다. Dispatcher는 전체 어플리케이션에서 한 개의 인스턴스만 사용됩니다.

## Store

어플리케이션의 모든 상태 변경은 Store에 의해 결정이 됩니다. Dispatcher로 부터 메시지를 수신 받기 위해서는 Dispatcher에 콜백 함수를 등록해야 합니다. Store가 변경되면 View에 변경되었다는 사실을 알려주게 됩니다. Store은 [싱글톤](https://ko.wikipedia.org/wiki/%EC%8B%B1%EA%B8%80%ED%84%B4_%ED%8C%A8%ED%84%B4)으로 관리됩니다.

## View

Flux의 View는 화면에 나타내는 것 뿐만이나라, 자식 View로 데이터를 흘려 보내는 뷰 컨트롤러의 역할도 함께 합니다.

## Action

Dispatcher에서 콜백 함수가 실행 되면 Store가 업데이트 되게 되는데, 이 콜백 함수를 실행 할 떼 데이터가 담겨 있는 객체가 인수로 전달 되어야 합니다. 이 전달 되는 객체를 Action이라고 하는데, Action은 대채로 액션 생성자(Action creator)에서 만들어집니다.