KB금융그룹 | 국민의 평생 금융파트너

애플리케이션 데이터 상태관리 라이브러리.

부모 -> 자식 : 속성을 통해서 props
자식-=>부모 : 이벤트 를 통해서 $emit & @이벤트

관계가 없는 혹은 형제끼리는? 전역 변수를 이용한 방식 provide/inject
=>문제와 한계가 있음.
=> 해결할 pinia

2025년 상반기 K-디지털 트레이닝

pinia를 이용한 상태 관리

[KB] IT's Your Life

KB 국민은행

pinia

- Composition API 방식으로 Vue 애플리케이션을 위한 중앙 집중화된 상태관리 기능을 제공
- Vue3의 공식 상태 관리 라이브러리
  - 프로젝트 생성시 추가할지 질문에 yes 답변하면 자동 추가

- 참고
  - 이전에는 vuex라는 상태 관리 라이브러리 사용

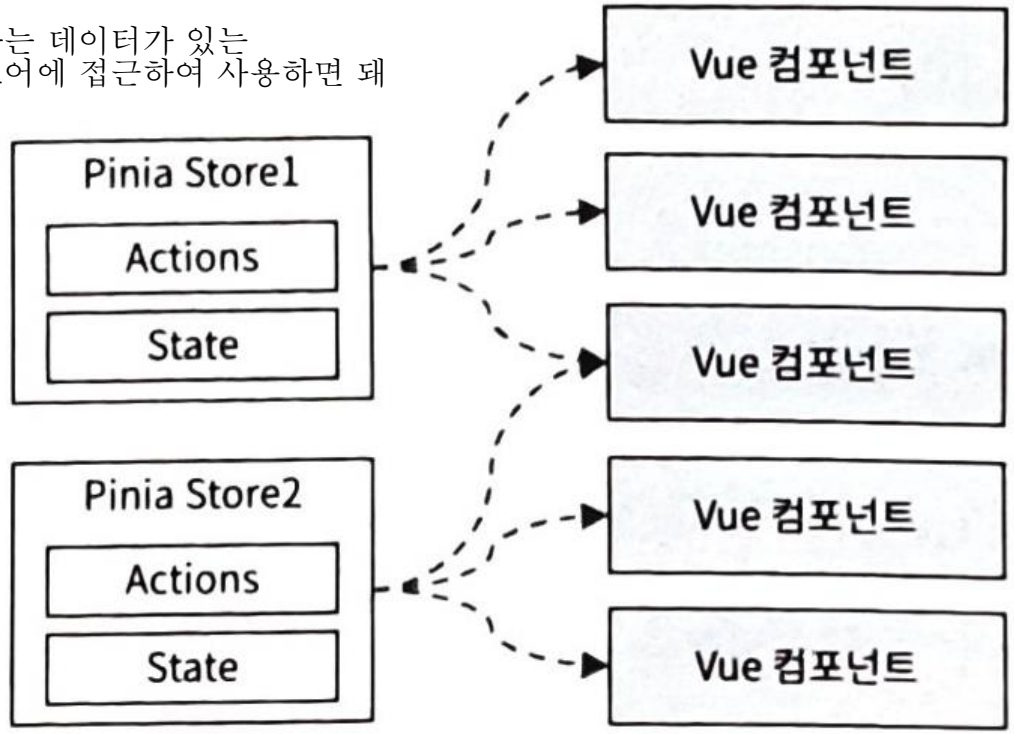
    vuex는 option api와 맞는 기술. 지저분함.

    composition api와 궁합이 안 맞음 pinia는 좋아

Pinia Store

pinia에서는 정의만 해주면
구조를 알아서 잡아준다.

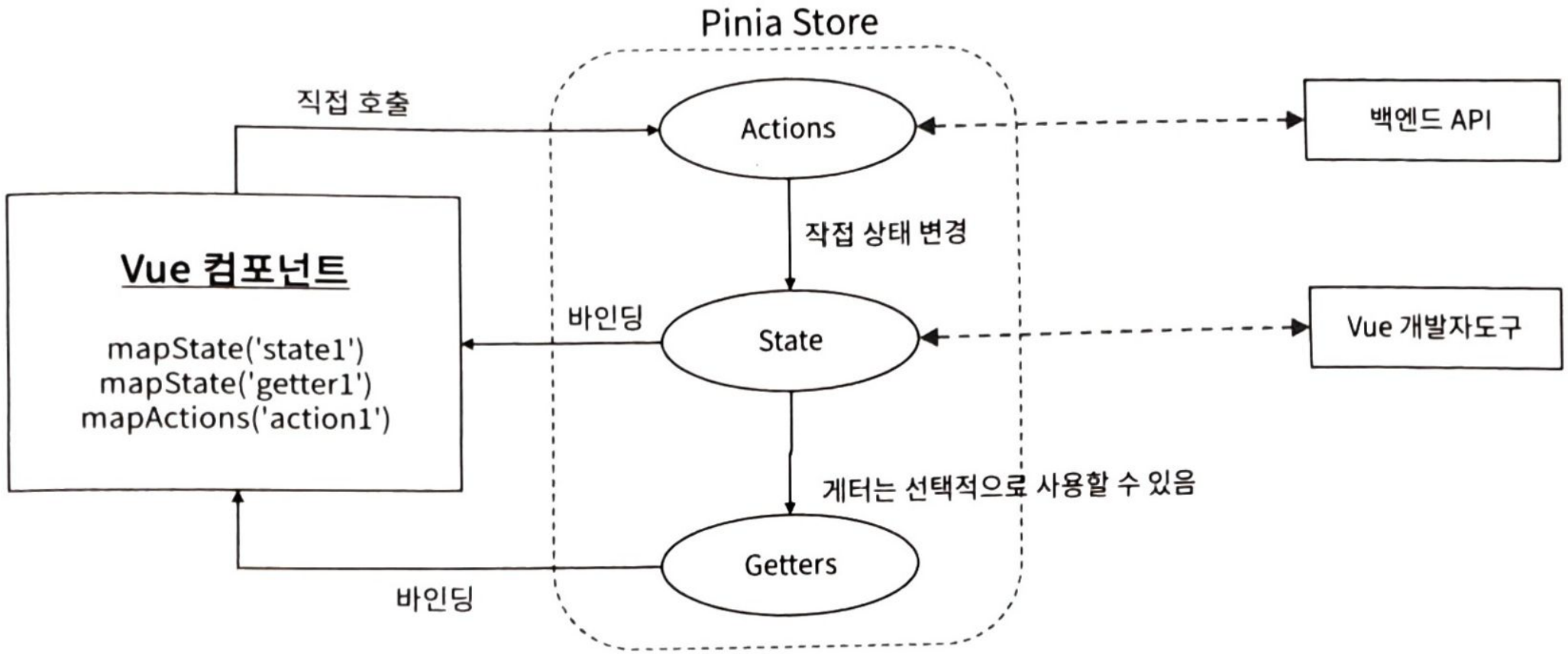
원하는 데이터가 있는
스토어에 접근하여 사용하면 돼



2 pinia 아키텍처와 구성 요소

pinia 아키텍처

상태 관리 툴에서 힘든 작업이 비동기 작업이다.
피니아는 깔끔하게 해결함.
피니아도 2가지 방식이 있다. 옵션, 컴포지션
여기서는 컴포지션만.



2 pinia 아키텍처와 구성 요소

상태관리하는 함수.

스토어 정의

defineStore 함수 이용

- defineStore('스토어명', 함수)

함수 인자

- 반응형 상태 정의
- 계산된 상태 정의
- 상태에 접근하는 action 함수 정의
- 외부에서 사용할 항목을 객체로 리턴

```
// [ 컴포지션 API 방법 적용 ]                                    셋업함수
export const useCount2Store = defineStore('count2', ()=>{
    const state = reactive({ count : 0 });                    상태데이터 정의
    const increment = ({ num }) => {
        state.count +=num;                                    상태데이터 가공 함수 정의
    }
    const count = computed(()=>state.count);                  읽기 전용
                                                              데이터 정의
    return { count, increment };
})
```

반환 되는 객체는
useCount2Store( ) 의 리턴 값이다.

즉 useCount2Store는 함수이고
defineStore 함수의 반환값은 함수다.

pinia를 사용하도록 Vue 애플리케이션 설정

- 프로젝트 생성시 pinia 추가하면 자동 생성됨

```
import { createApp } from 'vue'
import { createPinia } from 'pinia'
import App from './App.vue'

const pinia = createPinia()

const app = createApp(App)

app.use(pinia)
app.mount('#app')
```

컴포넌트에서 스토어 사용

- 스토어 import
- setup에서 반응성 있게 연결

[컴포지션 API를 사용한 컴포넌트에서의 스토어 사용]

```
<script>
import { useCount1Store} from '@/store/counter.js'
import { computed } from 'vue';

export default {
  setup() {
    const store = useCount1Store();
    const count =  computed(()=>store.count);
    const increment = store.increment;

    return { count, increment };
  }
}
</script>
```

store는 객체를 반환하는 함수

줄여서 쓰기 위해
추출하는 단계를
넣었음

computed한 것을 또 computed해?
그냥
count=store.count하면
count는 반응 성을 잃어버림.
그냥 store.count를 사용하면
괜찮은데
count=store.count하면
값의 복사만 이뤄어줘 반응성 잃음.

그래서 computed로 감싸서
반응성을 잃지 않으면서
줄여서 쓰기 위해
이렇게 한 것.

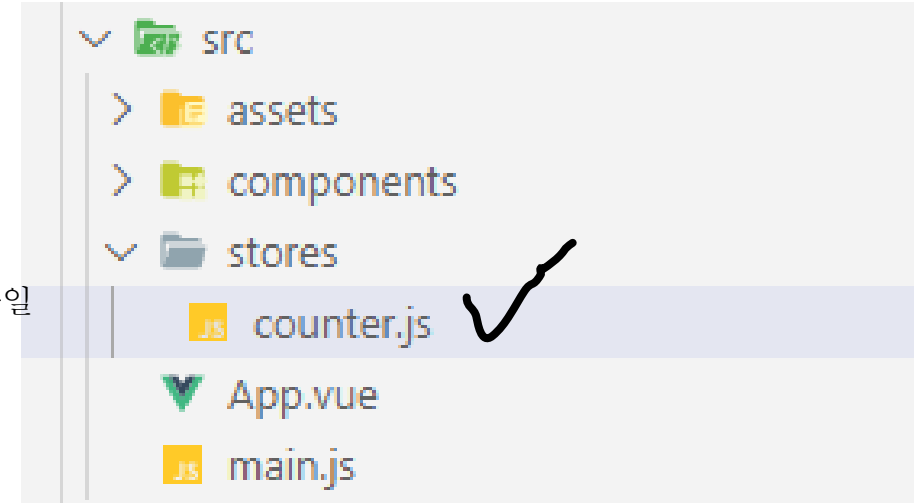
프로젝트 만들기

npm init vue pinia-test-app

Vue.js - The Progressive JavaScript Framework

✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? » No
✓ Add ESLint for code quality? ... No / Yes

샘플 피니아 스토어 js파일



src/main.js

```
import './assets/main.css'

import { createApp } from 'vue'
import { createPinia } from 'pinia'
import App from './App.vue'

const app = createApp(App)

app.use(createPinia())

app.mount('#app')
```

src/stores/counter.js

```
import { ref, computed } from 'vue'
import { defineStore } from 'pinia'

export const useCounterStore = defineStore('counter', () => {
  const count = ref(0)
  const doubleCount = computed(() => count.value * 2)
  function increment() {
    count.value++
  }

  return { count, doubleCount, increment }
})
```

상태 데이터

읽기 전용 & 가공 데이터

상태데이터 가공 함수

피니아로 만든 것은 vue디버깅 창에서 봐야 분석하기 쉽다.
store항목에서 봐라

App.vue

```
import { userCounterStore } from '경로/counter.js';

const store=userCounterStore( );

store.doubleCount;
store.increment( );

const newCount= computed(( ) => store.doubleCount );
```

todoList 스토어 생성

- src/stores/todoList.js
- 반응형 상태
  - state { todoList: [] }
- 액션
  - addTodo(todo)
  - deleteTodo(id)
  - toggleDone(id)
- 계산된 상태
  - doneCount → 리턴값이 기본 타입인 경우 사용자 측에서도 계산된 속성으로 연결해야 함!
  - todoList → 참조형인 경우 바로 사용 가능

컴포넌트에서

src/stores/todoList.js

```
export const useTodoListStore = defineStore("todoList", ()=> {
    // 방응형 상태
    const state = reactive({
        todoList : [
            { id: 1, todo: "ES6학습",  done: false },
            { id: 2, todo: "React학습",  done: false },
            { id: 3, todo: "ContextAPI 학습",  done: true },
            { id: 4, todo: "야구경기 관람", done: false },
        ]
    })

    // action
    const addTodo = (todo) => {
        state.todoList.push({ id: new Date().getTime(), todo, done: false })
    }

    const deleteTodo = (id) => {
        let index = state.todoList.findIndex((todo) => todo.id === id);
        state.todoList.splice(index, 1);
    }

    const toggleDone =  (id) => {
        let index = state.todoList.findIndex((todo) => todo.id === id);
        state.todoList[index].done = !state.todoList[index].done;
    }
```

src/stores/todoList.js

```
    // 계산된 속성
    const doneCount = computed(()=> {
        return state.todoList.filter((todoItem)=>todoItem.done === true).length;
    })

    const todoList = computed(()=> state.todoList);

    return { todoList, doneCount, addTodo, deleteTodo, toggleDone };
})
```

기억해 computed는 캐시를 운영한다.

src/App.vue

```
<template>
  <div>
    <h2>TodoList 테스트(Composition API)</h2>
    <hr />
    할일 추가 :
    <input type="text" v-model="todo" />
    <button @click="addTodoHandler">추가</button>
    <hr />
    <ul>
      <li v-for="todoItem in todoList">
        <span style="cursor: pointer" @click="toggleDone(todoItem.id)">
          {{ todoItem.todo }} {{ todoItem.done ? '(완료)' : '' }}
        </span>
           
        <button @click="deleteTodo(todoItem.id)">삭제</button>
      </li>
    </ul>
    <div>완료된 할일 수 : {{ doneCount }}</div>
  </div>
</template>
```

src/App.vue

```
<script setup>
import { useTodoListStore } from '@/stores/todoList.js';
import { ref, computed } from 'vue';

const todo = ref('');

const todoListStore = useTodoListStore();
const { todoList, addTodo, deleteTodo, toggleDone } = todoListStore;

const doneCount = computed(() => todoListStore.doneCount);// 기본 타입에 대해서는 계산된 속성을 다시 작성

const addTodoHandler = () => {
  addTodo(todo.value);
  todo.value = '';
};
</script>
```

3 간단한 pinia 예제 작성