Day7

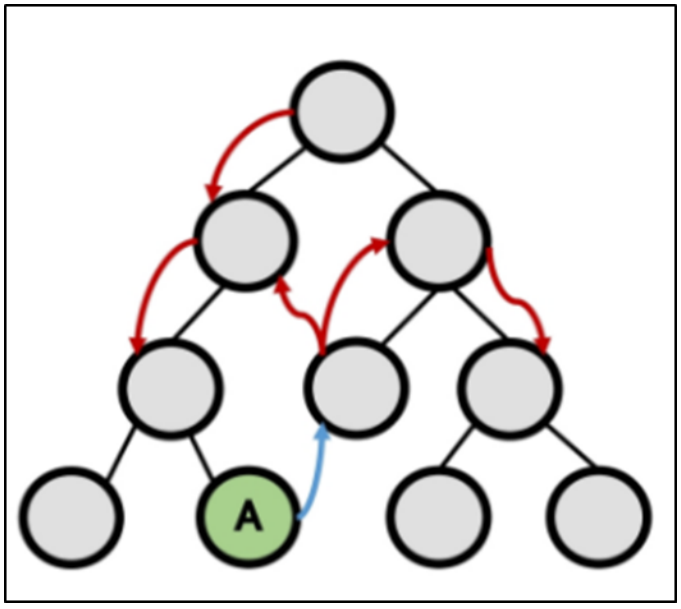
자료	Vue
구분	Vue

pinia

`props` / `emit` 만으로 프로젝트를 완성할 수 있으나, 다음 경우에 문제가 된다.

- 컴포넌트 구조가 복잡해질 경우
- 부모 자식 관계가 아닐 경우

그림으로 보면 다음과 같다.



이렇게, 컴포넌트 계통의 깊이가 길어질수록 데이터를 다루기가 매우 어려워진다.

그래서 이러한 문제점을 극복하기 위해 vue 에서는 vuex의 이벤트 method 들을 사용하기도 하고

Props / emit과 비슷한 Provide / Inject 를 이용해서 '직계' 부모-자식 관계가 아니더라도 컴포넌트 간에 데이터를 주고 받는 방법이 있다. 하지만, Vue.js 공식 가이드 문서에서는 Provide / Inject 를 굳이 사용을 추천하지 않는다.

- **상세:**

> ! `provide`와 `inject`는 주로 고급 플러그인/컴포넌트 라이브러리를 위해 제공됩니다. 일반 애플리케이션 코드에서는 사용하지 않는 것이 좋습니다.

이 옵션 쌍은 함께 사용하여 상위 컴포넌트가 컴포넌트 계층 구조의 깊이에 관계없이 모든 하위 항목에 대한 종속성을 주입하는 역할을 하도록 허용합니다. React에 익숙하다면 이것은 React의 컨텍스트 기능과 매우 유사합니다.

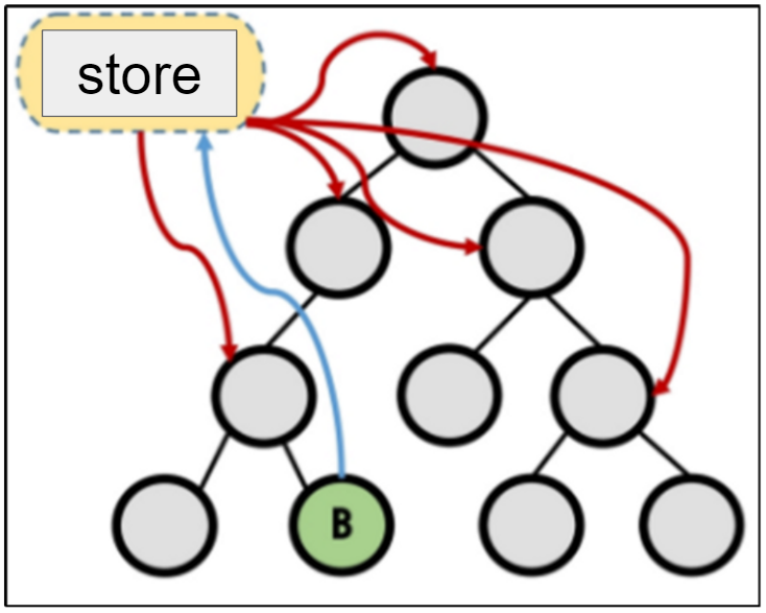
https://v2.ko.vuejs.org/v2/api#provide-inject

큰 프로젝트일수록 Vue.js 에서 기본 제공하는 기능 이외의 서드파티 패키지를 고려해야 한다.

Vue 3 공식문서 에서는 Pinia 를 상태 관리 패키지 (state management package) 로 추천하고 있다.

ㄱ

Pinia 의 기본 아이디어는 다음과 같다.



store 라는 곳에 state를 저장해서, 각 컴포넌트마다 state 를 가져오거나, 변경한다.

이 경우, 위의 두 가지 문제가 해결된다.

- 컴포넌트 구조가 복잡해지더라도, store 에 접근하면 된다.

- 부모 자식 관계를 신경쓰지 않고, store 에 접근하면 된다.

props/emit 과 Pinia 모두 프로젝트 진행 시 많이 사용을 한다. 즉, 모든 state를 Pinia 로 작업하지는 않는다는 것을 알고 있자. 상황에 따라서 적절하게 섞어서 사용하면 된다.

실습을 위해 프로젝트를 하나 만들고 초기 셋팅을 하자

```
$ npm create vue@latest

// 그리고 아래 옵션을 Yes로 체크하자
√ Add Vue Router for Single Page Application development? ... No / Yes
√ Add Pinia for state management? ... No / Yes

$npm install

// 그리고 필요없는 파일들을 삭제하자.
1. assets 폴더 안에 파일들
2. components 폴더안을 비우자
3. main.js 에서 import './assets/main.css'  지우기

4. App.vue 코드 지우고 아래 코드 남겨놓기

<script setup>
import { RouterLink, RouterView } from 'vue-router'
</script>

<template>
  <header>
    <nav>
      <RouterLink to="/">Home</RouterLink> |
      <RouterLink to="/about">About</RouterLink>
    </nav>
  </header>
  <RouterView />
</template>
```

`HomeView`

```
<script setup>
import HomeChild from "@/components/HomeChild.vue";
</script>

<template>
  <h1>Home</h1>
  <HomeChild />
</template>
```

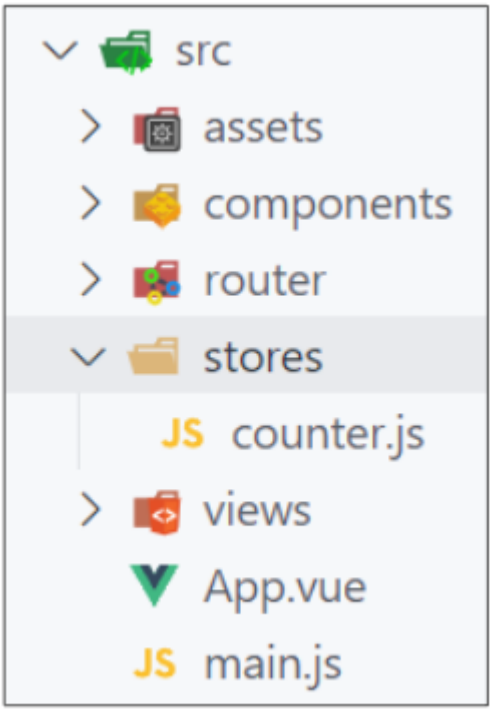
컴포넌트에 `HomeChild.vue` 그리고 `GrandChild.vue` 도 생성 해보자.

`HomeChild`

```
<script setup>
import GrandChild from "@/components/GrandChild.vue";
</script>

<template>
  <h2>Child</h2>
  <GrandChild />
</template>
```

`GrandChild`

```
<template>
  <h3>GrandChild</h3>
</template>
```



`src/` 디렉터리 안에 `stores/` 디렉터리가 바로 Pinia 의 영역이며, 이 디렉터리에는 상황에 따라 여러 개의 store 가 존재할 수 있다.

파일을 확인해 보자.

현재에는 `counter.js`가 존재해 있을 것이며, 하나의 js 파일 안에는 파일명과 일치하는 단 하나의 store, `counter` 가 존재할 것이다.

```
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 }
})
```

export const useCounterStore = defineStore('counter', () =>

여기서 부터 코드를 하나씩 살펴보자.

- `defineStore` 를 사용해 store 를 정의한다. `Pinia`는 `defineStore`라는 함수를 이용하여 각각의 파일마다 별도의 `store`를 정의한다.
- `defineStore` 의 첫번째 아규먼트는 store 의 이름이다. 파일명과 일치하게 네이밍을 하며, 이름은 전체 vue 프로젝트에서 중복되면 안된다.
- 두번째 아규먼트는 콜백함수다. state, computed, function 을 정의하며, 정의한 모든 것을 모아서 리턴을 해줘야 한다.
- `defineStore` 는 컴포저블을 리턴하며, `use` + store이름 + `Store` 로 네이밍하고, 맨 앞에 `export` 를 붙여준다.

`useCounterStore` 컴포저블에서 정의하는 세 가지가 핵심이다.

- state : 지금까지 `ref` 로 정의해서 사용한 것과 같으며, 모든 컴포넌트에서 자유롭게 사용할 수 있다. 위의 예제에서는 `count` 에 해당한다.
- computed : state 의 값을 변경 시키지 않고, 다른 형태로 보여주기 위해 사용한다. 예제에서는 `count` 의 값을 변경 시키지는 않고, 두 배로 표시해서 보여준다.
- function : 특정한 로직을 정의하되, 반드시 state 변경과 관련이 있어야 한다.

즉, computed 와 function 의 차이는 state 를 실제로 변경 시키는지 아닌지의 차이라고 보면 된다. computed 는 state 를 특정한 처리를 해서 "보여줄" 뿐이고, function 은 state 를 반드시 "변경해야" 한다.

- 컴포저블은 Vue 에서 이름이 `use` 로 시작하는 특별한 함수를 의미하며, API 의 일종이다. 다른 API와 다른 점은, state 를 포함하고 있어 여러 컴포넌트에서 state 를 공유한다는 점이다. 더 자세한 사항은 링크를 참조하자.

  https://v3-docs.vuejs-korea.org/guide/reusability/composables.html

이제 컴포넌트에서 사용해보자. `HomeView` 를 다음과 같이 정의한다.

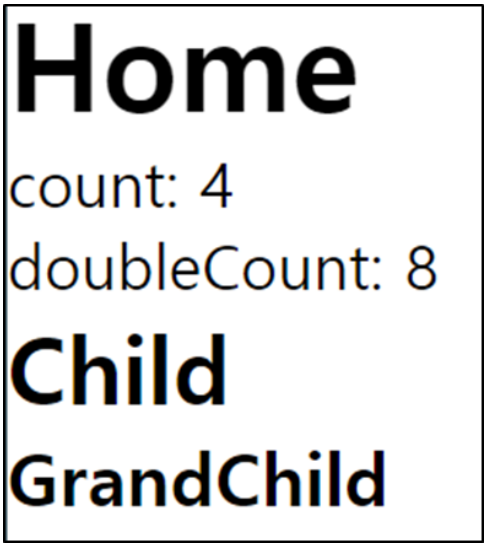
```
<script setup>
import { useCounterStore } from "@/stores/counter";
import HomeChild from "@/components/HomeChild.vue";

const counter = useCounterStore();

counter.count = 3;
counter.increment();
</script>

<template>
  <h1>Home</h1>
  <div>count: {{ counter.count }}</div>
  <div>doubleCount: {{ counter.doubleCount }}</div>
  <HomeChild />
</template>
```

`useCounterStore` 컴포저블을 store 경로에서 import하고, `counter` 라는 객체를 하나 리턴 받는다.

여기서 `counter` 는 `defineStore` 의 첫번째 아규먼트로 정의한 `counter` 라는 이름과 일치해야 한다.

각 구문별 사용예제

- `counter.count = 3` state 직접 변경
- `counter.increment()` 함수 실행. `counter.count` 는 1 증가할 것임
- `{{ counter.count }}` , `{{ counter.doubleCount }}` 평소에 state 출력하듯 그대로 쓰면 된다.



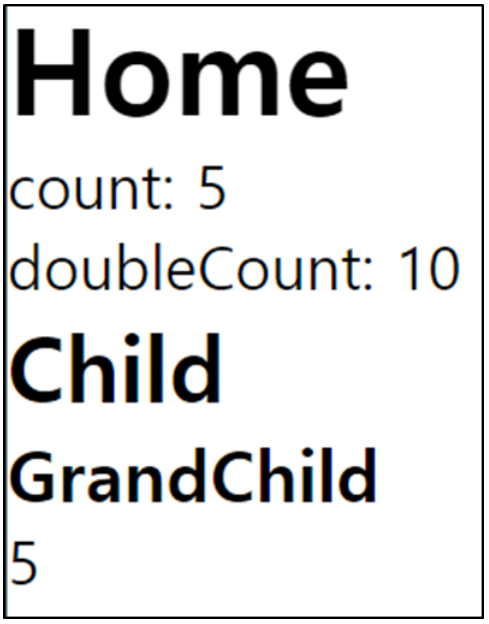
Pinia 의 최대 장점은 더이상 `props` / `emit` 처럼 컴포넌트가 부모-자식 관계일 필요가 없다는 것이다. 아무 컴포넌트에서나 원하는 state, computed, function 을 사용할 수 있다.

- state 를 직접 변경할 수 있는데 굳이 함수를 만드는 이유는 뭘까? state 는 매우 중요한 데이터이기 때문에, 아무 때나 접근할 수 있도록 허용하면 각종 문제가 생길 수 있다. 즉, function 이 setter 역할을 한다고 보면 된다.

`GrandChild` 컴포넌트를 다음과 같이 변경해보자.

```
<script setup>
import { useCounterStore } from "@/stores/counter";

const counter = useCounterStore();
counter.count++;
</script>

<template>
  <h3>GrandChild</h3>
  {{ counter.count }}
</template>
```

`counter.count` 를 1 증가해보았다.



`GrandChild` 뿐만 아니라 `HomeView` 에서도 즉시 반영 되는 것을 알 수 있다.

이번엔 `name` state 를 하나 더 추가해보고, 아규먼트를 받아 state 를 변경하는 함수를 만들어보자.

```
import { ref, computed } from "vue";
import { defineStore } from "pinia";

export const useCounterStore = defineStore("counter", () => {
  const count = ref(0);
  const name = ref("kevin");
  const doubleCount = computed(() => count.value * 2);
  function increment() {
    count.value++;
  }
  function changeName(newName) {
    name.value = newName;
  }

  return { count, name, doubleCount, increment, changeName };
});
```

`name` state 를 하나 추가했다. 즉, 하나의 store 는 여러 개의 state 를 가질 수 있다.

`changeName` function 을 선언했다. `newName` 로 인자값을 받아 `name` state 를 변경한다.

`return` 에 `name` , `changeName` 을 추가하는 것을 잊지 말자.

이번엔 `HomeChild` 에서 사용해보자.

```
<script setup>
import { useCounterStore } from "@/stores/counter.js";
import GrandChild from "@/components/GrandChild.vue";

const counter = useCounterStore();
counter.changeName("ssafy");
</script>

<template>
  <h2>Child</h2>
  {{ counter.name }}
  <GrandChild />
</template>
```

중간 중간에 랜더링이 업데이트가 안될 수 있으니 새로고침을 자주 해주자.

원래 `name` 의 값 `kevin` 은 `changeName` 에 의해 `ssafy` 로 잘 변경되어 출력 되는 것을 알 수 있다.

추가로, 만약 서버로부터 store 의 데이터를 가져올 일이 있다면, function 에 axios 비동기 구문으로

작성해두었다가 호출하는 방식으로 사용하곤 한다. 라이브 시간에 실습한 TodoList를 한번 만들어 보고

온라인 실습을 진행하자.