KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

단일 파일 컴포넌트를 이용한 Vue 애플리케이션 개발

[KB] IT's Your Life

KB 국민은행

단일 파일 컴포넌트 Single File Component

- 컴포넌트 하나를 .vue 파일 하나에 작성
- 한 파일에 컴포넌트 구성을 위해 템플릿, 스크립트, 스타일을 모두 포함
- 컴포넌트 단위로 관심사를 분리
- 파일 확장자는 .vue

번들링(Bundling)

- 여러 모듈을 묶어서 하나 또는 몇 개의 모듈 파일로 만드는 과정
- webpack, vite 등

Vite 기반의 도구

- npm init vue

  → vite 기반으로 프로젝트 구성

  프로젝트 구성을 위한 몇가지 질문에 답변해야 함

프로젝트 만들기

```
> npm init vue

Vue.js - The Progressive JavaScript Framework

√ Project name: ... test-vue-app
√ 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

Scaffolding project in ch07\test-vue-app...

Done. Now run:

  cd test-vue-app
  npm install
  npm run dev
```

디폴트 프로젝트 골격

```
test-vue-app
    |   package.json
    |   index.html
    |   jsconfig.json
    |   .gitignore
    |   vite.config.js
    |   README.md
    |
    ├──.vscode
    |       extensions.json
    |
    ├──public
    |       favicon.ico
    |
    └──src
        |   App.vue
        |   main.js
        |
        ├──assets
        |       base.css
        |       logo.svg
        |       main.css
        |
        └──components
            |   HelloWorld.vue
            |   TheWelcome.vue
            |   WelcomeItem.vue
            |
            └──icons
                    IconCommunity.vue
                    IconDocumentation.vue
                    IconEcosystem.vue
                    IconSupport.vue
                    IconTooling.vue
```

package.json

```
{
  "name": "test-vue-app",
  "version": "0.0.0",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "vue": "^3.3.11"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^4.5.2",
    "vite": "^5.0.10"
  }
}
```

패키지 설치

- cd <프로젝트>
- npm install

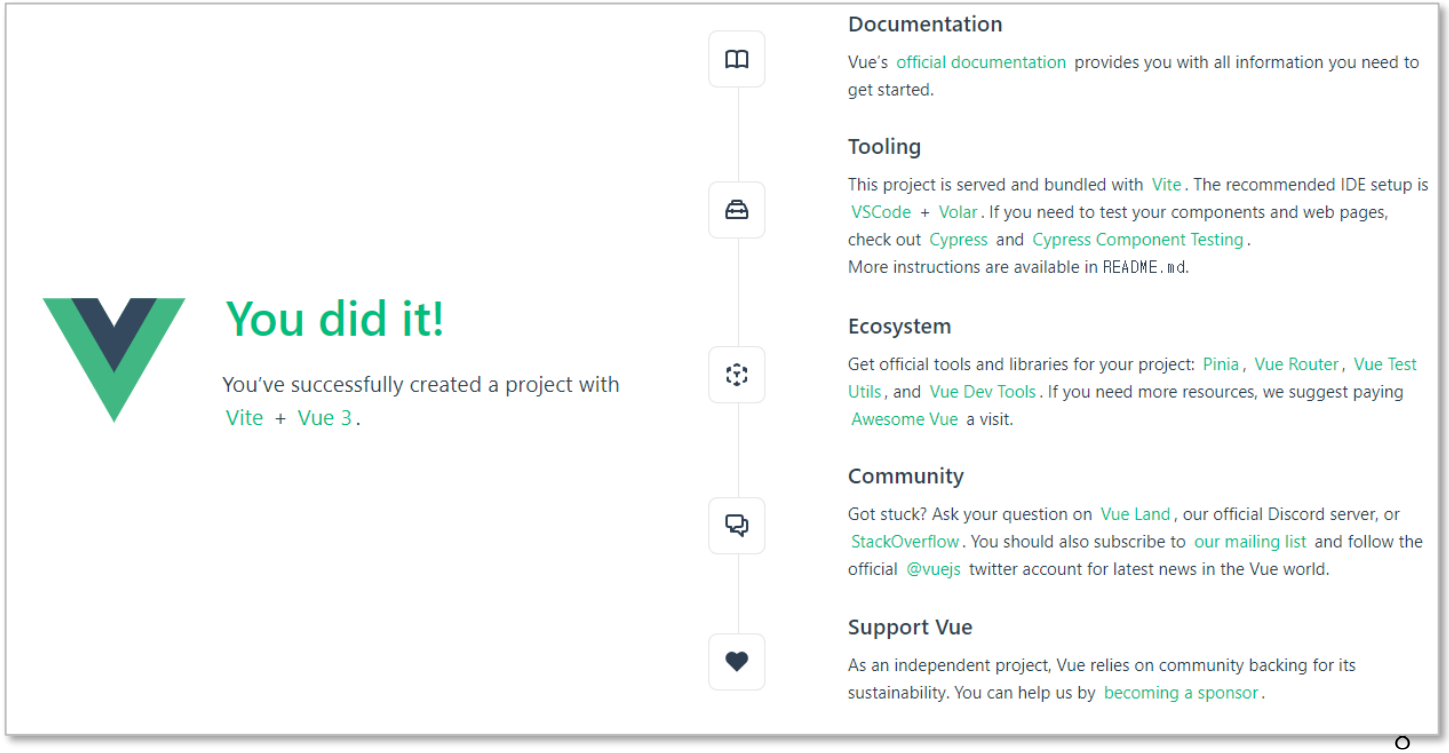
프로젝트 관리 명령어

- 빌드 명령어 : npm run build
- 개발 서버 시작 명령어: npm run dev
- 미리보기 명령어: npm run preview

기동

- npm run dev

```
> test-vue-app@0.0.0 dev
> vite

Re-optimizing dependencies because vite config has changed

  VITE v5.0.10  ready in 2759 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```



모듈 경로 표기법 설정

- @기호를 src 폴더로 인식시켜 절대 경로 사용
  → vite.config.js에서 설정

- vscode에서 @기호 인식시키기
  → jsconfig.json에서 설정

vite.config.js

```
import { fileURLToPath, URL } from 'node:url'

import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [
    vue(),
  ],
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src', import.meta.url))
    }
  }
})
```

jsconfig.json

```
{
  "compilerOptions": {
    "paths": {
      "@/*": ["./src/*"]
    }
  },
  "exclude": ["node_modules", "dist"]
}
```

핵심 파일들 – 진입점

```
test-vue-app
    |   package.json
    |   index.html
    |
    └──src
        |   App.vue
        |   main.js
        |
        ├──assets
        |       base.css
        |       logo.svg
        |       main.css
        |
        └──components
            |   HelloWorld.vue
            |   TheWelcome.vue
            |   WelcomeItem.vue
            |
            └──icons
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vite App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

src/main.js

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

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

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

단일 파일 컴포넌트 살펴보기

```
test-vue-app
    |   package.json
    |   index.html
    |
    └──src
        |   App.vue
        |   main.js
        |
        ├──assets
        |       base.css
        |       logo.svg
        |       main.css
        |
        └──components
            |   HelloWorld.vue
            |   TheWelcome.vue
            |   WelcomeItem.vue
            |
            └──icons
```

App.vue

```
<script setup>
import HelloWorld from './components/HelloWorld.vue'
import TheWelcome from './components/TheWelcome.vue'
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="./assets/logo.svg" width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="You did it!" />
    </div>
  </header>

  <main>
    <TheWelcome />
  </main>
</template>
```

App.vue

```
<style scoped>
header {
  line-height: 1.5;
}

.logo {
  display: block;
  margin: 0 auto 2rem;
}

@media (min-width: 1024px) {
  header {
    display: flex;
    place-items: center;
    padding-right: calc(var(--section-gap) / 2);
  }

  …
}
</style>
```

간단한 단일 컴포넌트 작성과 사용

CheckboxItem 컴포넌트 만들기

- src/components 하위 파일/디렉토리 삭제
- src/components/CheckboxItem.vue 파일 추가

src/main.js 수정

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

import { createApp } from 'vue';
import App from './App.vue';

createApp(App).mount('#app');
```

src/components/CheckboxItem.vue

```
<template>
  <li><input type="checkbox" v-model="checked" />옵션1</li>
</template>

<script>
export default {
  name: 'CheckboxItem',
  data() {
    return {
      checked: false,
    };
  },
};
</script>
```

컴포넌트 등록

- 컴포넌트를 사용하기 위해서는 먼저 등록을 해야 함
- 등록 방법
  - 전역 방법
    - 한번만 등록하면 어디에서 든 바로 사용가능
    - 사용하지 않는 컴포넌트에서도 포함되므로 패키징시 파일이 커질 수 있음

  - 지역 방법
    - 사용하는 컴포넌트내에서 매번 컴포넌트 등록

    → 권장

전역 컴포넌트 등록 방법

- src/main.js에서 만든 루트 Vue 인스턴스에 등록

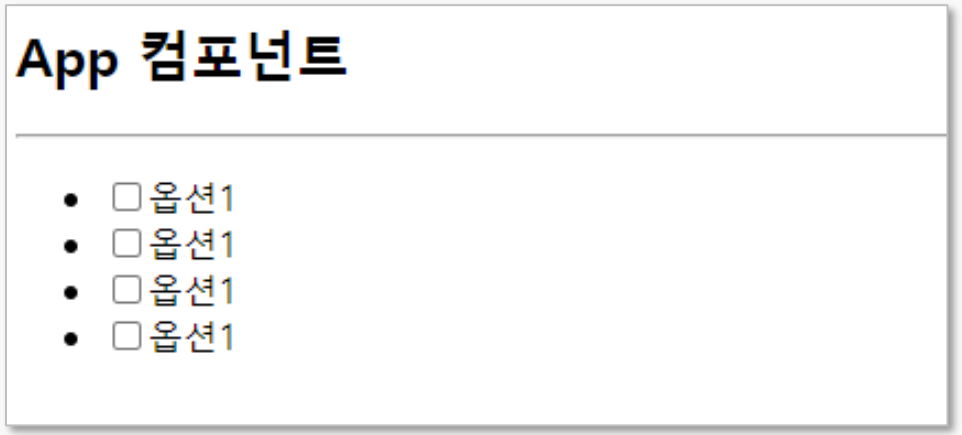
```
import { createApp } from 'vue';
import App from './App.vue';
import CheckboxItem from './components/CheckboxItem.vue';

createApp(App)
    .component('CheckboxItem', CheckboxItem)
    .mount('#app');
```

지역 컴포넌트로 등록하는 방법

- 사용할 컴포넌트 import
- components 옵션에 해당 컴포넌트 지정

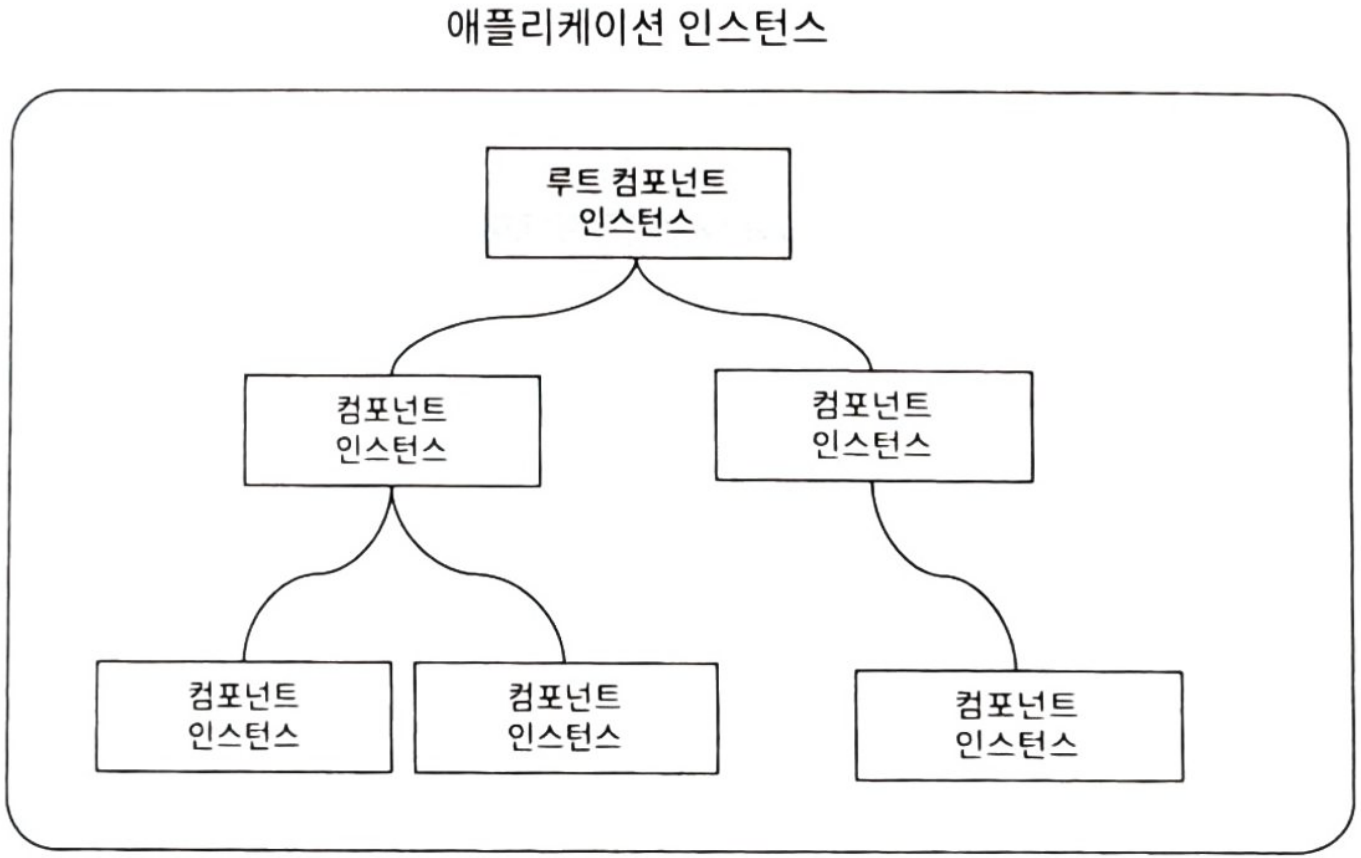
src/App.vue

```
<template>
  <div>
    <h2>App 컴포넌트</h2>
    <hr />
    <ul>
      <CheckboxItem />
      <CheckboxItem />
      <CheckboxItem />
      <CheckboxItem />
    </ul>
  </div>
</template>

<script>
import CheckboxItem from './components/CheckboxItem.vue';

export default {
  name: 'App',
  components: { CheckboxItem },
};
</script>
```



복잡한 화면의 Vue 애플리케이션

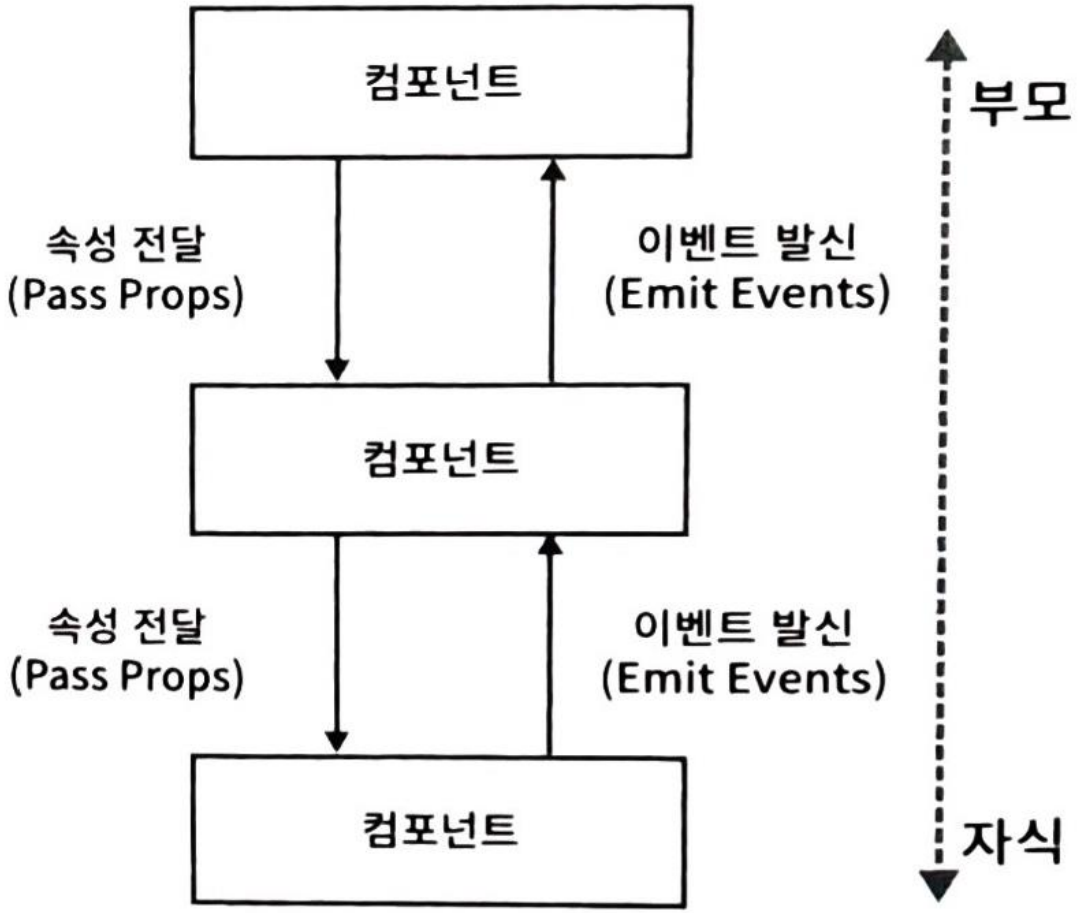
- 여러 컴포넌트들을 조합하여 개발
- 컴포넌트들은 부모-자식 관계로 트리 구조를 형성



컴포넌트간 정보 전달

- 부모 → 자식
  - 속성(Props)

- 자식 → 부모
  - 이벤트 발신(emit)
  - 부모의 이벤트 핸들러에서 정보 처리



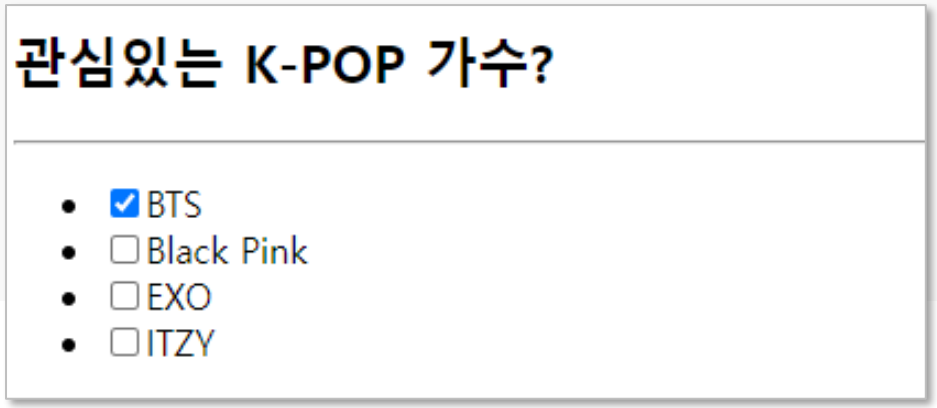
속성을 이용한 정보 전달

- 자식 컴포넌트는 props 옵션으로 속성 정의
- 부모 컴포넌트는 v-bind를 이용해 자식 컴포넌트의 속성에 정보를 전달

- 속성으로 전달받은 데이터는 변경 불가 !! → 읽기 전용(read only)
- 부모에서 속성 값을 변경 시 → 자식은 자동으로 다시 렌더링됨

src/components/CheckboxItem.vue

```
<template>
  <li><input type="checkbox" :checked="checked" />{{ name }}</li>
</template>

<script>
export default {
  name: 'CheckboxItem',
  props: ['name', 'checked'], // 속성명을 배열로 정의
};
</script>
```

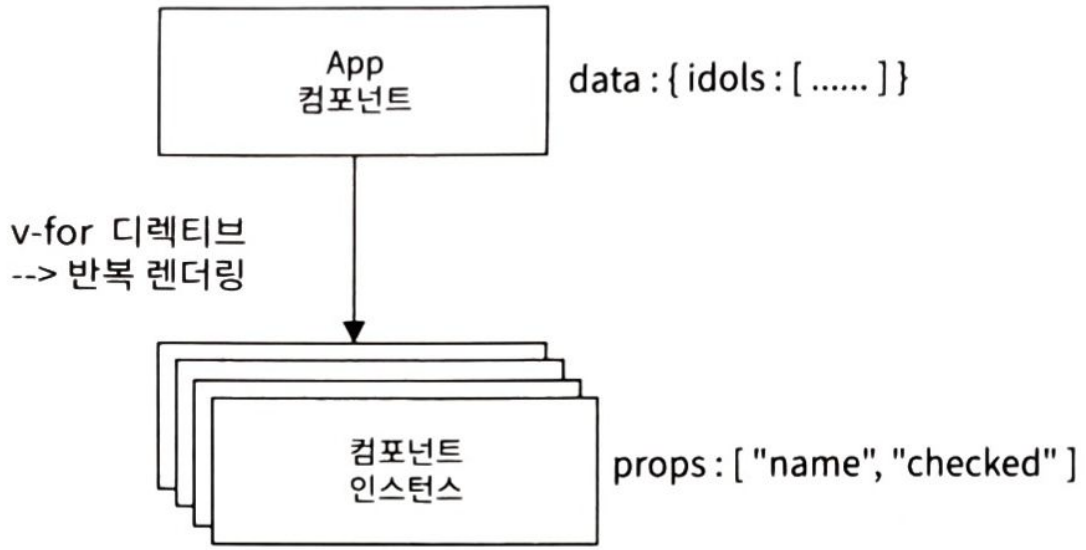


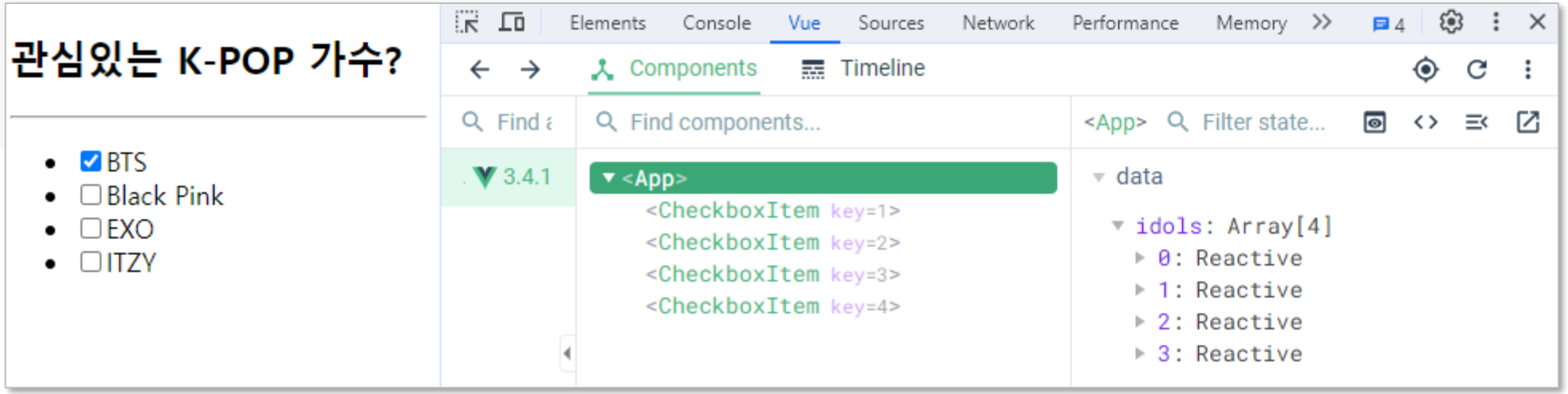
src/App.vue

```
<script>
import CheckboxItem from './components/CheckboxItem.vue';
export default {
  name: 'App',
  components: { CheckboxItem },
  data() {
    return {
      idols: [
        { id: 1, name: 'BTS', checked: true },
        { id: 2, name: 'Black Pink', checked: false },
        { id: 3, name: 'EXO', checked: false },
        { id: 4, name: 'ITZY', checked: false },
      ],
    };
  },
};
</script>
```

src/App.vue

```
<template>
  <div>
    <h2>관심있는 K-POP 가수?</h2>
    <hr />
    <ul>
      <CheckboxItem
        v-for="idol in idols"
        :key="idol.id"
        :name="idol.name"
        :checked="idol.checked"
      />
    </ul>
  </div>
</template>
```





속성을 이용해 객체 전달하기

- 전달할 속성이 많은 경우 → 객체 하나로 통합해서 전달
- 객체의 속성이 컴포넌트에서 사용할 속성에 대응

src/CheckboxItem2.vue

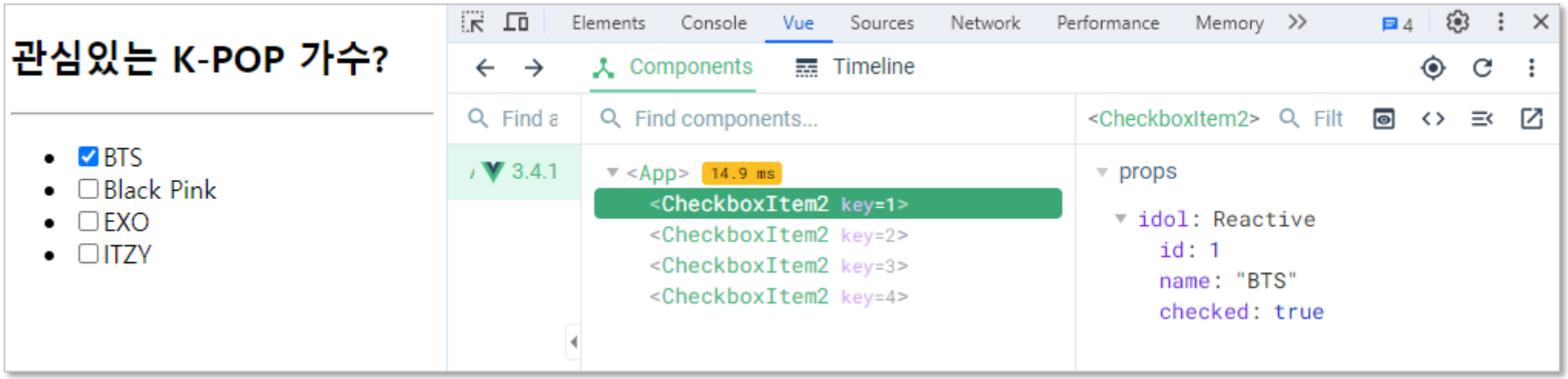
```
<template>
  <li><input type="checkbox" :checked="idol.checked" />{{ idol.name }}</li>
</template>

<script>
export default {
  name: 'CheckboxItem2',
  props: ['idol'],
};
</script>
```

src/App2.vue

```
<template>
  <div>
    <h2>관심있는 K-POP 가수?</h2>
    <hr />
    <ul>
      <CheckboxItem v-for="idol in idols" :key="idol.id" :idol="idol" />
    </ul>
  </div>
</template>

<script>
import CheckboxItem from './components/CheckboxItem2.vue';
export default {
  name: 'App',
  …
};
</script>
```

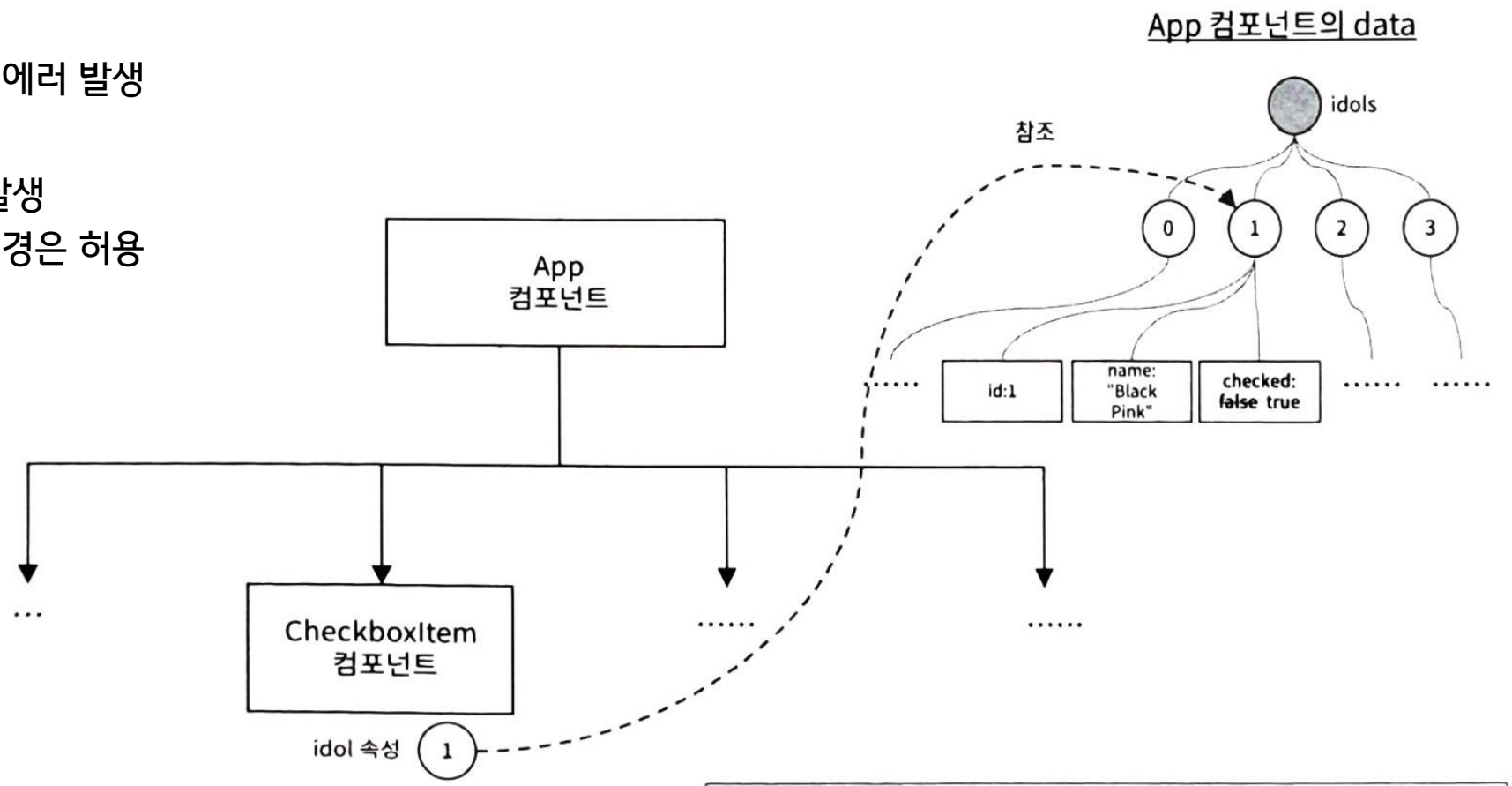
src/main.js

```
import { createApp } from 'vue';
// import App from './App.vue';
import App from './App2.vue';

createApp(App).mount('#app');
```



속성 변경

- 기본값 속성
  - 자식 컴포넌트에서 수정 시 에러 발생
- 참조타입 속성
  - 참조 자체를 수정 시 에러 발생
  - 참조 객체의 내부 속성값 변경은 허용
    → 비권장



** this.idol.checked = true로 변경하면?

- this.idol 속성의 메모리 주소는 변경되지 않음
- idol 속성의 메모리 주소를 참조해 객체 내부의 속성만 변경됨.

속성의 유효성 검증

- 속성에 대한 엄격한 유효성 검증 필요시
  → 배열이 아닌 객체 형태로 속성을 정의

```
export default {
  ......
  props : {
      속성명1 : 타입명,
      속성명2 : [타입명1, 타입명2],
      속성명3 : {
         type : 타입명,
         required : [true/false, 기본값:false],
         default : [기본값 또는 기본값을 리턴하는 함수, 기본값:undefined ]
      },
      ......
  }
}
```

속성의 유효성 검증

- type에 생성자 함수 지정
  - String
  - Number
  - Boolean
  - Array
  - Object
  - Date
  - Function
  - Symbol

src/components/CheckboxItem.vue

```
<template>
  <li><input type="checkbox" :checked="checked" />{{ id }} - {{ name }}</li>
</template>

<script>
export default {
  name: 'CheckboxItem',
  props: {
    id: [Number, String], // Number 또는 String 타입
    name: String,         // String 타입
    checked: {
      type: Boolean,      // Boolean 타입
      required: false,    // 옵션
      default: false,     // 생략시, 기본값은 false
    },
  },
};
</script>
```

사용자 정의 이벤트를 이용한 정보 전달

이벤트

- 자식 컴포넌트가 부모 컴포넌트에게 정보를 전달하는 방법

자식 컴포넌트가 이벤트를 발신(emit events)

- 컴포넌트 인스턴스의 $emit() 메서드 이용

$emit('이벤트명', [값])

부모 컴포넌트는 v-on 디렉티브로 이벤트 수신

사용자 정의 이벤트를 이용한 정보 전달

- 자식 컴포넌트에서

```
this.$emit('event-name', eventArgs1, eventArgs2, ...)
```

- 부모 컴포넌트에서

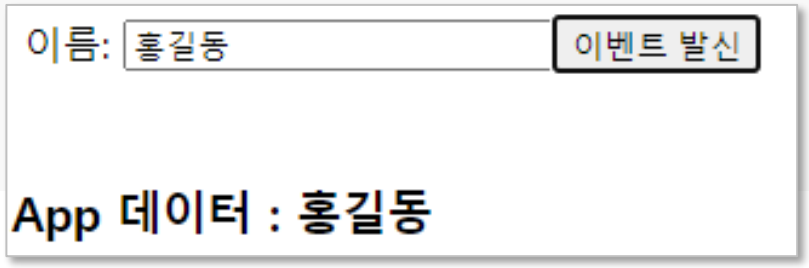
```
<child-component @event-name="handlerMethod" />

methods: {
  handlerMethod(eventArgs1, eventArgs2, ...) {
    // 전달받은 아규먼트로 처리할 코드 작성
  }
}
```

src/components/InputName.vue

```
<template>
  <div style="border: solid 1x gray; padding: 5px">
    이름: <input type="text" v-model="name" />
    <button @click="$emit('nameChanged', { name })">이벤트 발신</button>
  </div>
</template>

<script>
export default {
  name: 'InputName',
  data() {
    return { name: '' };
  },
};
</script>
```

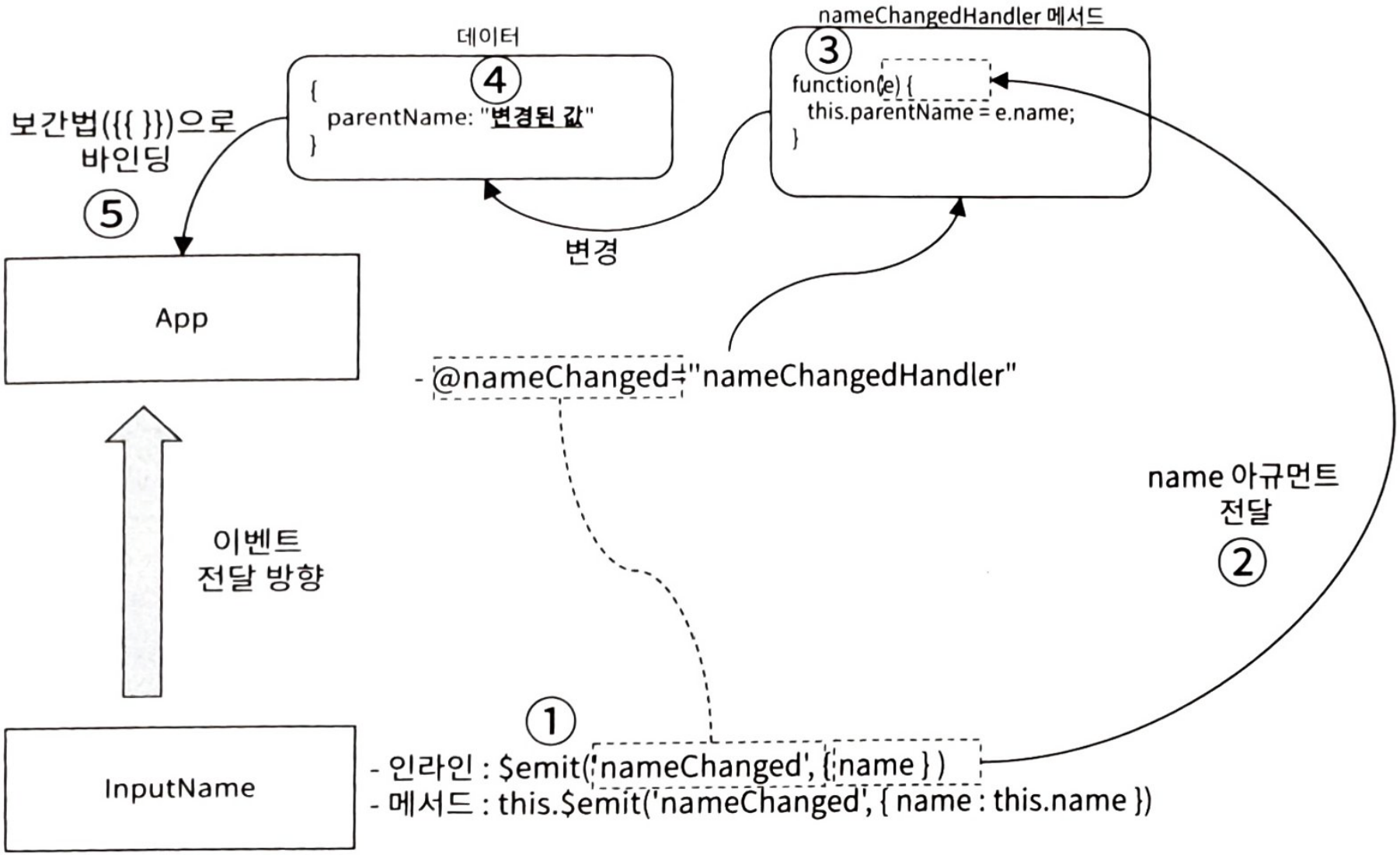
src/App4.vue

```
<template>
  <div>
    <InputName @nameChanged="nameChangedHandler" />
    <br />
    <h3>App 데이터 : {{ parentName }}</h3>
  </div>
</template>
```

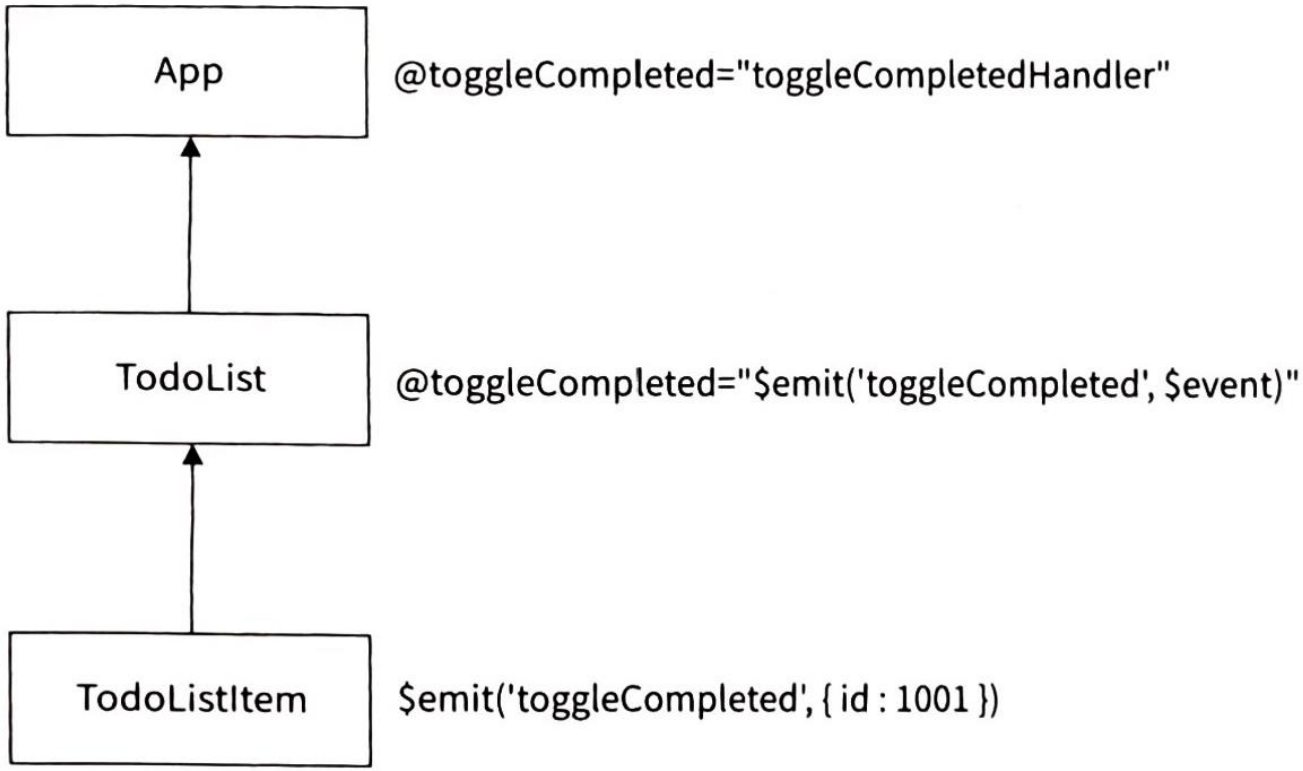



```
<script>
import InputName from './components/InputName.vue';

export default {
  name: 'App4',
  components: { InputName },
  data() {
    return {
      parentName: '',
    };
  },
  methods: {
    nameChangedHandler(e) {
      this.parentName = e.name;
    },
  },
};
</script>
```

이벤트 처리 과정


데이터
④
{
parentName: "변경된 값"
}
nameChangedHandler 메서드
③
function(e) {
this.parentName = e.name;
}
보간법({{ }})으로
바인딩
⑤
변경
App
- @nameChanged="nameChangedHandler"
name 아규먼트
전달
②
이벤트
전달 방향
①
InputName
- 인라인 : $emit('nameChanged', { name })
- 메서드 : this.$emit('nameChanged', { name : this.name })

부모-자식-손자 컴포넌트 계층 구조에서 이벤트 전송



이벤트 유효성 검증

emits 옵션 등록

- 발신하는 이벤트에 대한 유효성 검사를 수행

```
const Component = {
    ......
    emits : ["이벤트명1", "이벤트명2" ]
    .......
}
```

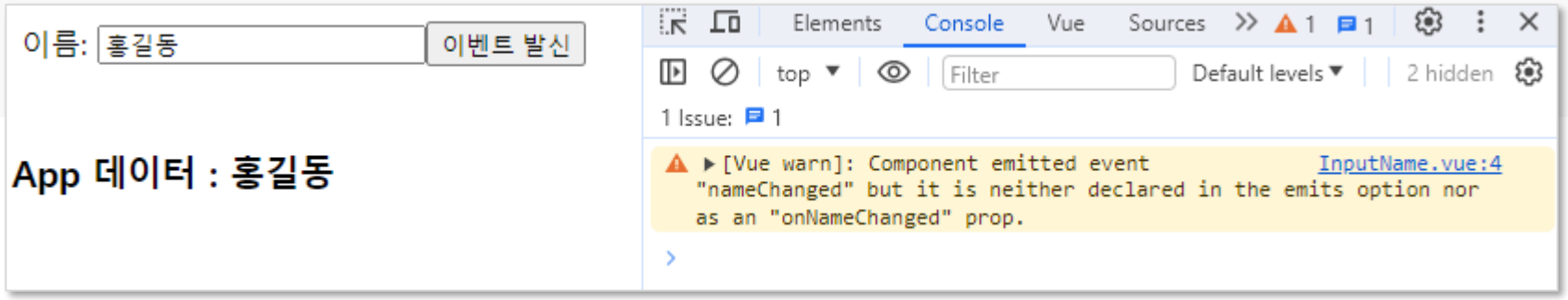
지정한 이벤트 이외의 이벤트 발생시 경고 출력

```
const Component = {
    ......
    emits : {
        이벤트명1 : (e) => {
            //true가 리턴되면 유효
            //false가 리턴되면 유효하지 않음
        },
        //유효성 검사 하지 않음
        이벤트명2 : null,
        ......
    },
    .......
}
```

이벤트 인자에 대한 검증
true 리턴 : 유효
false 리턴 : 경고 출력

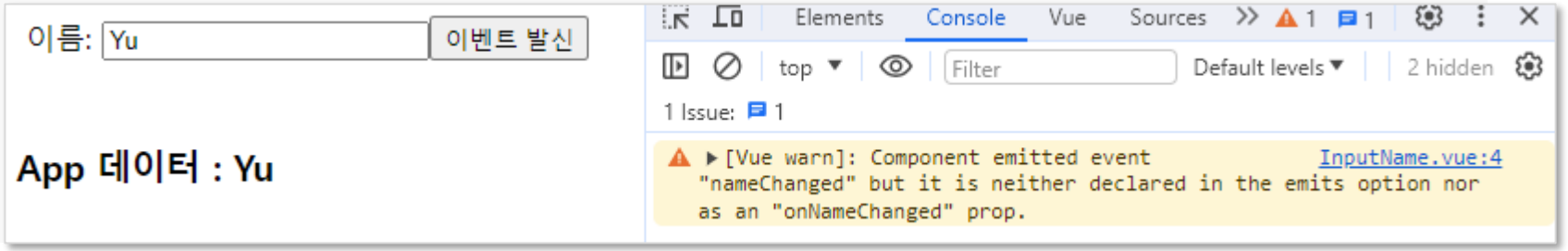
src/InputName.vue

```
<template>
  <div style="border: solid 1x gray; padding: 5px">
    이름: <input type="text" v-model="name" />
    <button @click="$emit('nameChanged', { name })">이벤트 발신</button>
  </div>
</template>

<script>
export default {
  name: 'InputName',
  emits: [ 'nameChanged1'],  // nameChanged1 이외의 이벤트 발생시 경고 출력
  data() {
    return { name: '' };
  },
};
</script>
```



src/InputName.vue

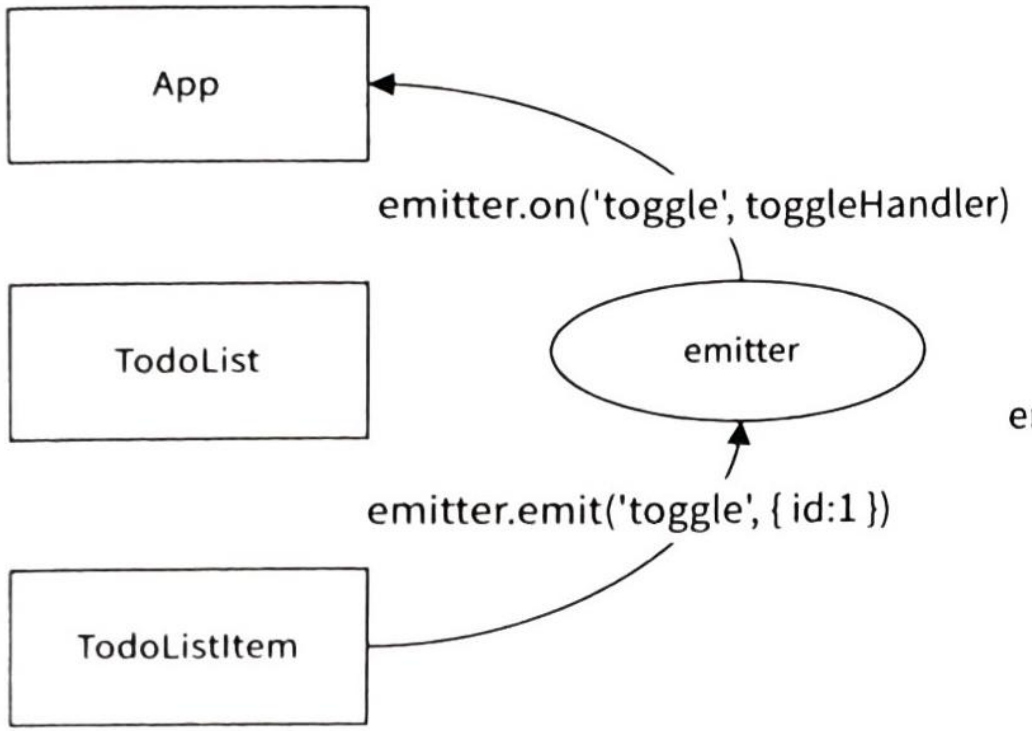
```
<template>…
</template>

<script>
export default {
  name: 'InputName',
  // emits: ['nameChanged1'], // nameChanged1 이외의 이벤트 발생시 경고창 출력
  emits: {
    namedChanged: (e) => {
      return e.name && typeof e.name === 'string' && e.name.trim().length >= 3
        ? true
        : false;
    },
  },
  data() {
    return { name: '' };
  },
};
</script>
```

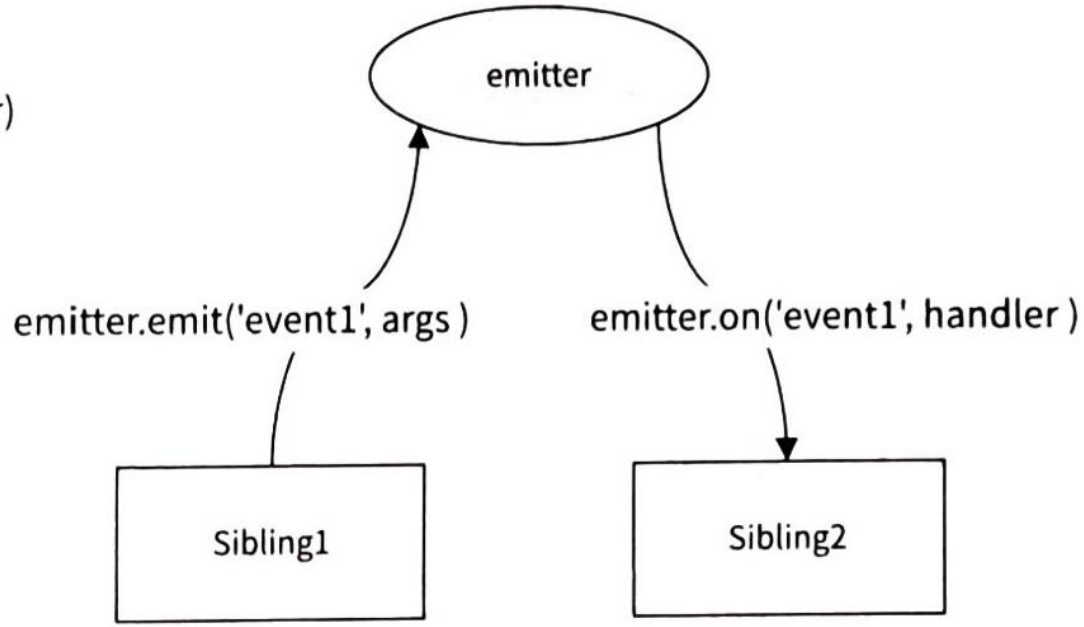


이벤트 에미터

- 부모-자식-손자와 같은 많은 계층에서 이벤트 전달은 매우 힘듦

→ 하나의 공유 이벤트 에미터(Event Emitter)를 만들어 특정 컴포넌트에게 이벤트를 전송





이벤트 에미터

- mitt 패키지 라이브러리 사용
  - https://github.com/developit/mitt
  - 설치

npm init --save mitt

- main.js에서 생성 후 전역 속성(globalProperties.emitter)에 등록

src/main.js

```
import { createApp } from 'vue';
import App from './App5.vue';
import mitt from 'mitt';

const emitter = mitt();
emitter.on('*', (type, e) => console.log(`## 이벤트 로그: ${type} ->`, e));

const app = createApp(App);
app.config.globalProperties.emitter = emitter;
app.mount('#app');
```

src/components/Sender.vue

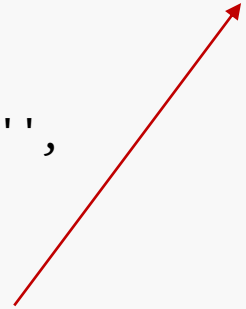
```
<template>
  <div style="border: solid 1px gray; margin: 5px; padding: 5px">
    <h2>Sender</h2>
    <button @click="sendMessage">이벤트 발신</button>
  </div>
</template>

<script>
export default {
  name: 'Sender',
  methods: {
    sendMessage() {
      this.emitter.emit('message', Date.now() + '에 발신된 메시지');
    },
  },
};
</script>
```

src/components/Receiver.vue

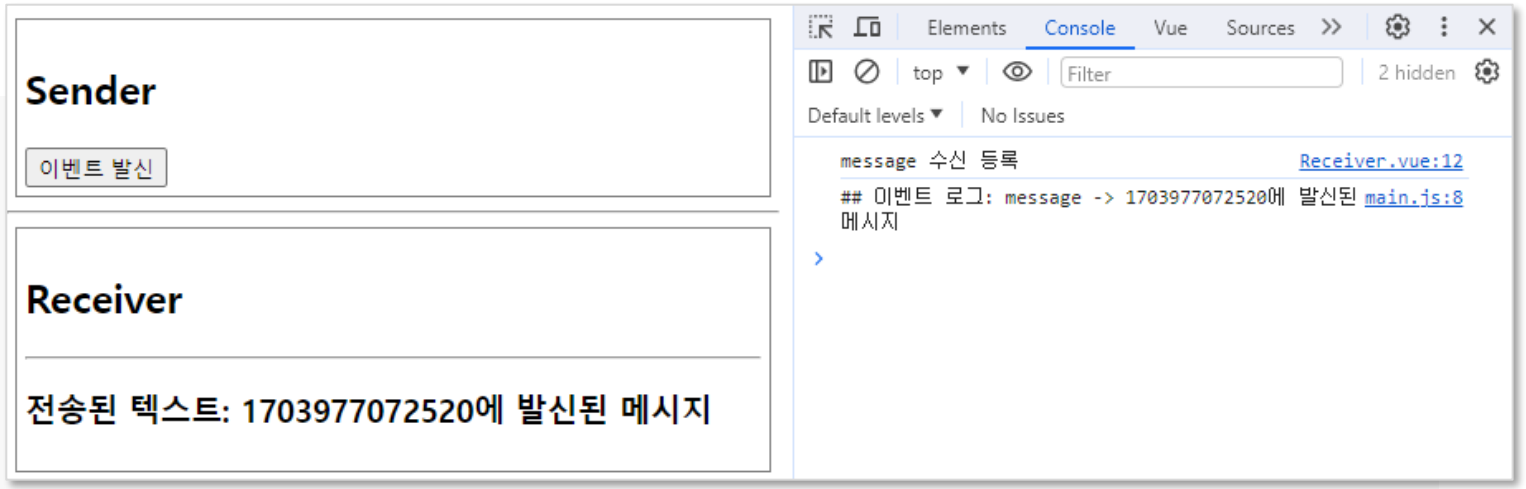
```
<template lang="">
  <div style="border: solid 1px gray; margin: 5px; padding: 5px">
    <h2>Receiver</h2>
    <hr />
    <h3>전송된 텍스트: {{ textMessage }}</h3>
  </div>
</template>
```

src/components/Receiver.vue

```
<script>
export default {
  name: 'Receiver',
  created() {
    this.emitter.on('message', this.receiveHandler);
  },
  data() {
    return {
      textMessage: '',
    };
  },
  methods: {
    receiveHandler(msg) {
      this.textMessage = msg;
    },
  },
};
</script>
```



src/App5.vue




```
<template>
  <div>
    <Sender />
    <hr />
    <Receiver />
  </div>
</template>

<script>
import Receiver from './components/Receiver.vue';
import Sender from './components/Sender.vue';

export default {
  name: 'App5',
  components: { Sender, Receiver },
};
</script>
```