# VUEX vs PINIA

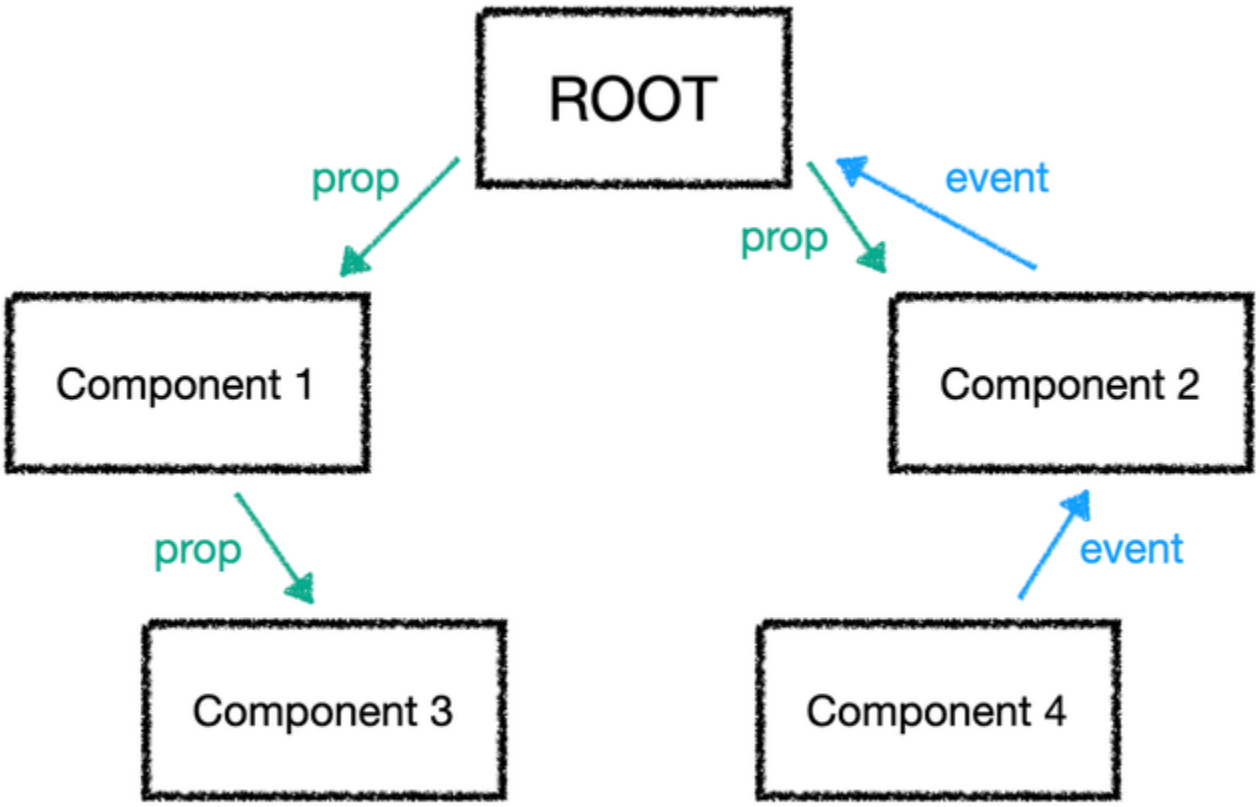
이 문서는 Vue.js의 상태관리 라이브러리 VueX와 pinia를 비교한 문서입니다.



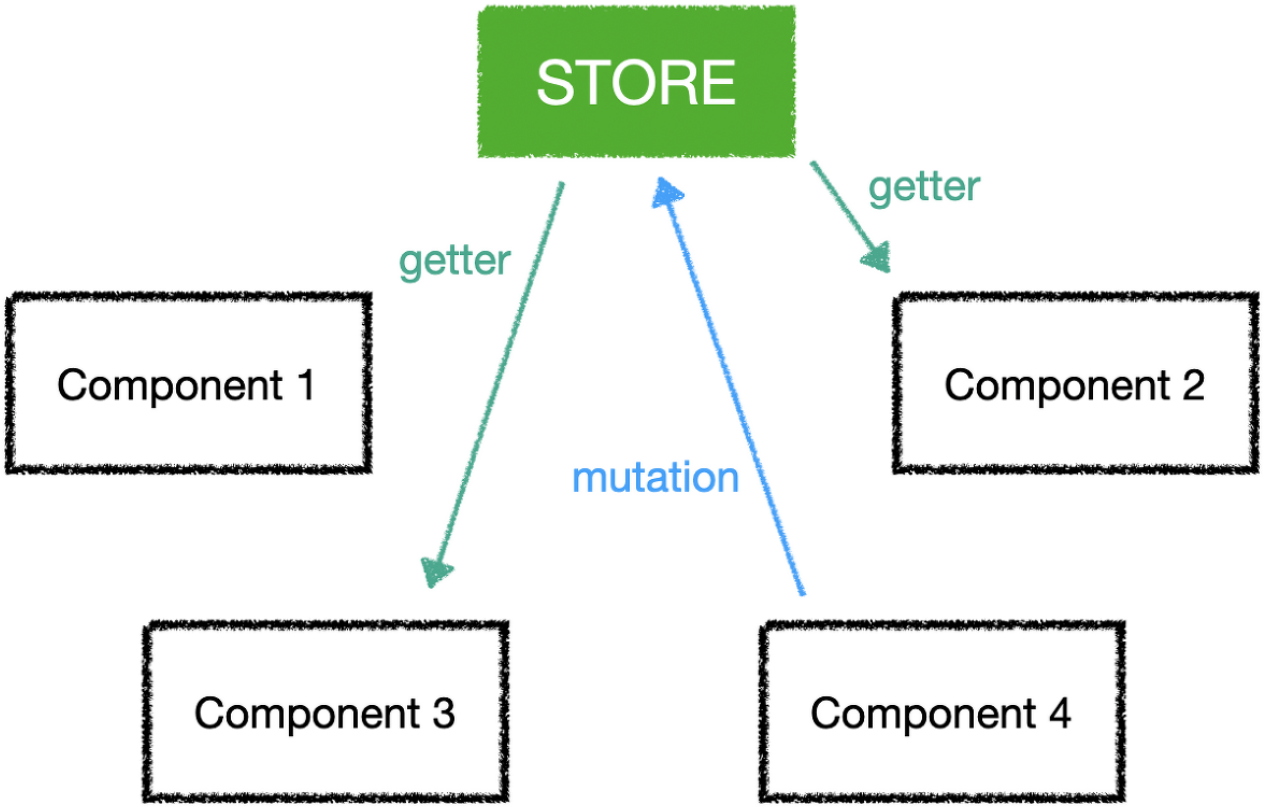
## Vue.js 상태관리 라이브러리

Vue.js는 하나의 화면에는 실제 수많은 컴포넌트로 설계 되어 구성되어져 있기 때문에 컴포넌트들 간에는 부모 컴포넌트와 자식 컴포넌트의 관계가 존재합니다.

부모-자식 컴포넌트의 관계가 많아질 경우 최상위에 부모 컴포넌트의 state의 값을 변경하기 위해 하위의 있는 자식들은 계속해서 인자로 넘겨줘야 한다는 꼬리 물기 식의 불필요한 코드가 반복적으로 생길 수 있습니다.

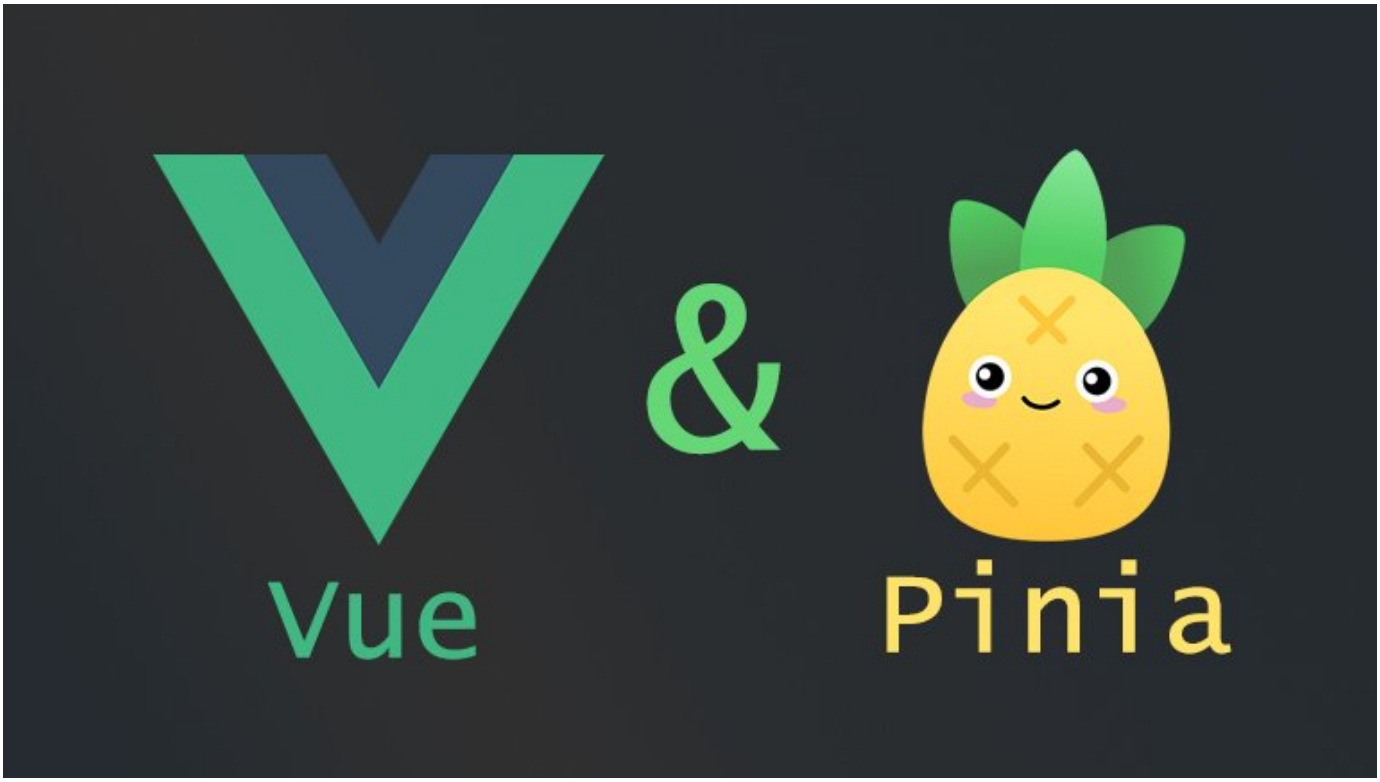



이러한 단점을 보안하기 위해 VUEX, PINIA와 같은 상태 관리 라이브러리가 등장했습니다. 이러한 라이브러리들은 모든 컴포넌트에 대한 **중앙 집중식 저장소 역할**을 합니다.


STORE
getter
getter
Component 1
Component 2
mutation
Component 3
Component 4

---

VueX, pinia




vue.js의 상태관리 라이브러리로는 VUEX를 흔히 사용하지만, 얼마전 VueConf Toronto 2021 에서 Vue 의 창시자 Evan You 가 직접 등장해 추천하는 상태 관리 플러그인을 Vuex 가 아닌 Pinia 로 공표하여 PINIA가 주목받고 있는 추세입니다.

차이점

이 둘의 차이점은 크게 3가지로 나눌 수 있습니다.

| VUEX | PINIA |
|---|---|
| Composition API 지원 (X) | Composition API 지원 (O) |
| 단일 Store 사용<br>> namespaced modules o | 여러개의 Store 생성 가능<br>> namespaced modules x |
| Mutations, Actions, Getters | Actions, Getters |

### 1. Composition API

첫번째로 pinia에서는 Vue3의 Composition API에 코드 작성 방식을 제공하고 있습니다. 따라서 기능별로 나누어 개발할 수 있기 때문에 좀 더 논리적 관점에서의 개발이 가능하게 됩니다.

1-1. Optional

```
// stores/list.js
import { defineStore } from "pinia";

export const useListStore = defineStore("list",{
  state : () => ({ list: [] }),
  actions: {
    addList(param){
      this.list.push(param);
    }
    //addList: (param) => this.list.push(param);
  },
  getters: {
    getAllList(state){
      return state.list;
    }
    //getAllList: (state) => state.list
  }
});
```

1-2. Composition

```
// stores/list.js
import { defineStore } from "pinia";
import { ref, computed } from "vue";

export const useListStore = defineStore("list", () => {
  const list = ref([]);
  function addList(param) {
    list.value.push(param);
  }
  const getDataAll = computed(() => list.value);
  return { list, addList, getDataAll };
});
```

### 2. 생성 가능한 Store의 갯수

2-1. VueX

어플리케이션의 규모가 커지면 관리해야 할 요소의 성격이 완전히 다른 경우가 종종 생깁니다. VueX는 **하나의 Store만 가질 수** 있기 때문에 이처럼 성격이 다른 경우에도 하나의 파일에서 관리해야 합니다.

이때 **VueX에서는 namespaced modules를** 이용하여 폴더를 구분한 후 성격에 따른 state와 method를 분리하여 구현할 수 있습니다. 아래와 같이 폴더를 구분할 경우 폴더 이름이 곧 Vuex 네임스페이스가 됩니다.

```
store
 Auth
   actions.js
   mutations.js
   index.js

 Todo
    actions.js
    mutations.js
    index.js
```

namespace를 생성한후에 실제 컴포넌트에서는 다음과 같이 사용합니다.

```
<template>
  <Home />
</template>

<script>
  import { mapMutations, mapState } from "vuex";

  export default {
    name: "MainPage",
    components: {
      Home: () => import("@/components/pages/home"),
    },
    methods: {
      //  namespace  .
      ...mapMutations("ansData", ["mutateANSData"]),
    },
  };
</script>
```

2-2. pinia

pinia는 이와 비교하여 **여러 개의 store**를 가질 수 있기 때문에 namespaced modules 역시 사용하지 않습니다. 이처럼 Vuex와 비교했을 때 Pinia는 간결한 구조를 가지지만 namespaced 되어 독립된 store 객체를 가진다는 컨셉은 유효합니다.

```
import Vue from "vue";
import { useCartStore } from "./stores/cart";
import { useUserStore } from "./stores/user";

export default defineComponent({
  name: "App",
  setup() {
    const user = useUserStore();
    const cart = useCartStore();

    function addItemToCart() {
      cart.addItem(user.name);
    }
  ...
```

### 3. Mutation 유무

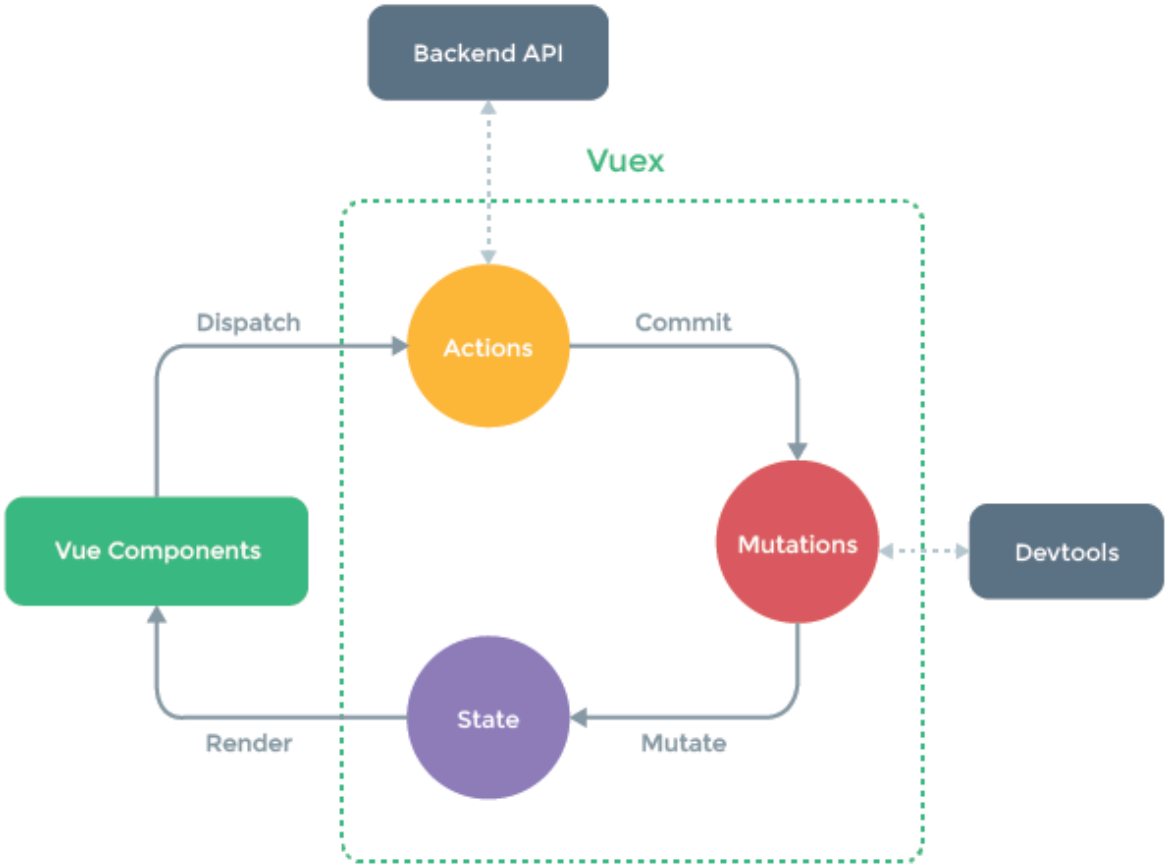
3-1. VueX

VueX에서는 Store라는 저장소에 컴포넌트 간 공유할 data 속성인 state를 정의합니다. 이렇게 정의한 state 데이터를 변경할 수 있는 로직을 각각 Component에서 정의할 수도 있지만, 공통되는 로직을 매번 따로 정의할 경우, 가독성이 떨어지고 성능이 떨어질 수 있습니다. 따라서 Mutations, Actions, Getters 를 사용하여 **공통으로 사용되는 로직을 한곳에서 관리**합니다.

이 3가지는 공통으로 사용되는 로직을 한곳에서 관리한다는 공통점이 가지지만, 아래와 같은 차이점을 가지고 있습니다.

| Mutations | Actions | Getters |
|---|---|---|
| method | method | Computed |
| 동기 | 비동기 | |

Mutations 의 성격상 안에 정의한 로직들이 순차적으로 일어나야 각 컴포넌트의 반영 여부를 제대로 추적할 수가 있기 때문에 동기적인 작업만 수행할 수 있습니다. 따라서 비동기적인 로직을 수행하기 위해서는 **Action 내부에 Mutations 을 commit하는 등의 작업을 추가로 진행**해야 한다.


Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
State
Render
Mutate

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

Vue.use(Vuex);

export default new Vuex.Store({
    state: {
        count: 0,
    },
    mutations: {
        increment(state) {
            state.count++;
        },
        decrement(state) {
            state.count--;
        }
    },
        actions: {
          incrementAsync ({ commit }) {
            setTimeout(() => {
              commit('increment')
            }, 1000)
          }
        }
})
```

3-2. pinia

pinia에서는 비동기적, 동기적 로직 모두 action에서 수행할 수 있기 때문에 mutations이 없습니다. 따라서 불필요한 코드가 줄고, 좀 더 간결한 코드를 작성할 수 있습니다.

```
// stores/list.js
import { defineStore } from "pinia";

export const useListStore = defineStore("list",{
  state : () => ({ count: 0 }),
  actions: {
    incrementAsync () {
            setTimeout(() => {
              state.count++;
            }, 1000)
          }
  }
});
```

---

## 마무리

지금까지 VueX와 pinia의 차이점에 대해 정리해보았습니다.

감사합니다.