Day5

자료	Vue
구분	Vue

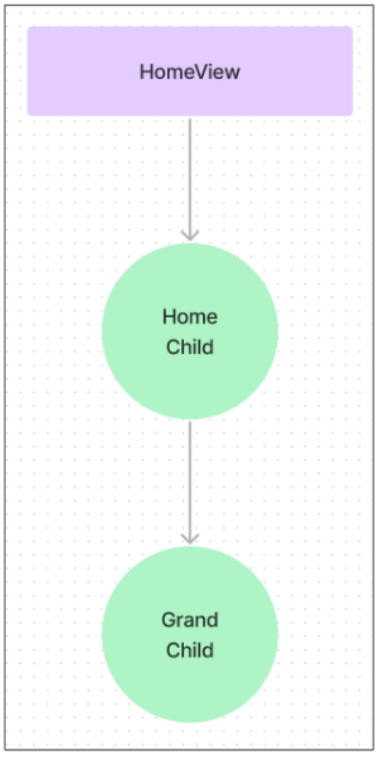
프로젝트를 생성을 하고 아래 실습을 진행하자.

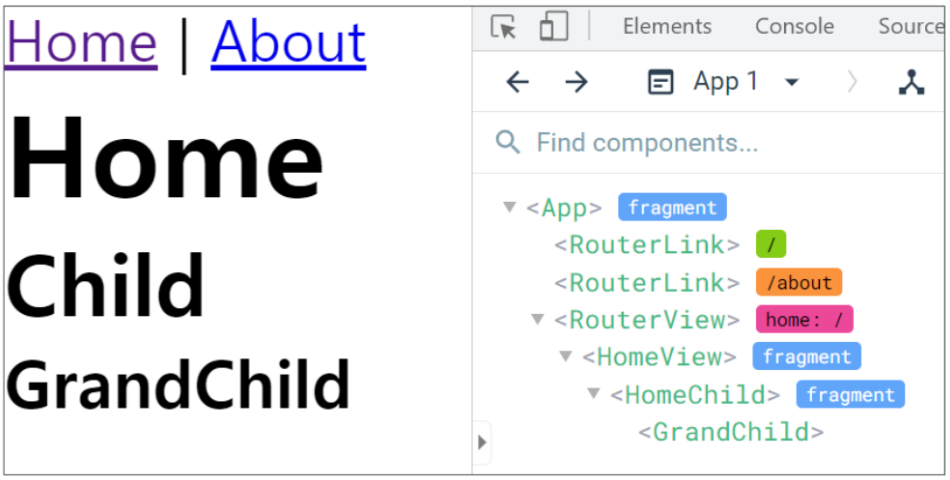
props

`components/` 디렉터리 안에 있는 모든 컴포넌트를 지우고, `App.vue` 를 다음과 같이 초기 상태로 작성하자.

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

도전: 다음 구조로 컴포넌트를 만들자.





그리고 `HomeView` 컴포넌트를 다음과 같이 변경한다.

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

const name = ref("ssafy");
const age = ref(10);
</script>

<template>
  <h1>Home</h1>
  <div>{{ name }}</div>
  <div>{{ age }}</div>
  <HomeChild />
</template>
```

현재 데이터는 어디에 있는가? `HomeView` 에 있다. 지금 하고자 하는 것은 `HomeChild` 자식 컴포넌트로, `name` 과 `age` 를 내려보내는 것이다.

왜 이런 일을 하고싶을까? 인스타그램이나 페이스북에 접속하면, 한 화면에 프로필 사진 또는 이름이 중복되어서 표시되는 영역이 확인될 것이다.

지금 보면 sujeongsshi 라는 아이디가 보이는 곳은 총 두 곳이다.

하나는 피드의 헤더 부분, 다른 하나는 피드의 컨텐트 부분이다.

이 둘은 엄연히 다른 컴포넌트이므로, 아무 생각 없이 개발한다면 각각의 컴포넌트에 별도로 선언된 state 로 관리될 것이다.

문제는 아이디 sujeongsshi 의 이름이 변경 되었을 때다. 그럼 각 컴포넌트별로, 서버로 두 번 요청을 하게 된다.

그럼 아래 딜레마가 생기게 된다.

- 만약 두 개 정도가 아니라, 열 개의 서로 다른 컴포넌트에서 동일한 아이디를 사용한다면?
- 만약 아이디같은 `string` 타입이 아니라, 프로필 사진 URL 같이 훨씬 긴 데이터라면?
- 이런 상황이 수천명의 사용자에게 동일하게 적용된다면?

이 문제를 해결하기 위해, 중요한 state 는 부모에서 통합해서 관리하고자 한다.

실제 state 는 부모 컴포넌트에 있고, 자식은 부모의 state 를 그저 보여주는 역할만 하는 것이다.

`HomeView` 의 `<template>` 을 다음과 같이 변경한다.

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

const name = ref("ssafy");
const age = ref(10);
</script>

<template>
  <h1>Home</h1>
  <div>{{ name }}</div>
  <div>{{ age }}</div>
  <HomeChild :nameProp="name" :ageProp="age" />
</template>
```

무엇이 사용되었는지 보이는가? `v-bind` 가 사용되었다.

- `v-bind` 는 두 가지 사용 목적이 있다.
  1. 태그의 애트리뷰트를(속성) 변수화 시키는 목적

     ex) `<a :href="URL">`
  2. 자식에게 state 전달 목적

자식인 `HomeChild` 에선 다음과 같이 받는다.

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

const props = defineProps({
  nameProp: String,
  ageProp: Number,
});
</script>

<template>
  <h2>Child</h2>
  <div>{{ nameProp }}</div>
  <div>{{ ageProp }}</div>
  <GrandChild />
</template>
```

`defineProps` 를 사용해 `props` 를 정의한다. `props` 는 부모로부터 내려받은 state 의 모음을 뜻한다. `defineProps` 의 아규먼트로 객체 하나를 정의하며, 각각의 내려받은 prop 이름을 키로 적고, 값으로 대문자로 시작하는 타입을 적는다.

그리고 `<template>` 에서 해당 prop 을 보여주도록 하자.

결과를 보면 알 수 있듯이, 부모의 state 를 잘 받아온 것을 알 수 있다.

- `defineProps` 는 따로 import 없이 바로 사용 가능한데, 이를 어려운말로 컴파일타임 매크로라고도 한다.
- prop 으로 받은 state 는 자식 컴포넌트로 복사된 것인가? 아니다. 실제 state 는 부모인 `HomeView` 에 있을 뿐이다. `HomeChild` 는 그저 부모의 state 를 가져다가 보여 주기만 할 뿐이지, 실제 state 는 존재하지 않는다. 따라서 복사가 아니다.

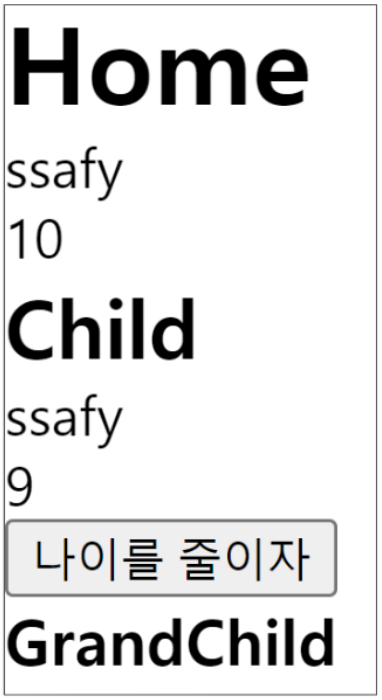
만약, `HomeChild` 에 버튼을 달아서, `ageProp` 을 변경하려고 하면 어떤 일이 발생할까?

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

const props = defineProps({
  nameProp: String,
  ageProp: Number,
});
</script>

<template>
  <h2>Child</h2>
  <div>{{ nameProp }}</div>
  <div>{{ ageProp }}</div>
  <button @click="ageProp = 9">나이를 줄이자</button>
  <GrandChild />
</template>
```

함수를 쓰지 않고, 클릭 이벤트 호출 시 간단히 값을 바꾸는 구문을 작성했다.



자식 prop 은 변경되었으나, 원래 부모 state 는 변경되지 않았다.

하나 기억해야 할 원칙이 있다.

자식은 부모의 state 를 함부로 수정 할 수 없다 !

할 수 없다는 것을 확인 했으니까, 원래 코드로 돌려놓고, 새로고침 한 번 해주자.

만약, 부모로부터 받은 값을 자식에게 전달 해 주고 싶다면?

즉, 중계 역할을 하고 싶다면 어떻게 하면 될까?

그냥 주면 된다. 이번엔 `GrandChild` 로 `name` 만 넘겨줄 것이다.

먼저, 부모인 `HomeChild` 다.

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

const props = defineProps({
  nameProp: String,
  ageProp: Number,
});
</script>

<template>
  <h2>Child</h2>
  <div>{{ nameProp }}</div>
  <div>{{ ageProp }}</div>
  <GrandChild :nameProp="nameProp" />
</template>
```

이번에도 이름을 `nameProp` 으로 통일 시켰다. 실제 개발 시 prop 의 혼동을 피하기 위해 자주 쓰이는 기법이다.

즉, 반드시 자식의 이름을 다르게 지을 필요는 없다.

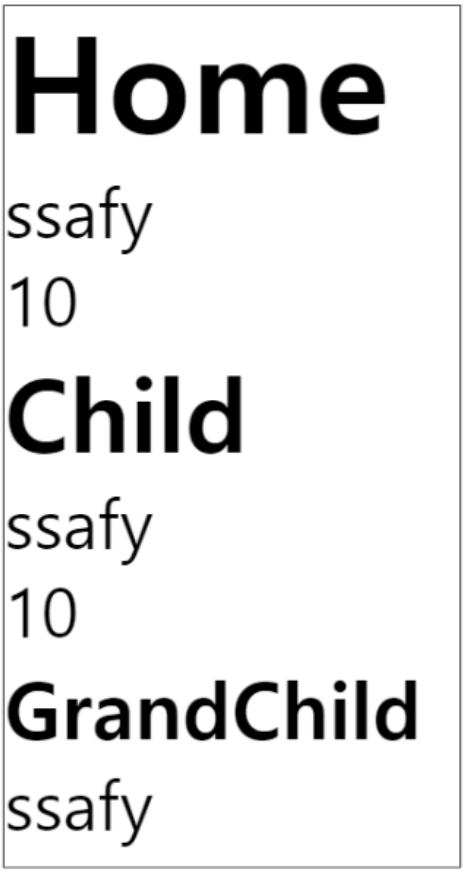
다음, 자식인 `GrandChild` 다.

```
<script setup>
const props = defineProps({
  nameProp: String,
});
</script>

<template>
```

```
  <h3>GrandChild</h3>
  <div>{{ nameProp }}</div>
</template>
```

결과는 다음과 같다.



참고로, HTML 문법은 대소문자를 구분하지 않는다.

Vue 의 권장사항을 조금만 더 받아 들이자면, 다음과 같이 HTML 에 해당하는 문법은 kebab-case 로 코드 수정할 수 있다.

이 경우, HTML 에서는 kebab-case `name-prop` 이지만,

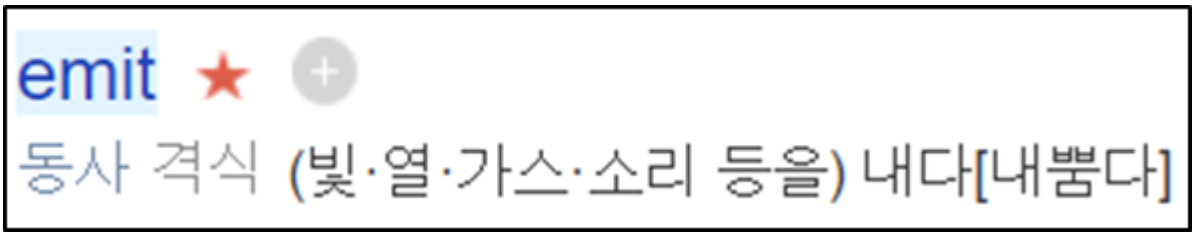
JavaScript 에서는 camelCase `nameProp` 으로 받아들인다.

```
const props = defineProps({
  nameProp: String,
});
```

여기까지 부모가 자식에게 값을 전달하는 것을 해 보았다면 이번에는

자식이 부모에게 요청하는 것을 해 보자. 바로 emit 이다.

emit



Vue 에서 emit 은 자식 컴포넌트가 부모에게 "내뿜는" 이벤트를 뜻한다.

쉽게 말하면, 엄마 나 이거 해줘!

우리는 자식이 부모의 state 를 함부로 변경할 수 없다는 것을 실험을 통해 배웠다.

부모의 state 는 오로지 부모만 변경 가능하다.

만약 자식이 부모의 state 를 변경하고 싶다면, 부모에게 해달라고 요청을 해야 한다. (emit)

먼저 `HomeChild` 컴포넌트 (중간 컴포넌트) 를 다음과 같이 변경한다.

```
<script setup>
// ...

const emit = defineEmits(["emitTest"]);

function changeName() {
  emit("emitTest");
}
</script>

<template>
  <!-- ... -->
  <button @click="changeName">부모에게 요청하기</button>
```

```
  <GrandChild :name-prop="nameProp" />
</template>
```

`props` 와 마찬가지로, `defineEmits` 의 리턴값으로 함수 하나를 `emit` 으로 받는다.

`defineEmits` 의 아규먼트는 하나의 배열이며, 정의할 emit 이벤트를 쭉 써주면 된다. 배열 이라는 점에서 알 수 있듯, 하나의 컴포넌트에서 emit 이벤트는 하나가 아닐 수 있다.

그리고 `changeName` 함수를 선언했다. 이 함수는 버튼을 누르면 실행된다.

해당 함수에서 `emit` 함수를 호출하되, 정의된 emit event 중 하나의 이름을 첫 번째 아규먼트로 입력한다.

- `defineEmits` 도 `defineProps` 와 마찬가지로 import 없이 바로 사용 가능한 컴파일타임 매크로다.

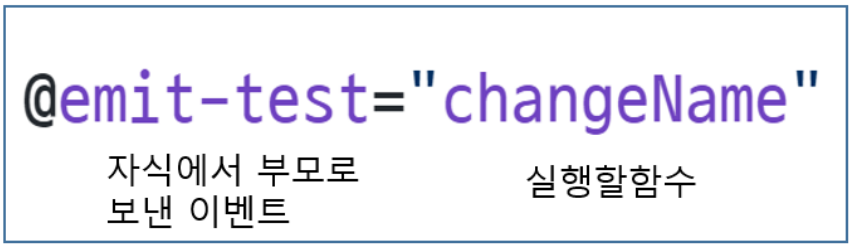
다음, 최상위 컴포넌트 `HomeView` 를 다음과 같이 수정한다.

```
<script setup>
// ...
function changeName() {
  name.value = "싸피";
}
</script>

<template>
  <!-- ... -->
  <HomeChild
    :name-prop="name"
    :age-prop="age"
    @emit-test="changeName"
  />
</template>
```

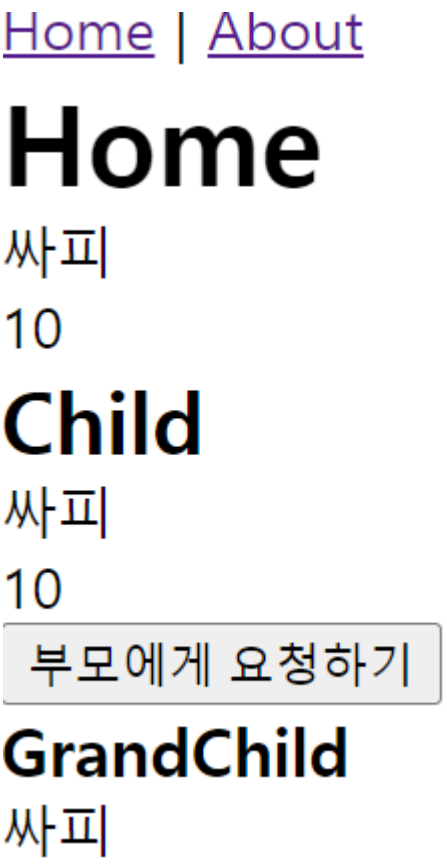
무엇이 사용되었는지 보이는가? `v-on` 이 사용되었다.

- `v-on` 은 두 가지 사용 목적이 있다.
  1. 이벤트 발생 시 실행시킬 함수 지정 ex) click, change, keyup …
  2. 자식에서 `defineEmits` 를 통해 지정한 이벤트 발생 시 실행시킬 함수 지정



prop 과 마찬가지로, HTML 문법은 kebab-case 로, JavaScript 문법은 camelCase 로 작성한다.

그리고 부모에게 요청하기를 클릭 후 `changeName` 함수가 실행되며, `name` state 는 `ssafy` 에서 `싸피` 로 변경 되는 것을 확인할 수 있다.



이렇게 `HomeChild` (자식)에서 `HomeView` (부모)에게 state 값을 바꿔 달라는 요청을 통해서 ssafy 가 싸피로 변경 되었다.

emit 이벤트에 아규먼트를 딸려 보낼 수도 있다. 실습 해보자.

중간 컴포넌트 `HomeChild` 의 `changeName` 함수를 다음과 같이 수정한다.

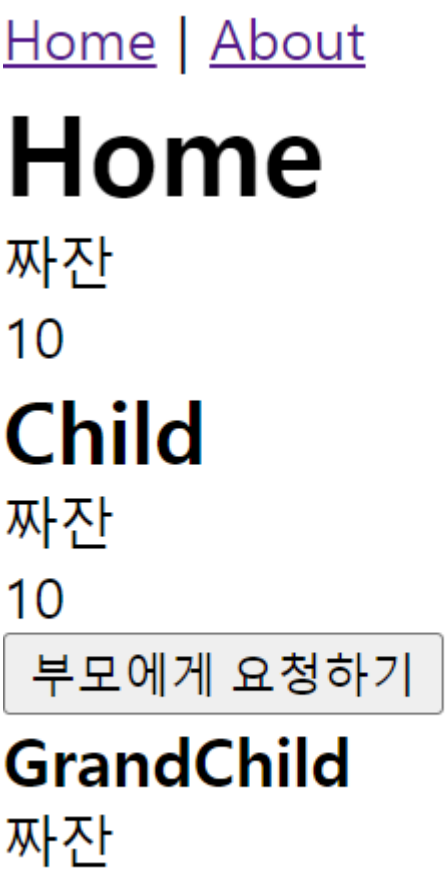
```
function changeName() {
  emit("emitTest", "짜잔");
}
```

위와 같이, `emit` 함수의 두번째 아규먼트부터 콤마 `,` 로 구분해, 딸려 보낼 값을 적는다.

그리고, 최상위 컴포넌트 `HomeView` 의 `changeName` 함수를 다음과 같이 수정한다.

```
function changeName(newName) {
  name.value = newName;
}
```

`name` state 는 `ssafy` 에서 `짜잔`으로 변경 되는 것을 확인할 수 있다.



이번엔 최하위 컴포넌트 `GrandChild` 에서, 최상위 컴포넌트 `HomeView` 의 `age` state 를 변경해보고자 한다.

먼저 `GrandChild` 컴포넌트다.

```
<script setup>
const props = defineProps({
  nameProp: String,
});
const emit = defineEmits(["changeAge"]);

function changeAge(newAge) {
  emit("changeAge", newAge);
}
</script>

<template>
  <h3>GrandChild</h3>
  <div>{{ nameProp }}</div>
  <button @click="changeAge(1)">한살로 돌아가자</button>
</template>
```

두 가지를 집고 넘어가자

- `changeAge` 함수에 아규먼트가 필요하다면 예제와 같이 `changeAge(1)` 로 쓸 수 있다.
- 함수명 `changeAge` 와 이벤트명 `changeAge` 가 같다. HTML 태그 안에서 이벤트 이름을 쓴다면 `change-age` 가 될 것이다. 앱이 커지면 props/emit 구조가 상당히 복잡해지기 때문에 이처럼 연결된 모든 이름을 통일 시키는 기법은 실무에서도 매우 자주 쓰인다.

다음, 중간 컴포넌트 `HomeChild` 로 와서 곰곰히 생각해 보니, `age` state 는 `HomeChild` 의 state 가 아니라 `HomeView` 컴포넌트의 state 이므로 한번 더 올려야 한다. 이것을 반드시 기억해두자.

`HomeChild` 는 다음과 같다.

```
<script setup>
// ...
const emit = defineEmits(["emitTest", "changeAge"]);
</script>

<template>
  <!-- ... -->

  <GrandChild
    :name-prop="nameProp"
    @change-age="emit('changeAge', $event)"
  />
```

```
</template>
```

우선, `defineEmits` 매크로에 `changeAge` emit 이벤트를 추가한다.

그리고 `v-on` 구문을 보면, `emit('changeAge', $event)` 라고 작성된 것을 볼 수 있는데, 이것은 이벤트를 받은 그대로, 부모로 넘긴다는 뜻이다.

- 따옴표에 주의하자. 큰 따옴표 `""` 가 바깥을 감싸면, 안에선 작은 따옴표 `''` 를 사용해야 한다.

최상위 컴포넌트 `HomeView` 는 다음과 같다.

```
<script setup>
// ...
function changeAge(newAge) {
  age.value = newAge;
}
</script>

<template>
  <!-- ... -->
  <HomeChild
    :name-prop="name"
    :age-prop="age"
    @emit-test="changeName"
    @change-age="changeAge"
  />
</template>
```

`age` state 는 10 에서 1 로 바뀜을 확인할 수 있다. <끝>