Vuex - **상태 관리 라이브러리**

개요

- 복잡한 애플리케이션의 컴포넌트들을 효율적으로 관리하는 Vuex 라이브러리 소개
- Vuex 라이브러리의 등장 배경인 Flux 패턴 소개
- Vuex 라이브러리의 주요 속성인 state, getters, mutations, actions 학습
- Vuex를 더 쉽게 코딩할 수 있는 방법인 Helper 기능 소개
- Vuex로 프로젝트를 구조화하는 방법과 모듈 구조화 방법 소개

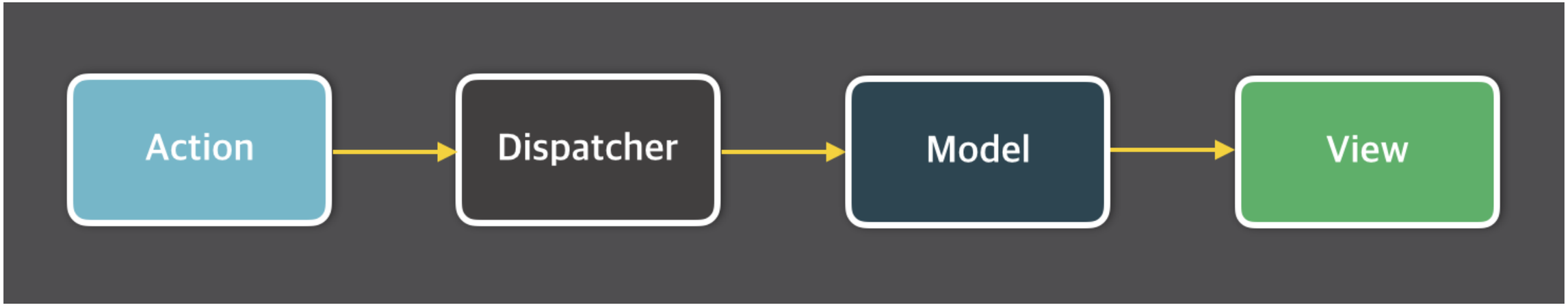
목차

Vuex란?

- 무수히 많은 컴포넌트의 데이터를 관리하기 위한 상태 관리 패턴이자 라이브러리
- React의 Flux 패턴에서 기인함
- Vue.js 중고급 개발자로 성장하기 위한 필수 관문

Flux란?

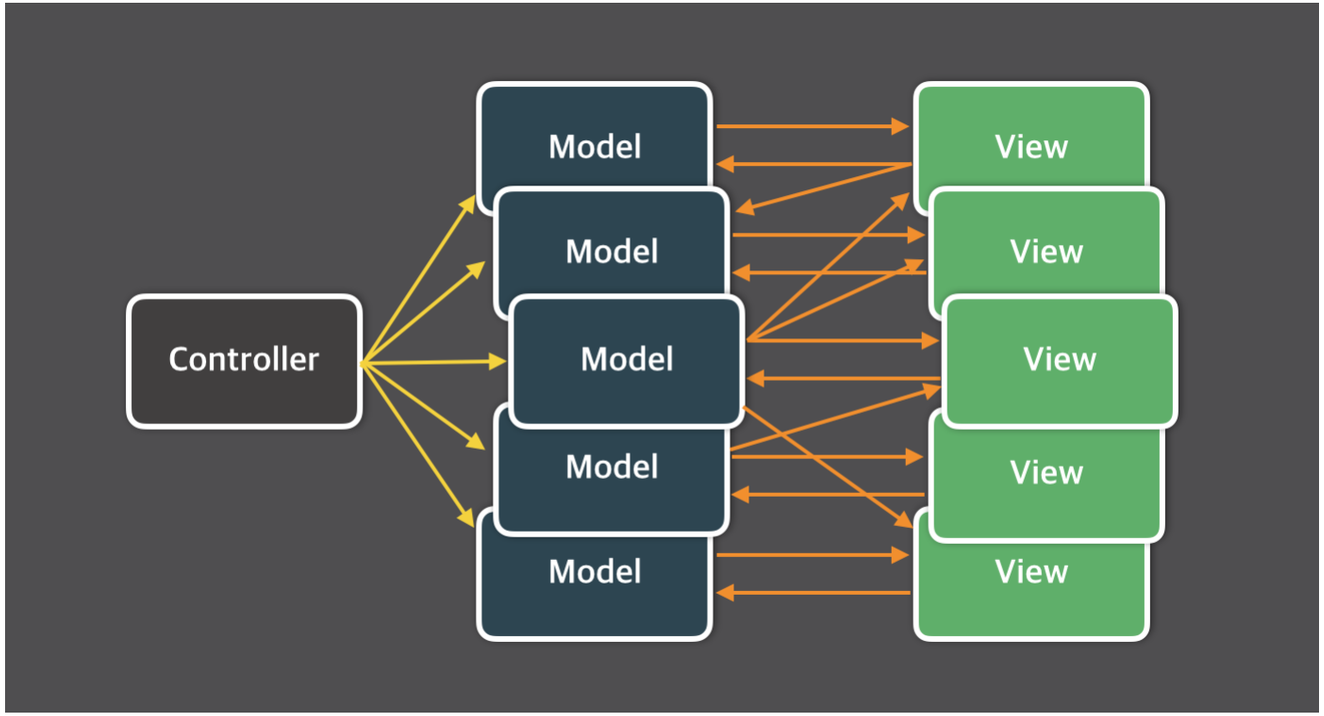
- MVC 패턴의 복잡한 데이터 흐름 문제를 해결하는 개발 패턴 - *Undirectional data flow*



1. action : 화면에서 발생하는 이벤트 또는 사용자의 입력
2. dispatcher : 데이터를 변경하는 방법, 메서드
3. model : 화면에 표시할 데이터
4. view : 사용자에게 비춰지는 화면

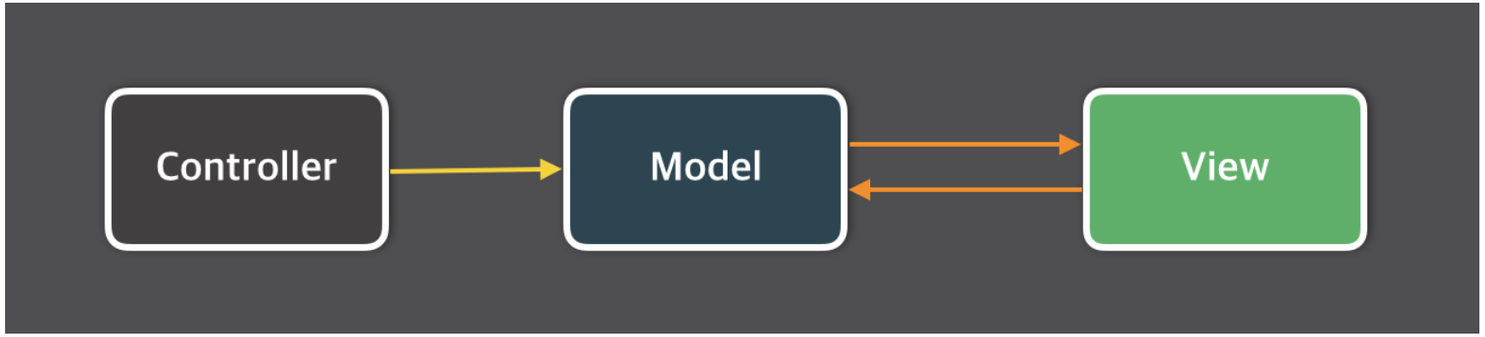
MVC 패턴의 문제점

- 기능 추가 및 변경에 따라 생기는 문제점을 예측할 수가 없음. (예) 페이스북 채팅 화면
- 앱이 복잡해지면서 생기는 업데이트 루프

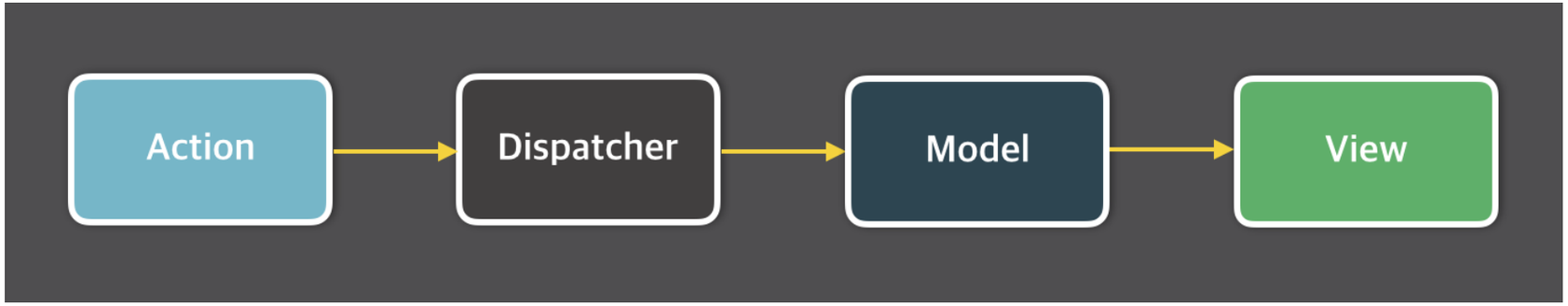


MVC 패턴과 Flux 패턴 비교

1. MVC 패턴

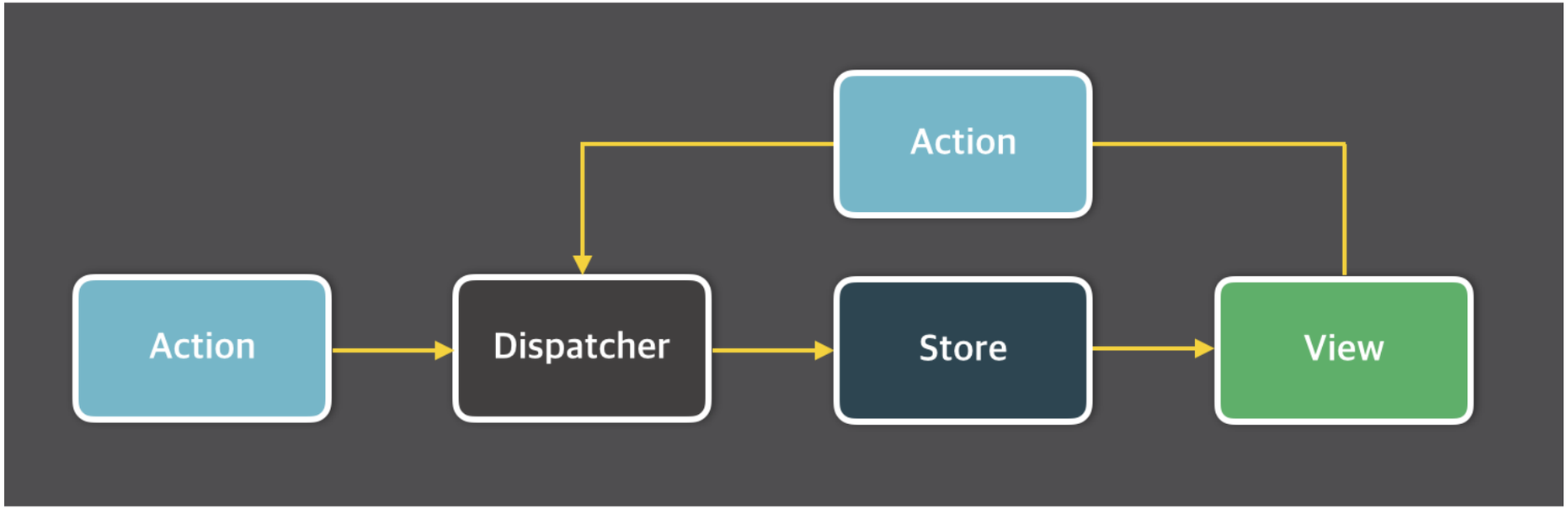


2. Flux 패턴



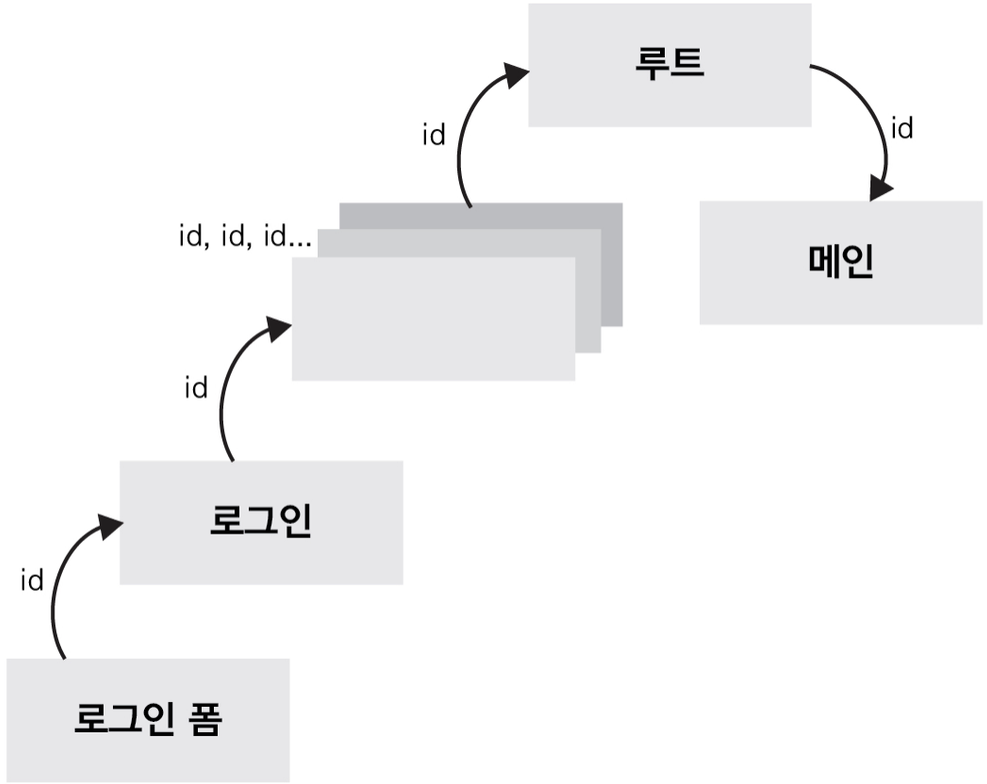
Flux 패턴의 단방향 데이터 흐름

- 데이터의 흐름이 여러 갈래로 나뉘지 않고 단방향으로만 처리



Vuex가 왜 필요할까?

- 복잡한 애플리케이션에서 컴포넌트의 개수가 많아지면 컴포넌트 간에 데이터 전달이 어려워진다.



이벤트 버스로 해결?

- 어디서 이벤트를 보냈는지 혹은 어디서 이벤트를 받았는지 알기 어려움

```
// Login.vue
eventBus.$emit('fetch', loginInfo);

// List.vue
eventBus.$on('display', data => this.displayOnScreen(data));

// Chart.vue
eventBus.$emit('refreshData', chartData);
```

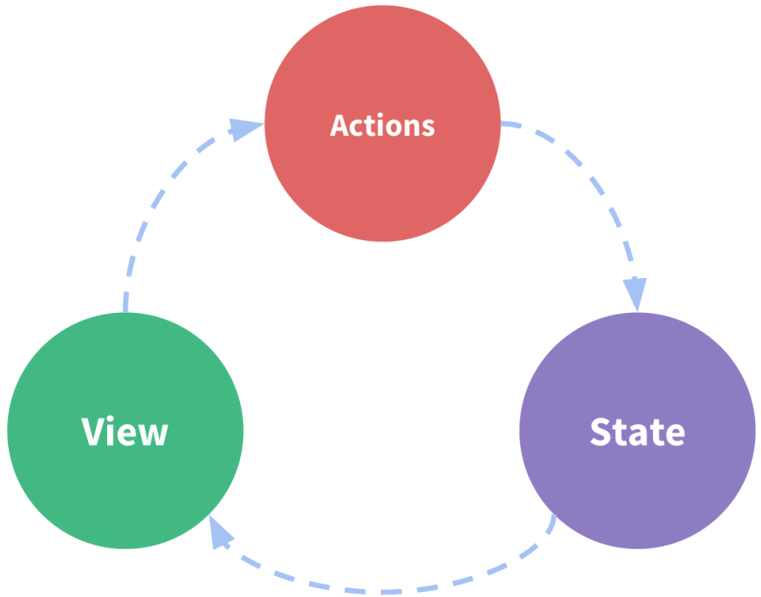
컴포넌트 간 데이터 전달이 명시적이지 않음

Vuex로 해결할 수 있는 문제

1. MVC 패턴에서 발생하는 구조적 오류
2. 컴포넌트 간 데이터 전달 명시
3. 여러 개의 컴포넌트에서 같은 데이터를 업데이트 할 때 동기화 문제

Vuex 컨셉

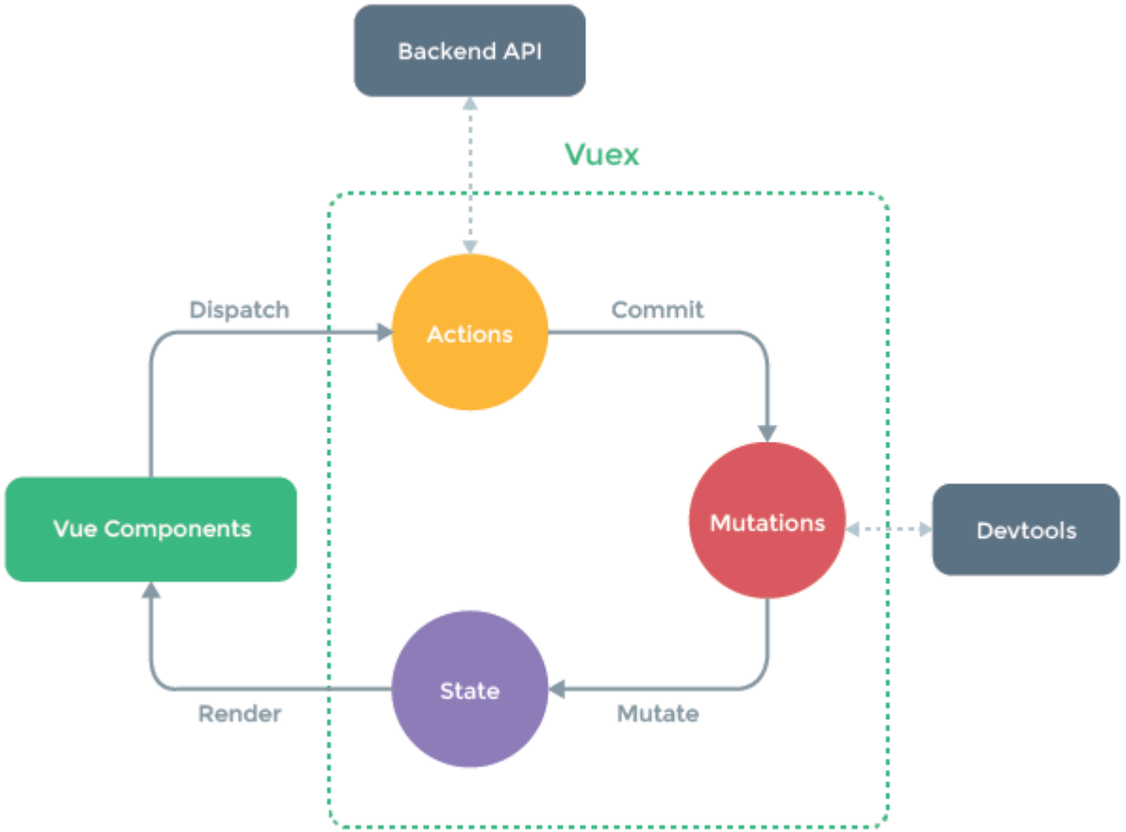
- State : 컴포넌트 간에 공유하는 데이터 `data()`
- View : 데이터를 표시하는 화면 `template`
- Action : 사용자의 입력에 따라 데이터를 변경하는 `methods`



단방향 데이터 흐름 처리를 단순하게 도식화한 그림

Vuex 구조

컴포넌트 -> 비동기 로직 -> 동기 로직 -> 상태



Vuex 설치하기

- Vuex는 싱글 파일 컴포넌트 체계에서 NPM 방식으로 라이브러리를 설치하는 게 좋다.

```
npm install vuex --save
```

※ ES6와 함께 사용해야 더 많은 기능과 이점을 제공받을 수 있음

Vuex 시작하기

- src 폴더 밑에 store 폴더를 만들고 store.js 파일 생성

```
import Vue from 'vue'
import Vuex from 'vuex'

export const store = new Vuex.Store({
  // ...
});
```

Vuex 기술 요소

- state : 여러 컴포넌트에 공유되는 데이터 `data`
- getters : 연산된 state 값을 접근하는 속성 `computed`
- mutations : state 값을 변경하하는 이벤트 로직 · 메서드 `methods`
- actions : 비동기 처리 로직을 선언하는 메서드 `aysnc methods`

state란?

- 여러 컴포넌트 간에 공유할 데이터 - **상태**

```
// Vue
data: {
  message: 'Hello Vue.js!'
}

// Vuex
state: {
  message: 'Hello Vue.js!'
}
```

```
<!-- Vue -->
<p>{{ message }}</p>

<!-- Vuex -->
<p>{{ this.$store.state.message }}</p>
```

getters란?

- state 값을 접근하는 속성이자 `computed()` 처럼 미리 연산된 값을 접근하는 속성

```
// store.js
state: {
  num: 10
},
getters: {
  getNumber(state) {
    return state.num;
  },
  doubleNumber(state) {
    return state.num * 2;
  }
}
```

```
<p>{{ this.$store.getters.getNumber }}</p>
<p>{{ this.$store.getters.doubleNumber }}</p>
```

mutations란?

- state의 값을 변경할 수 있는 **유일한 방법**이자 메서드
- 뮤테이션은 `commit()`으로 동작시킨다.

```
// store.js
state: { num: 10 },
mutations: {
  printNumbers(state) {
    return state.num
  },
  sumNumbers(state, anotherNum) {
    return state.num + anotherNum;
  }
}

// App.vue
this.$store.commit('printNumbers');
this.$store.commit('sumNumbers', 20);
```

mutations의 commit() 형식

- state를 변경하기 위해 mutations를 동작시킬 때 인자(payload)를 전달할 수 있음

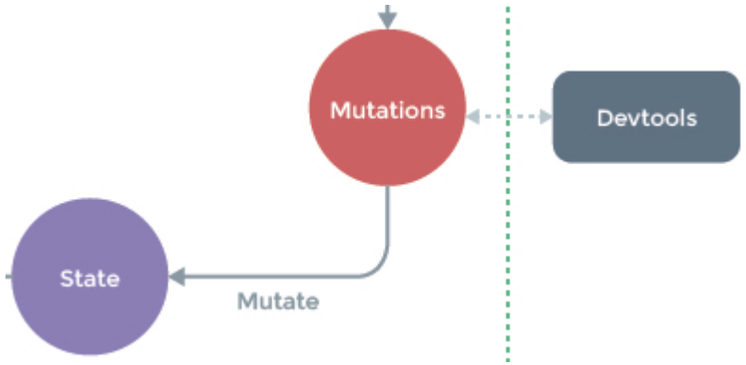
```
// store.js
state: { storeNum: 10 },
mutations: {
  modifyState(state, payload) {
    console.log(payload.str);
    return state.storeNum += payload.num;
  }
}

// App.vue
this.$store.commit('modifyState', {
  str: 'passed from payload',
  num: 20
});
```

state는 왜 직접 변경하지 않고 mutations로 변경할까?

- 여러 개의 컴포넌트에서 아래와 같이 state 값을 변경하는 경우 **어느 컴포넌트에서 해당 state를 변경했는지 추적하기가 어렵다**.

```
methods: {
  increaseCounter() { this.$store.state.counter++; }
}
```

- 특정 시점에 어떤 컴포넌트가 state를 접근하여 변경한 건지 확인하기 어렵기 때문
- 따라서, 뷰의 반응성을 거스르지 않게 명시적으로 상태 변화를 수행. **반응성**, **디버깅**, **테스팅 혜택**.



actions란?

- 비동기 처리 로직을 선언하는 메서드. 비동기 로직을 담당하는 mutations
- 데이터 요청, Promise, ES6 async과 같은 비동기 처리는 모두 actions에 선언

```
// store.js
state: {
  num: 10
},
mutations: {
  doubleNumber(state) {
    state.num * 2;
  }
},
actions: {
  delayDoubleNumber(context) { // context로 store의 메서드와 속성 접근
    context.commit('doubleNumber');
  }
}
// App.vue
this.$store.dispatch('delayDoubleNumber');
```

actions 비동기 코드 예제 1

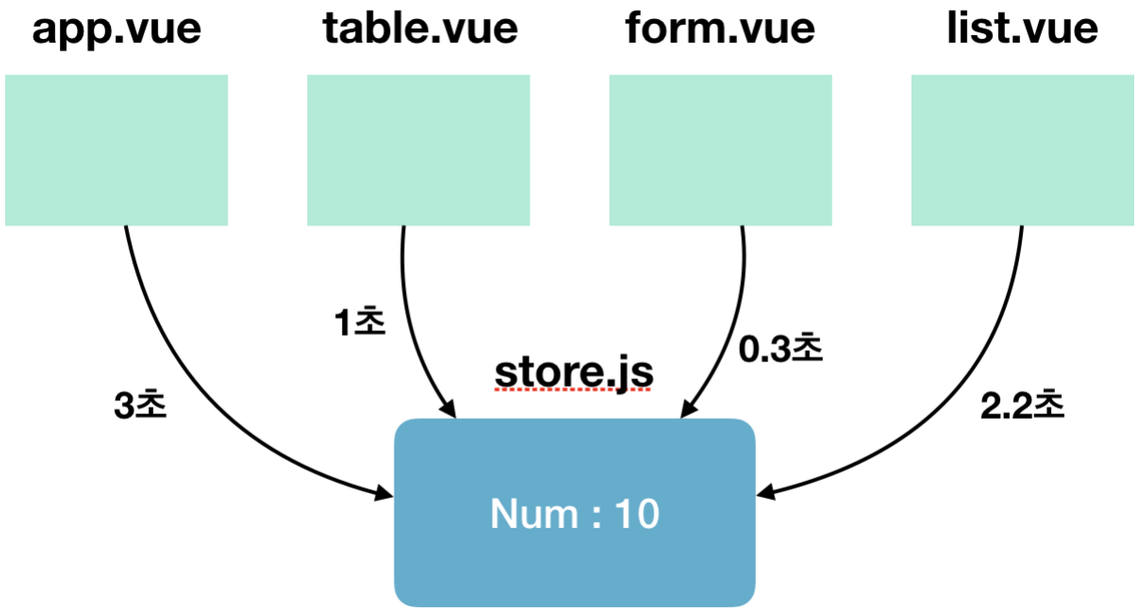
```
// store.js
mutations: {
  addCounter(state) {
    state.counter++
  },
},
actions: {
  delayedAddCounter(context) {
    setTimeout(() => context.commit('addCounter'), 2000);
  }
}

// App.vue
methods: {
  incrementCounter() {
    this.$store.dispatch('delayedAddCounter');
  }
}
```

actions 비동기 코드 예제 2

```
// store.js
mutations: {
  setData(state, fetchedData) {
    state.product = fetchedData;
  }
},
actions: {
  fetchProductData(context) {
    return axios.get('https://domain.com/products/1')
                .then(response => context.commit('setData', response));
  }
}

// App.vue
methods: {
  getProduct() {
    this.$store.dispatch('fetchProductData');
  }
}
```

왜 비동기 처리 로직은 actions에 선언해야 할까?

- 언제 어느 컴포넌트에서 해당 state를 호출하고, 변경했는지 확인하기가 어려움



[그림] 여러 개의 컴포넌트에서 mutations로 시간 차를 두고 state를 변경하는 경우

> **결론** : state **값의 변화를 추적하기 어렵기 때문에** mutations **속성에는 동기 처리 로직만 넣어야 한다.**

각 속성들을 더 쉽게 사용하는 방법 - Helper

Store에 있는 아래 4가지 속성들을 간편하게 코딩하는 방법

- state -> mapState
- getters -> mapGetters
- mutations -> mapMutations
- actions -> mapActions

헬퍼의 사용법

- 헬퍼를 사용하고자 하는 vue 파일에서 아래와 같이 해당 헬퍼를 로딩

```
// App.vue
import { mapState } from 'vuex'
import { mapGetters} from 'vuex'
import { mapMutations } from 'vuex'
import { mapActions } from 'vuex'

export default {
  computed() { ...mapState(['num']), ...mapGetters(['countedNum']) },
  methods: { ...mapMutations(['clickBtn']), ...mapActions(['asyncClickBtn']) }
}
```

Q) ...는 오타인가요? ES6의 Object Spread Operator입니다.

mapState

- Vuex에 선언한 state 속성을 뷰 컴포넌트에 더 쉽게 연결해주는 헬퍼

```
// App.vue
import { mapState } from 'vuex'

computed() {
  ...mapState(['num'])
  // num() { return this.$store.state.num; }
}

// store.js
state: {
  num: 10
}
```

```
<!-- <p>{{ this.$store.state.num }}</p> -->
<p>{{ this.num }}</p>
```

mapGetters

- Vuex에 선언한 getters 속성을 뷰 컴포넌트에 더 쉽게 연결해주는 헬퍼

```
// App.vue
import { mapGetters } from 'vuex'

computed() { ...mapGetters(['reverseMessage']) }

// store.js
getters: {
  reverseMessage(state) {
    return state.msg.split('').reverse().join('');
  }
}
```

```
<!-- <p>{{ this.$store.getters.reverseMessage }}</p> -->
<p>{{ this.reverseMessage }}</p>
```

mapMutations

- Vuex에 선언한 mutations 속성을 뷰 컴포넌트에 더 쉽게 연결해주는 헬퍼

```
// App.vue
import { mapMutations } from 'vuex'

methods: {
  ...mapMutations(['clickBtn']),
  authLogin() {},
  displayTable() {}
}

// store.js
mutations: {
  clickBtn(state) {
    alert(state.msg);
  }
}
```

```
<button @click="clickBtn">popup message</button>
```

mapActions

- Vuex에 선언한 actions 속성을 뷰 컴포넌트에 더 쉽게 연결해주는 헬퍼

```
// App.vue
import { mapActions } from 'vuex'

methods: {
  ...mapActions(['delayClickBtn']),
}

// store.js
actions: {
  delayClickBtn(context) {
    setTimeout(() => context.commit('clickBtn'), 2000);
  }
}
```

```
<button @click="delayClickBtn">delay popup message</button>
```

헬퍼의 유연한 문법

- Vuex에 선언한 속성을 그대로 컴포넌트에 연결하는 문법

```
// 배열 리터럴
...mapMutations([
  'clickBtn', // 'clickBtn': clickBtn
  'addNumber' // addNumber(인자)
])
```

- Vuex에 선언한 속성을 컴포넌트의 특정 메서드에다가 연결하는 문법

```
// 객체 리터럴
...mapMutations({
  popupMsg: 'clickBtn' // 컴포넌트 메서드 명 : Store의 뮤테이션 명
})
```

프로젝트 구조화와 모듈화 방법 1

아래와 같은 store 구조를 어떻게 모듈화 할 수 있을까?

```
// store.js
import Vue from 'vue'
import Vuex from 'vuex'

export const store = new Vuex.Store({
  state: {},
  getters: {},
  mutations: {},
  actions: {}
});
```

힌트! `Vuex.Store({})` 의 속성을 모듈로 연결

- ES6의 Import & Export를 이용하여 속성별로 모듈화

```
import Vue from 'vue'
import Vuex from 'vuex'
import * as getters from 'store/getters.js'
import * as mutations from 'store/mutations.js'
import * as actions from 'store/actions.js'

export const store = new Vuex.Store({
  state: {},
  getters: getters,
  mutations: mutations,
  actions: actions
});
```

프로젝트 구조화와 모듈화 방법 2

- 앱이 비대해져서 1개의 store로는 관리가 힘들 때 `modules` 속성 사용

```
// store.js
import Vue from 'vue'
import Vuex from 'vuex'
import todo from 'modules/todo.js'

export const store = new Vuex.Store({
  modules: {
    moduleA: todo, // 모듈 명칭 : 모듈 파일명
    todo // todo: todo
  }
});

// todo.js
const state = {}
const getters = {}
const mutations = {}
const actions = {}
```

실습 - Vue Todo 앱에 Vuex 적용

앞에서 배운 내용으로 Vuex를 적용해보기

참고 자료

- Vuex 튜토리얼 1
- Vuex 튜토리얼 2
- Vuex 튜토리얼 3
- Vuex 공식문서

끝