# Day5 Props and Emit

| | |
|---|---|
| 자료 | Vue |
| 구분 | Vue |
| 과목 | |

새로 프로젝트를 생성을 하고

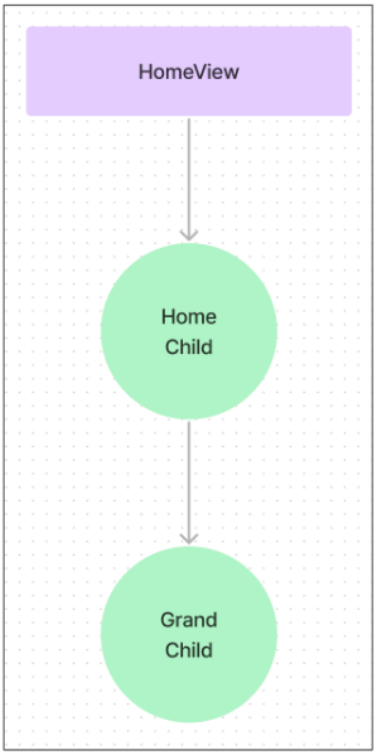
의미없는 파일들은 미리 삭제를 한 후에 아래 실습을 진행하자.

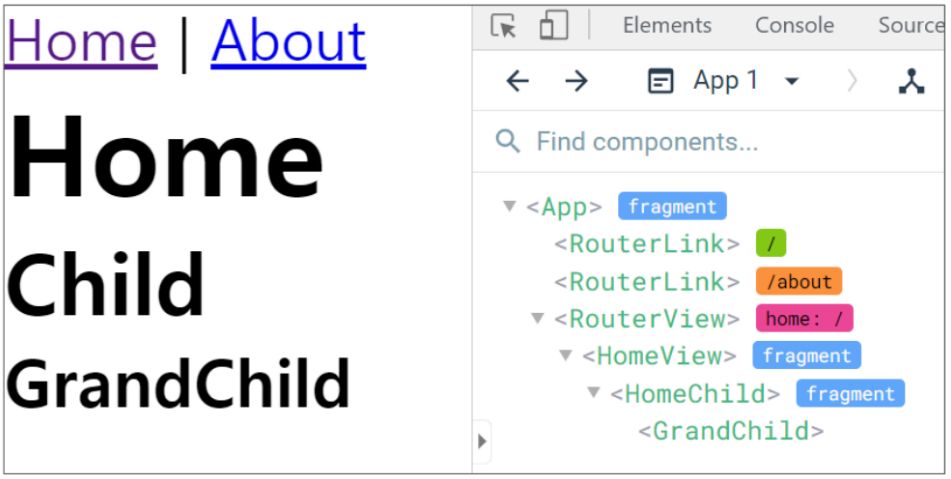
## Props

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

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

도전: 다음 구조로 컴포넌트를 만들자.




그리고 `HomeView` 컴포넌트를 다음과 같이 변경한다.

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

<script setup>
```

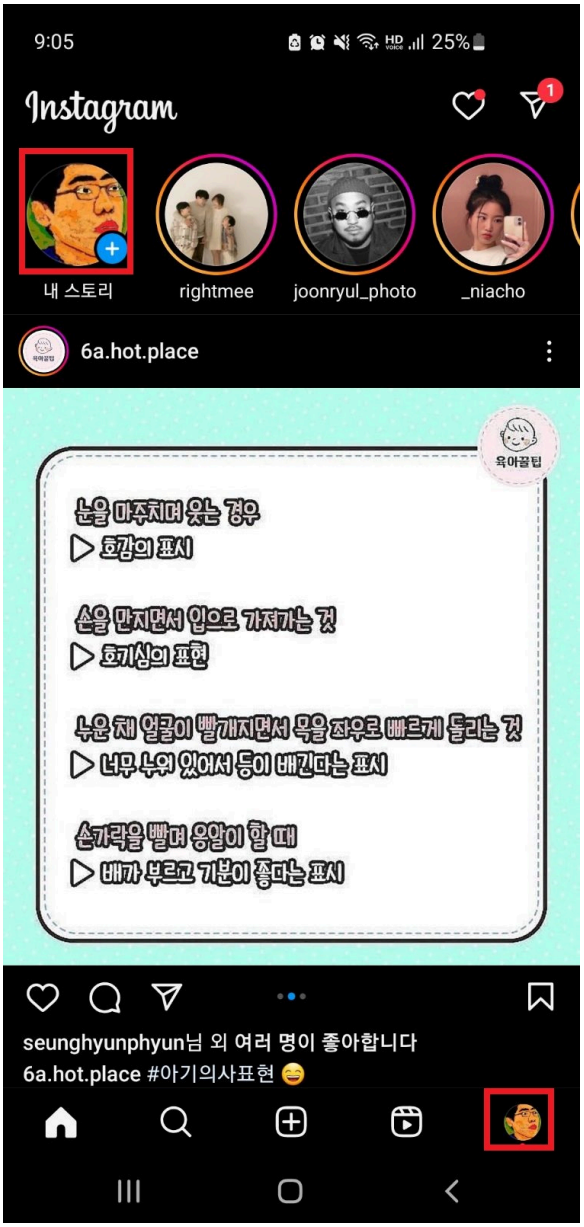
```vue
import { ref } from "vue";
import HomeChild from "@/components/HomeChild.vue";

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


부모 컴포넌트가 가지고 있는 데이터값을

자식 컴포넌트에서도 사용하고 싶다면, 해당 데이터 값을 전달 해주면 된다.

현재 state(데이터)는 어디에 있는가? `HomeView` 에 있다. 여기서 우리가 하고자 하는 것은 자식 컴포넌트인 `HomeChild` 로, `name` 과 `age` 값을 내려 보내는 것이다.

왜 이러한 기능이 필요할까? 인스타그램이나 페이스북에 접속하면, 한 화면에 프로필 사진 또는 이름이 중복되어서 여러 곳에 표시되는 영역을 확인 할 수 있을 것이다.




.

여러 컴포넌트에서 중복으로 사용하는 데이터는 각 컴포넌트에서 개별적으로 state를 선언해 관리할 수도 있다.

하지만 이런 방식은 데이터가 변경될 때 문제가 발생할 수 있다. 예를 들어, 동일한 데이터를 여러 컴포넌트가 각각 관리하고 있다면, 하나의 값이 바뀔 때마다 모든 컴포넌트에서 별도로 요청을 보내야 하고, 그 결과 데이터의 일관성이 깨질 위험이 있다.

대표적인 사례로는 사용자 정보나 프로필 사진처럼 여러 컴포넌트에서 자주 참조되는 데이터를 들 수 있다. 이런 데이터가 한 컴포넌트에서 변경되었을 때, 다른 컴포넌트들이 그 변경

사항을 인식하지 못하면 동기화 문제가 생기고, 때로는 불필요한 중복 업데이트로 인해 성능 저하가 발생할 수도 있다.

이런 문제를 방지하기 위해, 중요하고 반복적으로 사용되는 state는 부모 컴포넌트에서 통합 관리하는 것이 바람직 하다고 볼 수 있다. 자식 컴포넌트는 그 데이터를 직접 관리하지 않고, 부모로부터 전달받아 화면에 표시하는 역할만 담당하게 하면, 데이터의 일관성을 유지하면서 효율적인 상태 관리를 할 수 있다.

`HomeView` 의 `<template>` 을 다음과 같이 변경 해보자.

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

  <!-- 이렇게 (수정)하자 -->
  <HomeChild :nameProp="name" :ageProp="age" />
</template>

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

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

무엇이 사용되었는지 보이는가? `v-bind` 가 사용되었다.

- `v-bind` 는 두 가지 사용 목적이 있다.
  1. 태그의 속성을 (attribute) 변수화 시키는 목적! 그리고

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


태그의 속성의 값을 state의 값으로 적용할 때 `v-bind` 를 사용 했었다.

그런데 한가지 기능이 더 있다. 바로 자식 컴포넌트에 state를 전달할 때에도 `v-bind` 를 사용한다.

자식 컴포넌트인 `HomeChild` 에서는 다음과 같이 받아보자.

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

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

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

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

그리고 `<template>` 에서 해당 prop 의 값을 보여줬다.


위 결과를 보면 알 수 있듯이, 부모의 state 를 자식 컴포넌트에서 잘 받아온 것을 확인 할 수 있다.

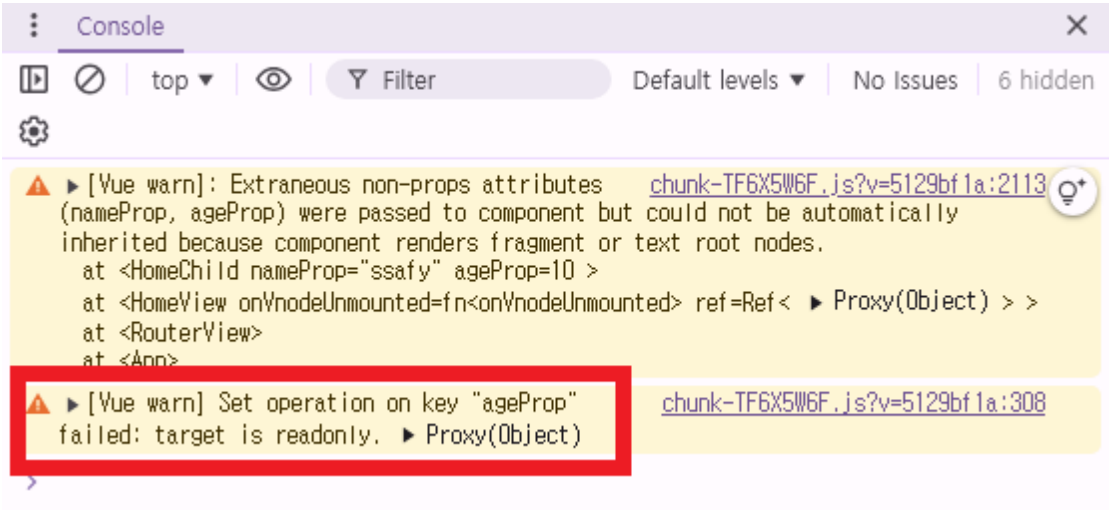
- `defineProps` 는 따로 import 없이 바로 사용 가능한데, 그 이유는 어려운말을 조금 사용하자면compile-time macro 라는 것 때문에 가능한 것이다.
- 자식이 부모로 부터 받은 prop 들은 전부 읽기전용다. 따라서 변경이 불가능 하다! (만약 변경을 하고자 한다면 조금 이따가 학습 할 Emit을 사용해야 한다.)

  (자식이 부모로 부터 받은 값을 함부로 변경않된다)
- prop 으로 받은 state 는 자식 컴포넌트로 복사 된 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 복사의 개념이 아니다.

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

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

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

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




전달받은 Props는 읽기 전용이다. 따라서 변경을 하고자 하면 위와 같이 에러가 난다.

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

만약, 부모로부터 받은 값을, 다시 또 나의 자식(손자)에게 전달 해 주고 싶다면?

즉, 여기서 `HomeChild` 가 부모 컴포넌트인 `HomeView` 로 부터 전달받은 name을 `HomeChild` 의 자식컴포넌트인 `GrandChild` 로 전달 하는 중계 역할을 하려면 어떻게 해야 할까?

그냥 주면 된다. 이번엔 `GrandChild` 로 `name` 만 넘겨주는 것을 실습 해 보자.

먼저, 이번에 부모 역할을 할 `HomeChild` 이다.

```vue
<template>
  <h2>Child</h2>
  <div>{{ nameProp }}</div>
```

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

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

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

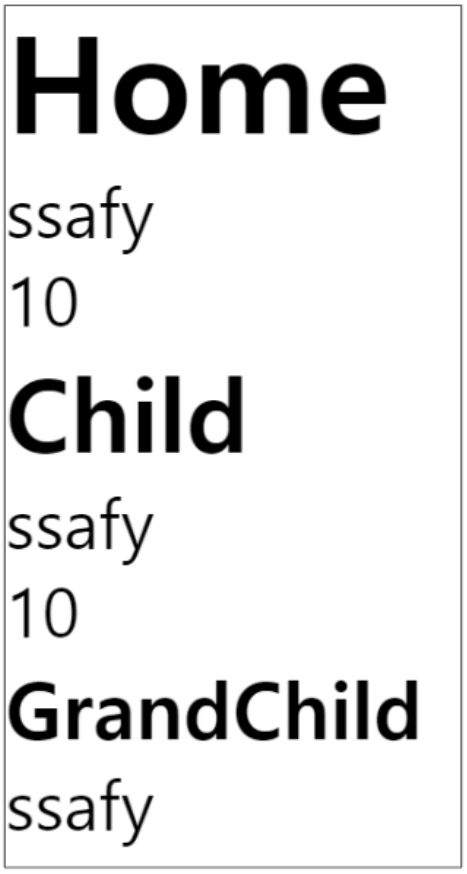
이번에도 이름을 `nameProp` 로 통일 시켰다. 실제 개발 시 prop 의 혼동을 피하기 위해 이름을 바꾸지 않고 동일하게 사용을 하는 편이다.

굳이 자식에게 전달할 prop의 이름을 다르게 지을 필요는 없다.

다음, 자식인 `GrandChild` 다.

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

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

결과는 다음과 같다.




Vue의 권장사항을 한번 살펴보도록 하자.

- Tempate (HTML 부분)의 속성에는 케밥-케이스 사용을 권장하며
- Script (Vue 로직 구현)에는 카멜케이스 사용을 권장한다.

Vue 내부적으로 템플릿에는 케밥케이스로 작성된 속성 이름을 —> Script에서 카멜케이스로

자동 변환한다.

예를들면 `HomeChild` 에서 작성된 코드를 아래와 같이 수정해보자.

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

```
</template>

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

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

```
<GrandChild :name-Prop="nameProp" :age-Prop="ageProp" />
```

위 코드는 Vue의 권장사항에 따라 Template 속성을 name-Prop 와 같이 '케밥케이스(kebab-case)'로 작성했다.


그 다음 내부적으로 이를 카멜 케이스로 변환하여 JavaScript에서 처리한다.

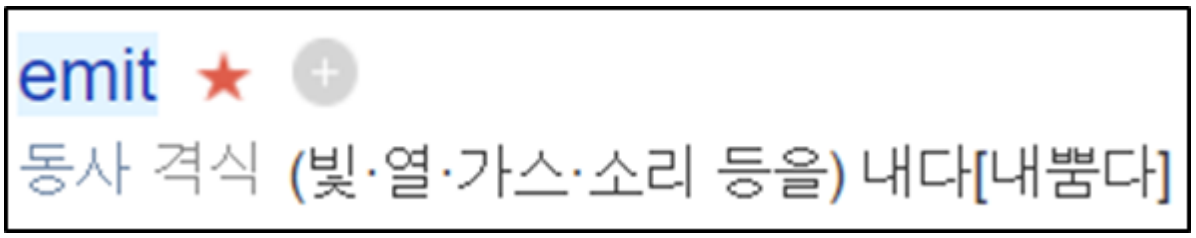
따라서 자식 컴포넌트 `GrandChild` 에서는 다음과 같이 '카멜케이스(camelCase)'로 전달 받을 수 있다.

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

</script>
```

여기까지 부모가 자식에게 값을 전달하는 것을 해 보았다. 아주 쉽다.

자, 이제는 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 자식컴포넌트에서 부모컴포넌트의 state를 함부로 변경할 수 없다는 것을 실험을 통해 배웠다. ( 부모의 state 는 오로지 부모만 변경 가능하다. )

만약 자식이 부모의 state를 변경하고자 한다면, 부모컴포넌트에게 state를 변경해달라고 요청을 해야 한다. 이 요청을 Emit 이라고 한다.

이제부터 HomeChild 는 HomeView 에게 state를 변경해 달라고 요청할 것이다.

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

```
<template>
  <h2>Child</h2>
  <div>{{ nameProp }}</div>
  <div>{{ ageProp }}</div>
  <button @click="changeName">부모에게 요청하기</button>
  <GrandChild :name-Prop="nameProp" :age-Prop="ageProp" />

</template>

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

const props = defineProps({
```

```
  nameProp: String,
  ageProp: Number,
});

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

function changeName() {
  emit("emitTest"); //부모에게 이벤트 전달 요청!
}
</script>
```

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

`defineEmits` 의 argument는 하나의 배열이며, 정의할 emit 이벤트를 쭉 써주면 된다. 배열이라는 점에서 알 수 있듯, 하나의 컴포넌트에서 emit 이벤트는 하나가 아닐 수도 있다.

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

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

눈여겨 볼 점은 `defineEmits` 도 `defineProps` 와 마찬가지로 import 없이 바로 사용 가능하다.

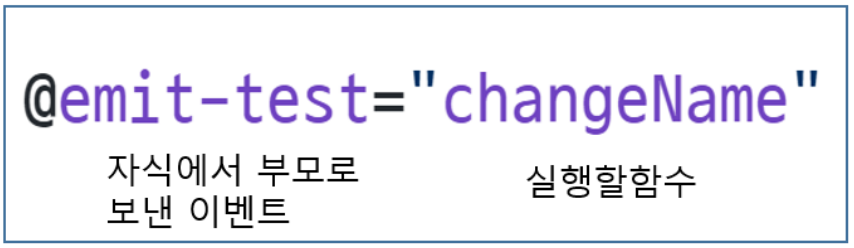
다음, 최상위 컴포넌트 `HomeView` 를 다음과 같이 수정한다.

```
<template>
  <h1>Home</h1>
  <div>{{ name }}</div>
  <div>{{ age }}</div>
  <HomeChild
    :name-prop="name"
    :age-prop="age"
    @emit-test="changeName"
  />
</template>

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

```
const name = ref("ssafy");
const age = ref(10);

function changeName() {
  name.value = "싸피";
}

</script>
```

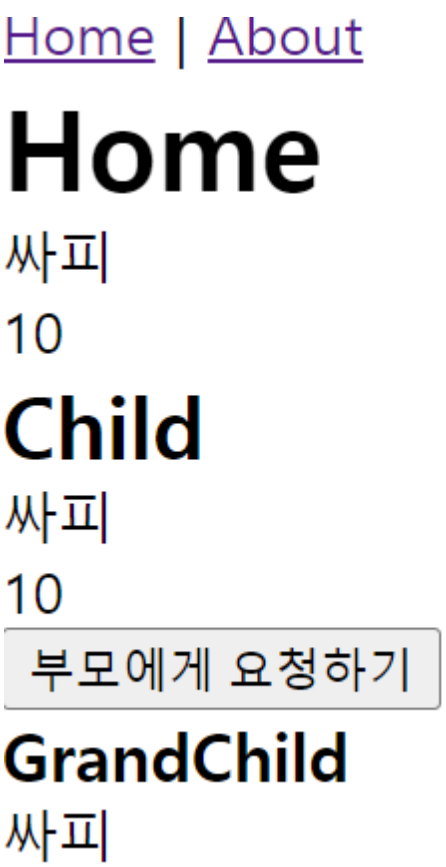
무엇이 사용되었는지 보이는가? `v-on` 이 사용되었다.

- `v-on` 은 두 가지 사용 목적이 있다.
  1. 이벤트 발생 시 실행시킬 함수 지정 ex) click, change, keyup 등등..
  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 arguments를 같이 보낼 수도 있다. 실습 해보자.

중간 컴포넌트 `HomeChild` 의 `changeName` 함수를 다음과 같이 수정 해보자.

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

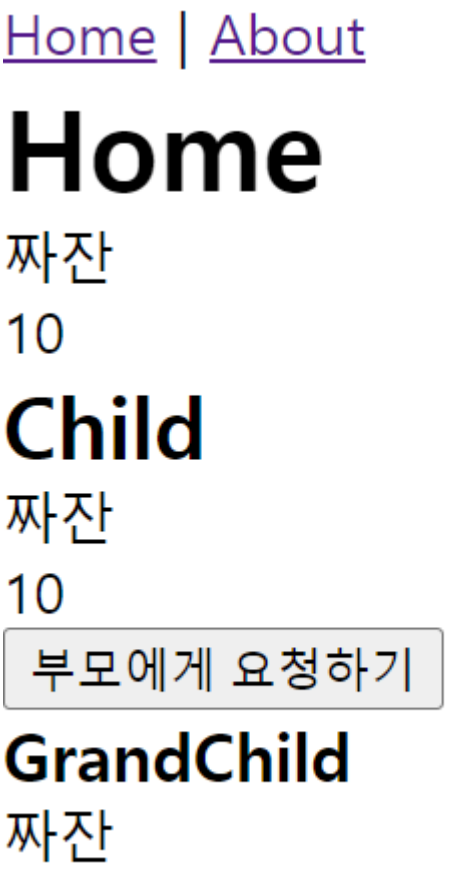
위와 같이, `emit` 함수의 두번째 arguments부터 콤마 `,` 로 구분해, 함께 보낼 값을 적는다.

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.




자, 이번에는 최하위 컴포넌트 `GrandChild` 에서, 최상위 컴포넌트 `HomeView` 의 `age` state 를 변경 해보는것을 실습 해보자.

먼저 `GrandChild` 컴포넌트다.

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

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

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

아래 두 가지를 정도만 확인해 보자

- `changeAge` 함수에 argument가 필요하다면 예제와 같이 `changeAge(1)` 로 쓸 수도 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.

그 다음, 중간 컴포넌트 `HomeChild` 로 와서 곰곰히 생각해 보니, `age` state 는 `HomeChild` 의 state 가 아니라 `HomeView` 컴포넌트(최상위부모)의 state 이므로 한번 더 올려야 한다. 이러한 것들에 잘 유의하자.

`HomeChild` 는 다음과 같다.

```
<template>
  <h2>Child</h2>
  <div>{{ nameProp }}</div>
  <div>{{ ageProp }}</div>
  <button @click="changeName">부모에게 요청하기</button>
  <GrandChild :name-Prop="nameProp" :age-Prop="ageProp"
              @change-age="emit('changeAge', $event)"/>


</template>

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

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

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

function changeName() {
  emit("emitTest", "짜잔"); //무모에게 이벤트 전달 요청!
}
</script>
```

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

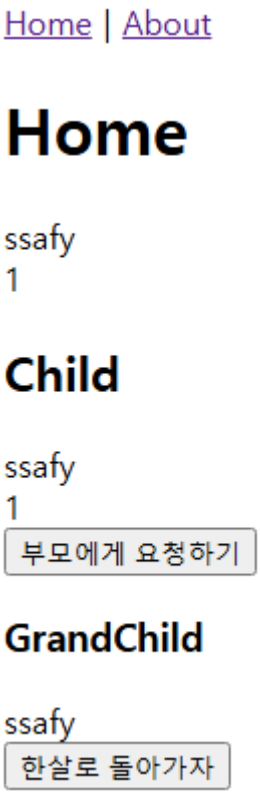
그리고 `v-on` 구문을 보면, `emit('changeAge', $event)` 라고 작성된 것을 볼 수 있는데, 이것은 이벤트를 받은 그대로, 부모로 넘긴다는 뜻이다.

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

최상위 컴포넌트 `HomeView` 는 다음과 같다.

```
<template>
  <h1>Home</h1>
  <div>{{ name }}</div>
  <div>{{ age }}</div>
  <HomeChild
    :name-prop="name"
    :age-prop="age"
    @emit-test="changeName"
    @change-age="changeAge"
  />
</template>

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

```
const name = ref("ssafy");
const age = ref(10);

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




`age` state 는 10 에서 1 로 바뀜을 확인할 수 있다.

<끝>