

원쌤의 Vue.js 퀵스타트

7. 단일 파일 컴포넌트를 이용한 Vue 애플리케이션 개발



❖ 단일 파일 컴포넌트란?

- Single File Component
- 컴포넌트 하나를 .vue 파일 하나에 작성
- .vue 파일 하나에 템플릿, 스크립트, 스타일 정보를 담고 있음
 - 컴포넌트 단위로 관심사를 분리시킬 수 있음

❖ 단일 파일 컴포넌트 기반 개발을 위해 필요한 도구

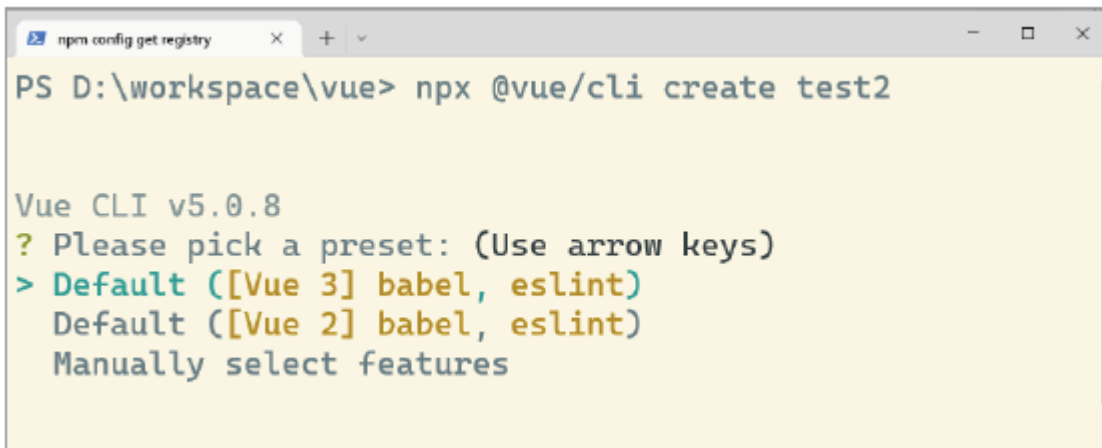
- 모듈 번들러 : webpack, rollup 등
- 트랜스파일러 : ES6, Typescript 코드를 트랜스파일할 수 있는 도구
- 이런것들을 함께 사용하도록 프로젝트를 구성하는 것은 쉬운 일이 아님
 - 프로젝트 구성을 자동으로 해주는 프로젝트 설정 도구를 사용해야 함

1. 프로젝트 설정 도구

❖ Vue CLI 도구

- webpack 기반의 Vue 공식 프로젝트 설정 도구였음

```
//프로젝트 생성 방법
//프리셋 선택 화면에서 [Vue 3]를 선택함
//프로젝트 생성 후 생성된 프로젝트 디렉터리로 이동하여 npm run serve 명령어로 실행함
npx @vue/cli create [프로젝트명]
cd [프로젝트명]
npm run serve
```



```
npm config get registry
PS D:\workspace\vue> npx @vue/cli create test2

Vue CLI v5.0.8
? Please pick a preset: (Use arrow keys)
> Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

❖Vue CLI 의 단점

- npm run serve로 개발용 웹서버를 실행할 때 긴 시간 필요
 - webpack 번들러를 사용하기 때문
 - 관련 모듈들의 덩치가 큼 --> 긴 다운로드 시간
 - 작성한 코드를 테스트 하기 위해 긴 시간 필요
 - 빌드 --> 번들링 --> 개발용 서버 구동

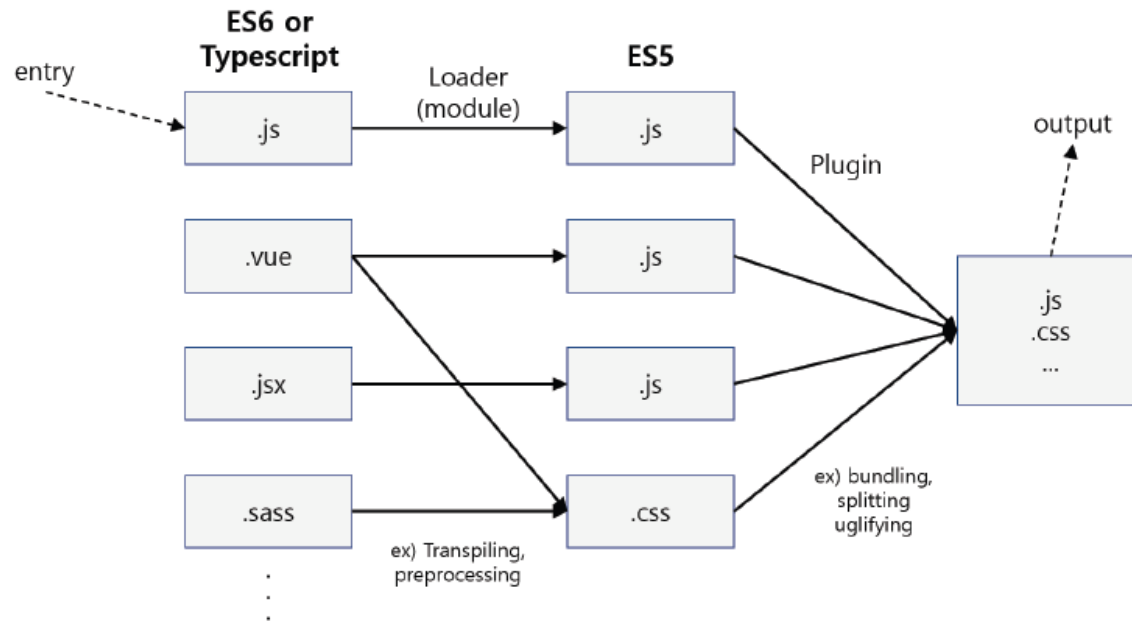
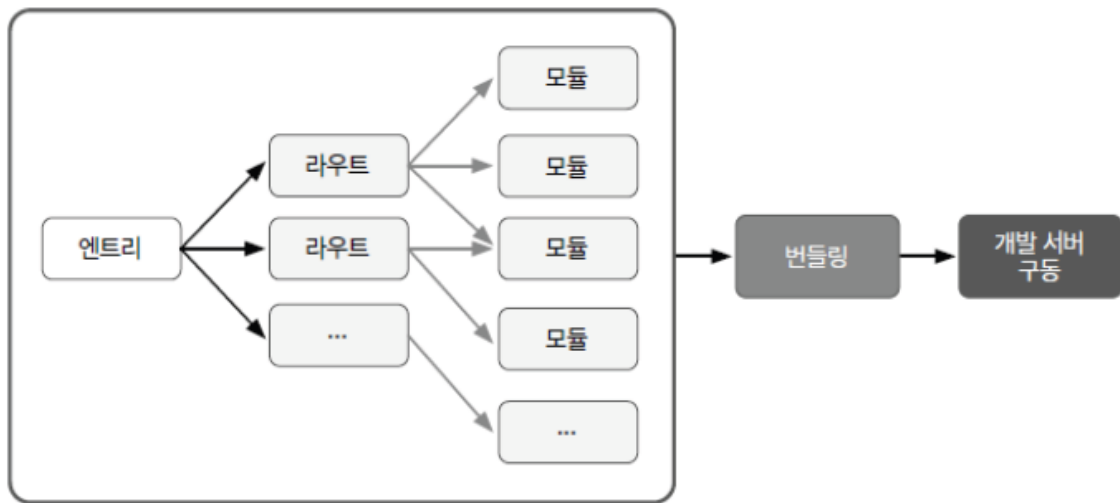


그림 07-02 Vue CLI로 생성한 프로젝트에서의 Webpack 빌드 과정

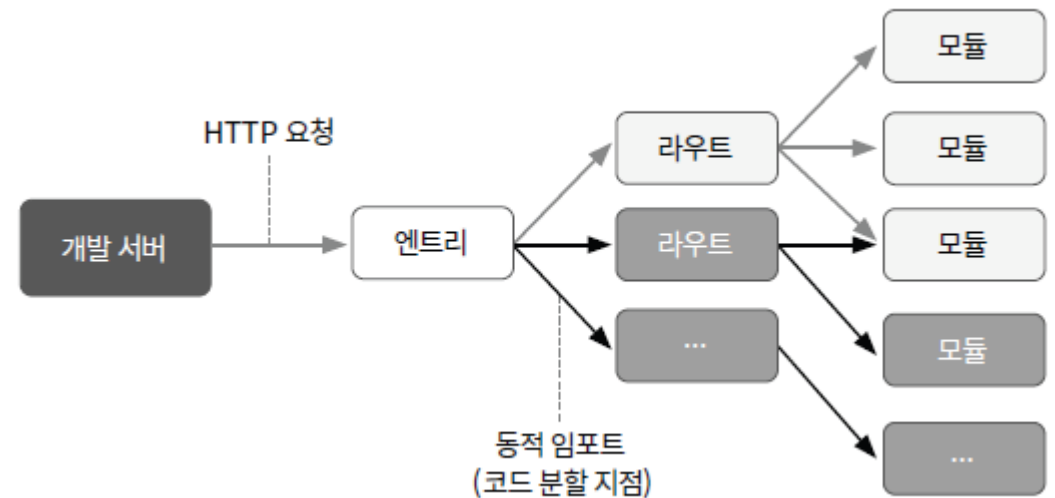
1. 프로젝트 설정 도구

❖ Vite 기반 도구

- "빠르다" --> 이름처럼 빌드와 개발서버 구동이 빠름
- Vue의 창시자인 Evan You가 만든 차세대 프론트엔드 개발 도구
- webpack(Javascript 기반도구) vs Vite의 ESBuild(GO로 만들어진 네이티브 도구)
- 특히 개발서버 구동 시간이 짧음
 - Native ESM이라는 브라우저의 자체 모듈 기능 이용



모듈기반의 개발 서버의 느린 시작



Native ESM 기반의 개발서버의 빠른 시작

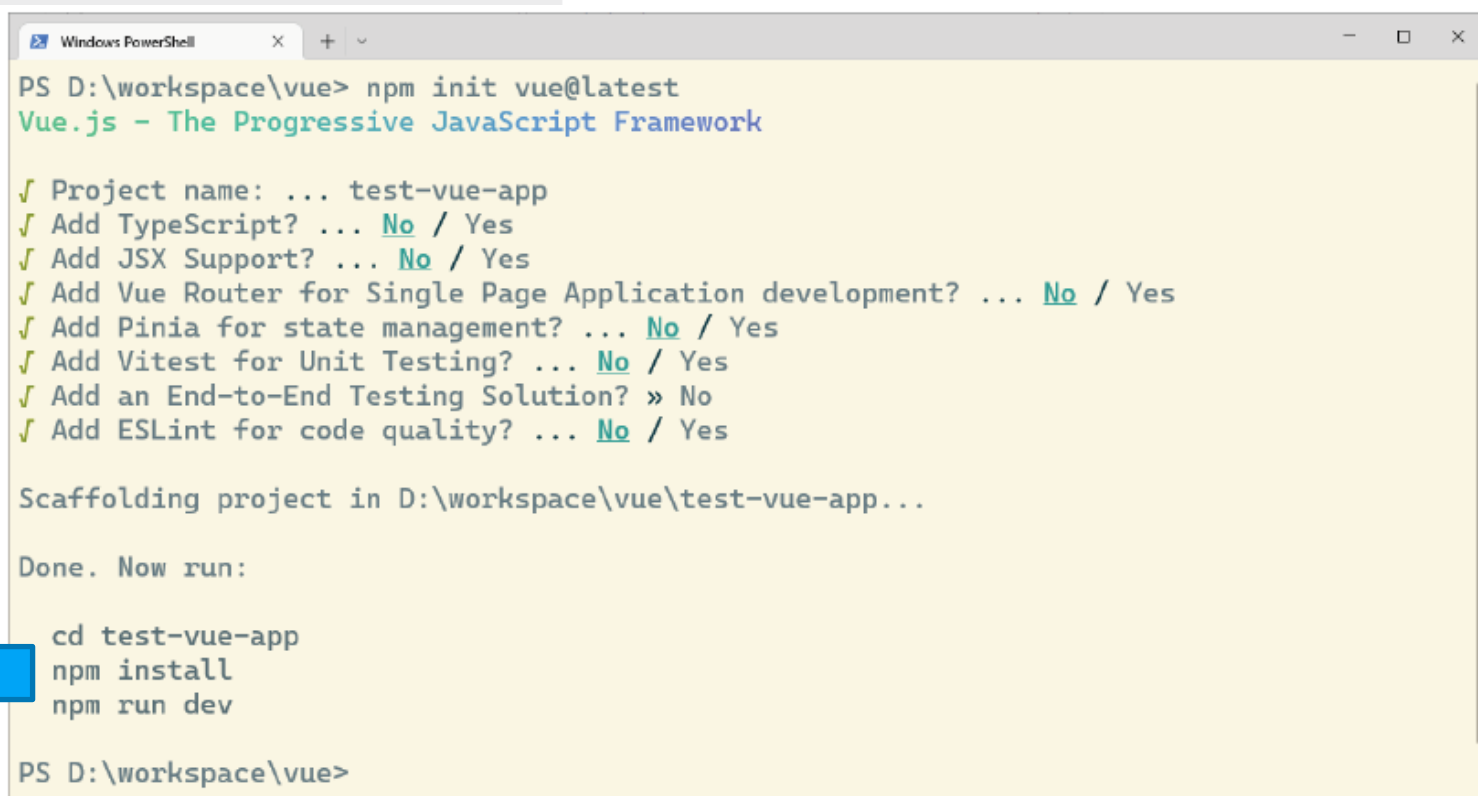
1. 프로젝트 설정 도구

❖ 프로젝트 생성 두가지 방법

```
//vite를 직접 이용하여 프로젝트 생성  
npm init vite [프로젝트명] -- --template vue
```

```
//create-vue 도구 사용  
//프로젝트명을 비롯한 여러 단계의 입력이 필요함  
//프로젝트명을 제외한 나머지는 일단 기본값으로 생성함  
npm init vue@latest
```

```
cd [프로젝트명]  
npm install  
npm run dev
```



```
Windows PowerShell
PS D:\workspace\vue> npm init vue@latest
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 D:\workspace\vue\test-vue-app...

Done. Now run:

  cd test-vue-app
  npm install
  npm run dev

PS D:\workspace\vue>
```

그림 07-05 create-vue 도구를 사용한 프로젝트 생성 예시

1. 프로젝트 설정 도구

❖이 과정에서는 create-vue 도구 사용

❖Vite 기반 프로젝트 폴더의 구조

- src: Javascript 코드, .vue 파일을 이곳에 작성합니다. 시작진입(Entry) 파일은 src/main.js 입니다.
- public: 이미지와 같은 정적 파일, 자원을 이곳에 작성합니다.
- dist: 빌드 후 생성된 산출물이 저장되는 디렉터리입니다.
- index.html: Vue 애플리케이션의 컴포넌트 트리는 index.html의 id가 app인 div 요소 내부에 렌더링됩니다.

❖빌드, 개발서버 구동 명령어

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



create-vue 기반 프로젝트에서 경로 지정 방법

기본적으로는 상대 경로 표기법을 사용하지만 vite.config.js의 기본 설정 때문에 절대 경로를 사용할 수 있습니다. 다음의 vite.config.js에서 볼드체로 표현된 부분에 의해 @는 src를 가리킵니다.

```
// vite.config.js의 vite 설정 정보
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))
    }
  }
})
```

이제 경로를 사용하실 때는 참조 파일의 경로로부터 상대 경로를 사용할 수도 있고, @를 중심으로 절대 경로를 사용할 수도 있습니다.

❖ Vite 기반 프로젝트의 상세한 설명

- vite.config.js 파일을 이용함
- 상세한 내용은 다음 문서
 - <https://vitejs.dev/config/#config-file>

```
// -- src/a/a1/A1.vue 에서 src/b/B.vue를 참조하는 경우
// 상대 경로로 참조하기
import B1 from '../../b/B.vue';
// 절대 경로로 참조하기
import B from '@b/B.vue';
```

만일 VSCode에서 절대 경로로 참조할 때 코드 자동완성(Code Intellisense) 기능을 지원받고 싶다면 다음과 같은 설정으로 jsconfig.json 파일을 추가해주세요.

```
// jsconfig.json 파일을 이용해 절대 경로에 대한 자동완성 기능 지원
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@/*": ["src/*"]
    }
  }
}
```


2. 생성된 프로젝트 구조 살펴보기

❖ 프로젝트 생성과 시작

```
// 프로젝트명을 test-vue-app으로 입력하고 나머지 값은 모두 기본값으로 설정합니다.  
npm init vue@latest  
cd test-vue-app  
npm install
```

```
PS D:\_Vue3QuickStart\Vue3예제\ch07> npm init vue@latest  
Need to install the following packages:  
  create-vue@3.6.1  
Ok to proceed? (y) y  
  
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 D:\_Vue3QuickStart\Vue3예제\ch07\test-vue-app...  
  
Done. Now run:  
  
  cd test-vue-app  
  npm install  
  npm run dev
```

- VSCode에서 '파일-폴더열기'

2. 생성된 프로젝트 구조 살펴보기

❖ src/main.js , index.html 검토

예제 07-01 : src/main.js 검토

```
01: import { createApp } from 'vue'
02: import App from './App.vue'
03:
04: import './assets/main.css'
05:
06: createApp(App).mount('#app')
```

예제 07-02 : index.html 검토

```
01: <!DOCTYPE html>
02: <html lang="en">
03:   <head>
04:     <meta charset="UTF-8">
05:     <link rel="icon" href="/favicon.ico">
06:     <meta name="viewport" content="width=device-width, initial-scale=1.0">
07:     <title>Vite App</title>
08:   </head>
09:   <body>
10:     <div id="app"></div>
11:     <script type="module" src="/src/main.js"></script>
12:   </body>
13: </html>
```

2. 생성된 프로젝트 구조 살펴보기

❖ 단일 파일 컴포넌트 살펴보기

▪ src/App.vue

예제 07-03 : src/App.vue 검토

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

<template>
  .....(생략)
</template>

<style scoped>
  .....(생략)
</style>
```

2. 생성된 프로젝트 구조 살펴보기

❖ 간단한 단일 파일 컴포넌트 작성과 사용

- src/components/CheckboxItem.vue

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

```
<script>
  export default (await import('vue')).defineComponent({
    name : "CheckboxItem",
    data() {
      return {
        checked: false
      };
    }
  })
</script>
```

```
<style scoped>
```

```
</style>
```

2. 생성된 프로젝트 구조 살펴보기

❖컴포넌트 등록 방법

■ 전역 컴포넌트

- Vue 애플리케이션 인스턴스의 component() 메서드를 이용해 등록함
- 첫번째 인자는 사용할 태그명, 두번째 인자는 컴포넌트 객체

예제 07-05 : src/main.js에서 전역 컴포넌트를 등록하는 방법

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

import './assets/main.css'

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

2. 생성된 프로젝트 구조 살펴보기

❖컴포넌트 등록 방법(이어서)

■ 지역 컴포넌트

- 특정 컴포넌트에서 컴포넌트를 직접 등록하여 사용하는 방법
- 컴포넌트 객체 내부의 components 옵션에 사용할 태그명과 컴포넌트 객체를 등록하여 사용함

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

- 태그명과 컴포넌트 이름이 동일하다면 ES6의 객체 리터럴 표기법을 적용하여 간결하게 적용

```
components: { CheckboxItem },
```

■ 지역 컴포넌트 > 전역 컴포넌트

2. 생성된 프로젝트 구조 살펴보기

❖ App.vue 다시 작성하기

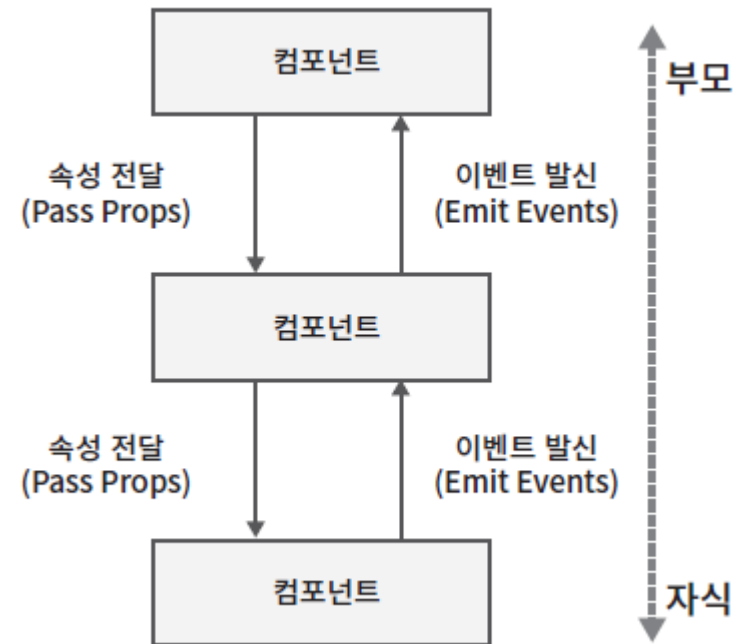
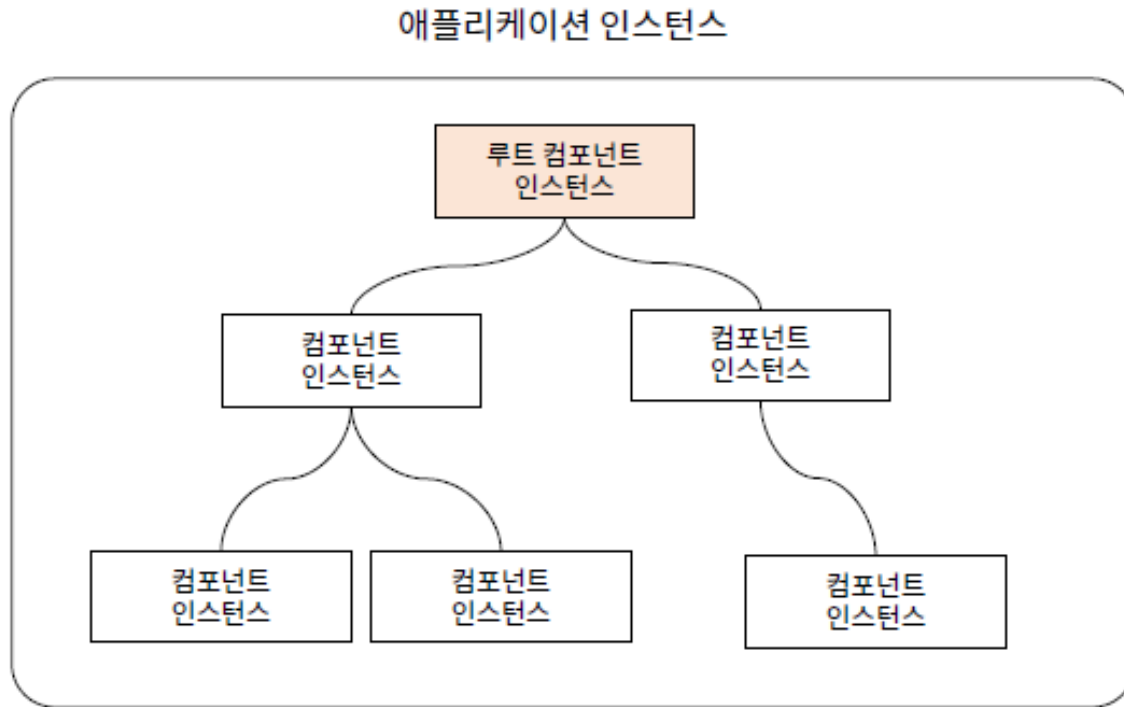
```
<template>
  <div>
    <h2> App 컴포넌트</h2>
    <hr />
    <ul>
      <CheckboxItem />
      <CheckboxItem />
      <CheckboxItem />
      <CheckboxItem />
    </ul>
  </div>
</template>

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


3. 컴포넌트의 조합

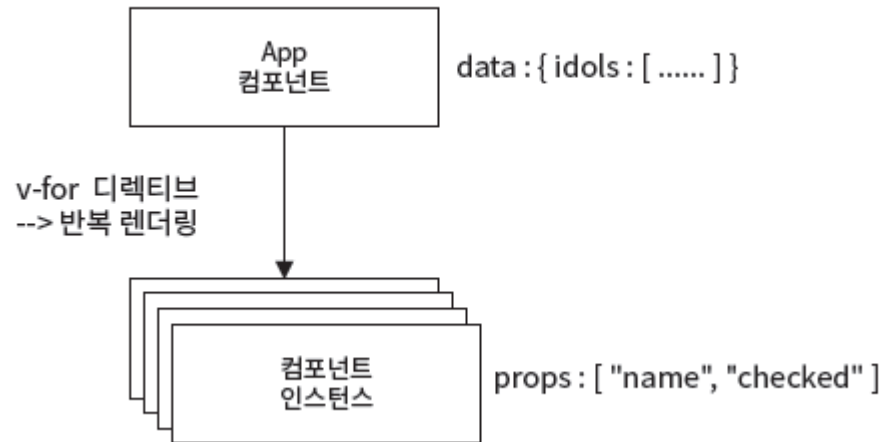
❖ 복잡한 화면은 여러 컴포넌트들을 조합하여 개발함

- 부모-자식 관계로 트리 구조 형성
- 부모-자식 간 정보 전달 방법



❖속성을 이용한 정보 전달

- 자식 컴포넌트 : props 옵션으로 속성을 정의
- 부모 컴포넌트 : v-bind 디렉티브를 이용해 자식 컴포넌트의 속성에 정보를 전달
- 속성으로 전달받은 데이터는 변경할 수 없음 -> 읽기 전용
 - 부모 컴포넌트에서 데이터를 변경하면 렌더링되면서 속성을 다시 전달
 - 자식 컴포넌트에서 변경된 속성을 확인할 수 있음



4.1 속성을 이용한 정보 전달

❖예제 07-07~8 : CheckboxItem.vue 변경, App.vue 변경

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

<script>
  export default {
    name : "CheckboxItem",
    props : ["name", "checked"],
  }
</script>
```

```
<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>

<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>
```

4.1 속성을 이용한 정보 전달

❖ 속성 전달 결과 확인

The screenshot shows a web browser at localhost:5173 displaying a form titled "관심있는 K-POP 가수?". The form contains a list of artists with checkboxes: ☒ BTS, ☐ Black Pink, ☐ EXO, and ☐ ITZY.

The Vue DevTools component inspector is open, showing the component tree with "App" selected. The component list shows "App" (3.2.45) and its children: `<CheckboxItem key=1>`, `<CheckboxItem key=2>`, `<CheckboxItem key=3>`, and `<CheckboxItem key=4>`.

The component details panel shows the `<App>` component's data property, which is an object with the following structure:

```
data
  idols: Array[4]
    0: Reactive
    1: Reactive
      id: 2
      name: "Black Pink"
      checked: false
    2: Reactive
    3: Reactive
```

4.2 속성을 이용해 객체 전달하기

❖ 예제 07-08의 리뷰

- 개별적인 속성 하나하나를 v-bind 로 전달 --> 불편함

```
<checkbox-item v-for="idol in idols" :key="idol.id"
            :name="idol.name" :checked="idol.checked" />
```

- 자식 컴포넌트의 속성명과 객체 내브의 속성명의 일치하다면?

```
<checkbox-item v-for="idol in idols" :key="idol.id" v-bind="idol" />
```

4.2 속성을 이용해 객체 전달하기

❖속성을 이용해 객체 전달하기

- 예제 07-09~10 : CheckboxItem2.vue 추가, App.vue 추가, src/main.js 변경

예제 07-09 : src/components/CheckboxItem2.vue 추가

```
01: <template>
02:   <li>
03:     <input type="checkbox" :checked="idol.checked" /> {{idol.name}}
04:   </li>
05: </template>
06:
07: <script>
08:   export default {
09:     name : "CheckboxItem2",
10:     props : ["idol"],
11:   }
12: </script>
```

예제 07-10 : src/App2.vue 추가

```
01: <template>
02:   <div>
03:     <h2>관심있는 K-POP 가수?</h2><hr />
04:     <ul>
05:       <CheckboxItem v-for="idol in idols" :key="idol.id" :idol="idol" />
06:     </ul>
07:   </div>
08: </template>
09:
10: <script>
11:   import CheckboxItem from './components/CheckboxItem2.vue'
12:   .....(생략 : App.vue의 export default { } 부분과 동일함)
13: </script>
```

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

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

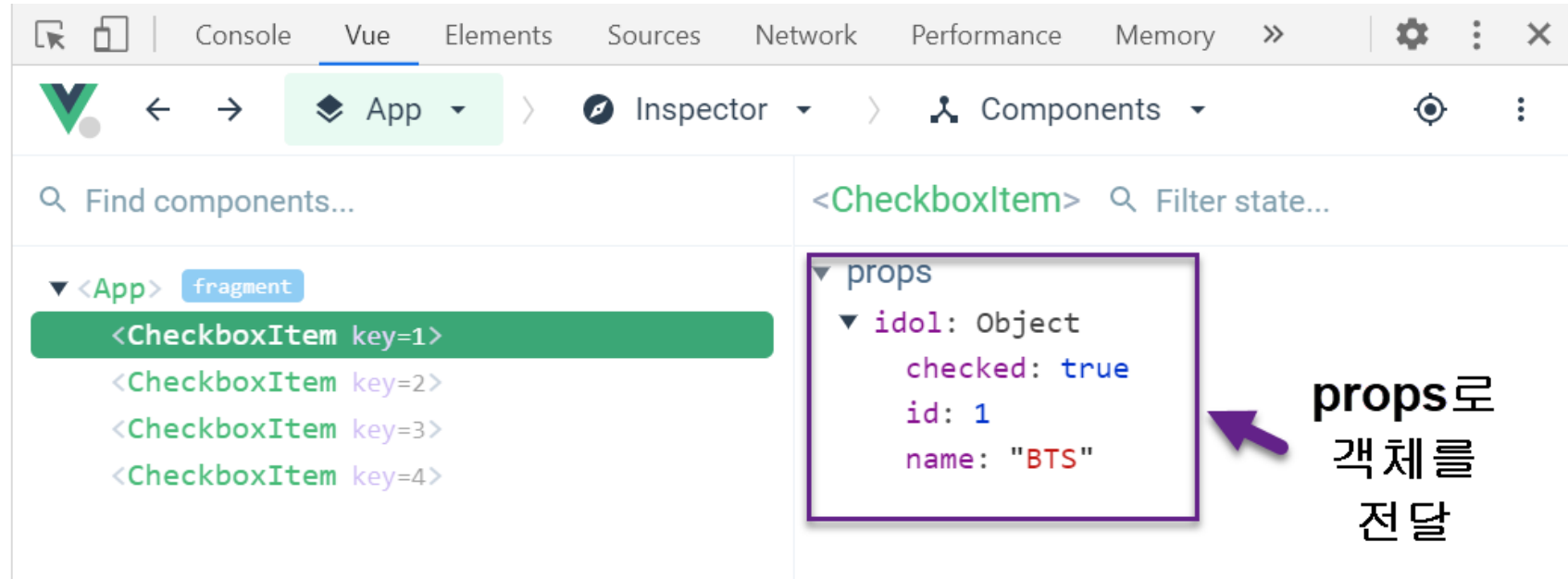
```
createApp(App).mount('#app')
```

4.2 속성을 이용해 객체 전달하기

❖ 예제 07-09~10 실행 결과

관심있는 K-POP 가수?

- ☒ BTS
- ☐ Black Pink
- ☐ EXO
- ☐ ITZY



props로
객체를
전달

❖ 속성으로 전달받은 값은 읽기 전용의 값

- 기본 타입 : 자식 컴포넌트에서 변경 불가능
- 참조 타입 : 변경이 가능하지만 권장하지 않음
 - 참조형이라 메모리 주소만 바뀌지 않는다면 가능함

4.2 속성을 이용해 객체 전달하기

❖자식 컴포넌트에서 기본 타입의 값을 변경하려 할 때의 오류 확인

[src/main.js에서 App컴포넌트를 사용하도록 변경]

```
import App from './App.vue'
//import App from './App2.vue'
```

[src/components/CheckboxItem.vue 컴포넌트에 created 생명주기 메서드 추가]

```
export default {
  name : "CheckboxItem",
  props : ["name", "checked"],
  created() {
    this.checked = true;
  }
}
```

```
⚠ ▶ [Vue warn]: Attempting to mutate prop "checked". Props are readonly.      runtime-core.esm-bundler.js:40
    at <CheckboxItem key=1 id=1 name="BTS" ... >
    at <App>

⚠ ▶ [Vue warn]: Unhandled error during execution of created hook              runtime-core.esm-bundler.js:40
    at <CheckboxItem key=1 id=1 name="BTS" ... >
    at <App>

✖ ▶ Uncaught TypeError: 'set' on proxy: trap returned falsish for property 'checked'  CheckboxItem.vue:12
    at Proxy.created (CheckboxItem.vue:12:26)
    at callWithErrorHandling (runtime-core.esm-bundler.js:157:36)
    at callWithAsyncErrorHandling (runtime-core.esm-bundler.js:166:21)
    at callHook (runtime-core.esm-bundler.js:3590:5)
    at applyOptions (runtime-core.esm-bundler.js:3492:9)
    at finishComponentSetup (runtime-core.esm-bundler.js:7368:9)
    at setupStatefulComponent (runtime-core.esm-bundler.js:7279:9)
    at setupComponent (runtime-core.esm-bundler.js:7201:11)
    at mountComponent (runtime-core.esm-bundler.js:5524:13)
    at processComponent (runtime-core.esm-bundler.js:5499:17)
```

4.2 속성을 이용해 객체 전달하기

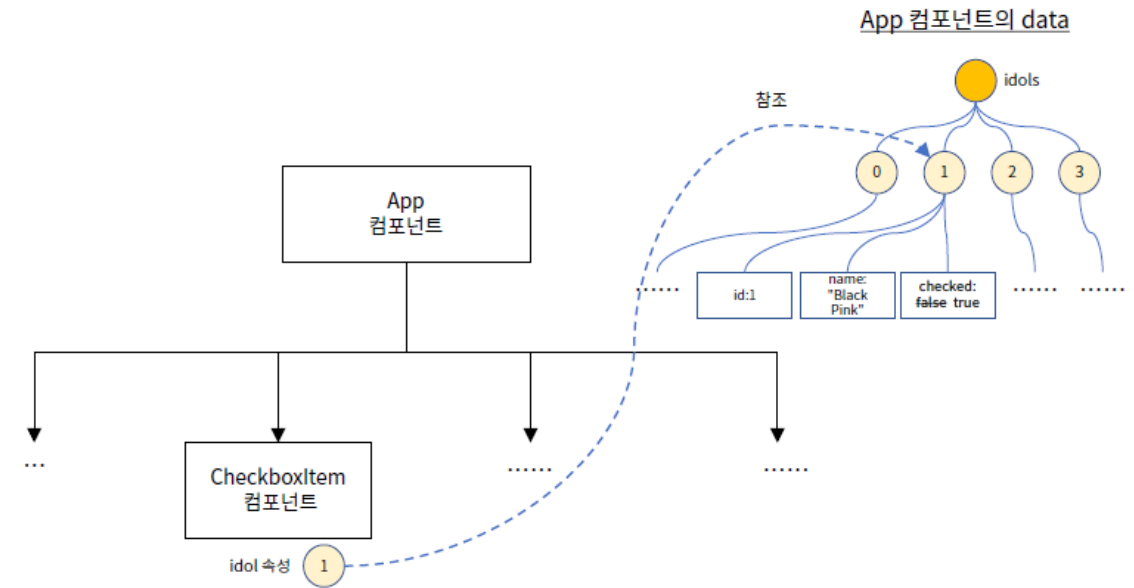
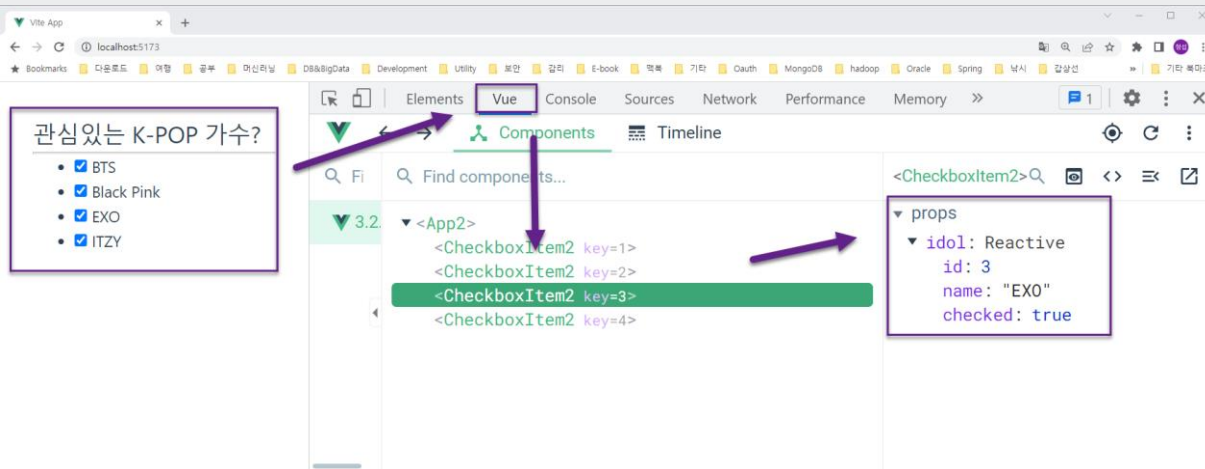
❖ 자식 컴포넌트에서 참조 타입의 값을 변경할 때 -> 변경이 가능하지만 권장(X)

[src/main.js에서 App2 컴포넌트를 사용하도록 변경]

```
//import App from './App.vue'
import App from './App2.vue'
```

[src/components/CheckboxItem2.vue 컴포넌트에 created 생명주기 메서드 추가]

```
export default {
  name: "CheckboxItem2",
  props: ["idol"],
  created() {
    this.idol.checked = true;
  }
}
```



**** this.idol.checked = true로 변경하면?**

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

그림 07-15 객체가 전달된 속성을 변경했을 때의 실행 구조

4.3 속성의 유효성 검증

❖ 속성에 대한 엄격한 유효성 검증이 필요하다면?

- 배열이 아닌 객체 형태로 속성 지정

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

- default 값이 참조형이라면 함수형태로 지정

```
default() {  
  return 객체, 또는 배열  
}
```

- 타입으로 사용할 수 있는 생성자

String	Number	Boolean	Array
Object	Date	Function	Symbol

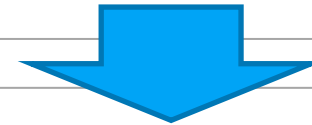
❖예제 07-11~12 : CheckboxItem.vue, App.vue 변경

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

<script>
  export default {
    name: "CheckboxItem",
    props: {
      id: [Number, String],
      name: String,
      checked: {
        type: Boolean,
        required: false,
        default: false
      }
    }
  }
</script>
```

의도적으로 잘못된
값을 전달해봄

```
data() {
  return {
    idols : [
      { id:1, name:"BTS", checked:true},
      { id:2, name:"Black Pink" },
      { id:3, name:"EXO" },
      { id:4, name:"ITZY" },
    ]
  }
}
```



```
data() {
  return {
    idols : [
      { id:1, name:"BTS", checked:true},
      { id:2, name:"Black Pink", checked:1 },
      { id:3, name:"EXO" },
      { id:4, name: { special:"ITZY" } },
    ]
  }
}
```

4.3 속성의 유효성 검증

❖의도적으로 잘못된 값을 전달했을 때 오류

The screenshot shows a web browser window with a Vite App running on localhost:5173. The page displays a form titled "관심있는 K-POP 가수?" (Interested K-POP Artists?). The form has four items, each with a checkbox and a name:

- ☒ 1 - BTS
- ☒ 2 - Black Pink
- ☐ 3 - EXO
- ☐ 4 - { "special": "ITZY" }

The browser's developer console is open, showing two Vue warnings:

```
[Vue warn]: Invalid prop: type check failed for prop "checked". Expected Boolean, got Number with value 1.  
    at <CheckboxItem key=2 id=2 name="Black Pink" ... >  
    at <App>
```

```
[Vue warn]: Invalid prop: type check failed for prop "name". Expected String with value "[object Object]", got Object  
    at <CheckboxItem key=4 id=4 name= {special: 'ITZY'} ... >  
    at <App>
```

4.3 속성의 유효성 검증

❖ 유효성 검증으로 지정할 수 있는 것

- 생성자 함수 또는 클래스
- 예제 07-14~16

예제 07-14 :src/Idol.js 추가

```
export default class Idol {  
  constructor(id, name, checked) {  
    this.id = id;  
    this.name = name;  
    this.checked = checked;  
  }  
}
```

예제 07-15 : src/components/CheckboxItem3.vue 추가

```
01: <template>  
02:   <li>  
03:     <input type="checkbox" :checked="idol.checked" /> {{idol.name}}  
04:   </li>  
05: </template>  
06:  
07: <script>  
08:   import Idol from '../Idol';  
09:   export default {  
10:     name: "CheckboxItem3",  
11:     props: {  
12:       idol: Idol,  
13:     },  
14:   }  
15: </script>
```

```
data() {  
  return {  
    idols: [  
      new Idol(1, "BTS", true),  
      new Idol(2, "Black Pink", false),  
      new Idol(3, "EXO", false),  
      new Idol(4, "ITZY", false),  
    ]  
  }  
}
```

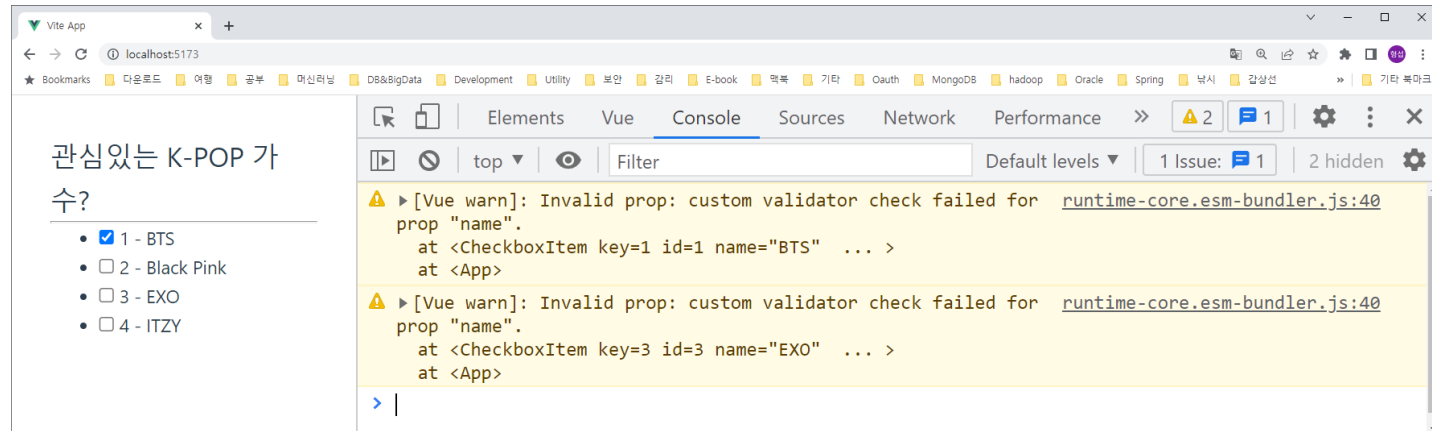

4.3 속성의 유효성 검증

❖ 사용자 정의 유효성 검증

- 유효성 검사 함수 사용 (validator)
- 예제 07-17~18 : CheckboxItem.vue 변경
 - name 속성이 4글자 이상인지를 확인하도록...

```
export default {
  name: "CheckboxItem",
  props: {
    id: [Number, String],
    //name: String,
    name: {
      validator(v) {
        return typeof(v) !== "string" ?
          false :
          v.trim().length >= 4 ? true : false
      }
    },
    checked: {
      type: Boolean,
      required: false,
      default: false
    }
  },
}
```

```
10: data() {
11:   return {
12:     idols : [
13:       { id:1, name:"BTS", checked:true},
14:       { id:2, name:"Black Pink" },
15:       { id:3, name:"EXO" },
16:       { id:4, name:"ITZY" },
17:     ]
18:   }
19: }
```



❖ 자식 -> 부모로 정보를 전달하려면?

- 사용자 정의 이벤트 사용
 - 자식 컴포넌트는 이벤트를 발신
 - 부모 컴포넌트에서는 v-on 디렉티브를 이용해 수신

[자식 컴포넌트에서]

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

[부모 컴포넌트에서]

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

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

❖ 예제 07-19 : InputName.vue

■ 이벤트 발신

```
<template>
  <div style="border:solid 1px 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>
```

❖ 예제 07-20 : App4.vue

■ 이벤트 수신

```
<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>
```

5.1 사용자 정의 이벤트를 이용한 정보 전달

❖ 예제 07-19~21 구조

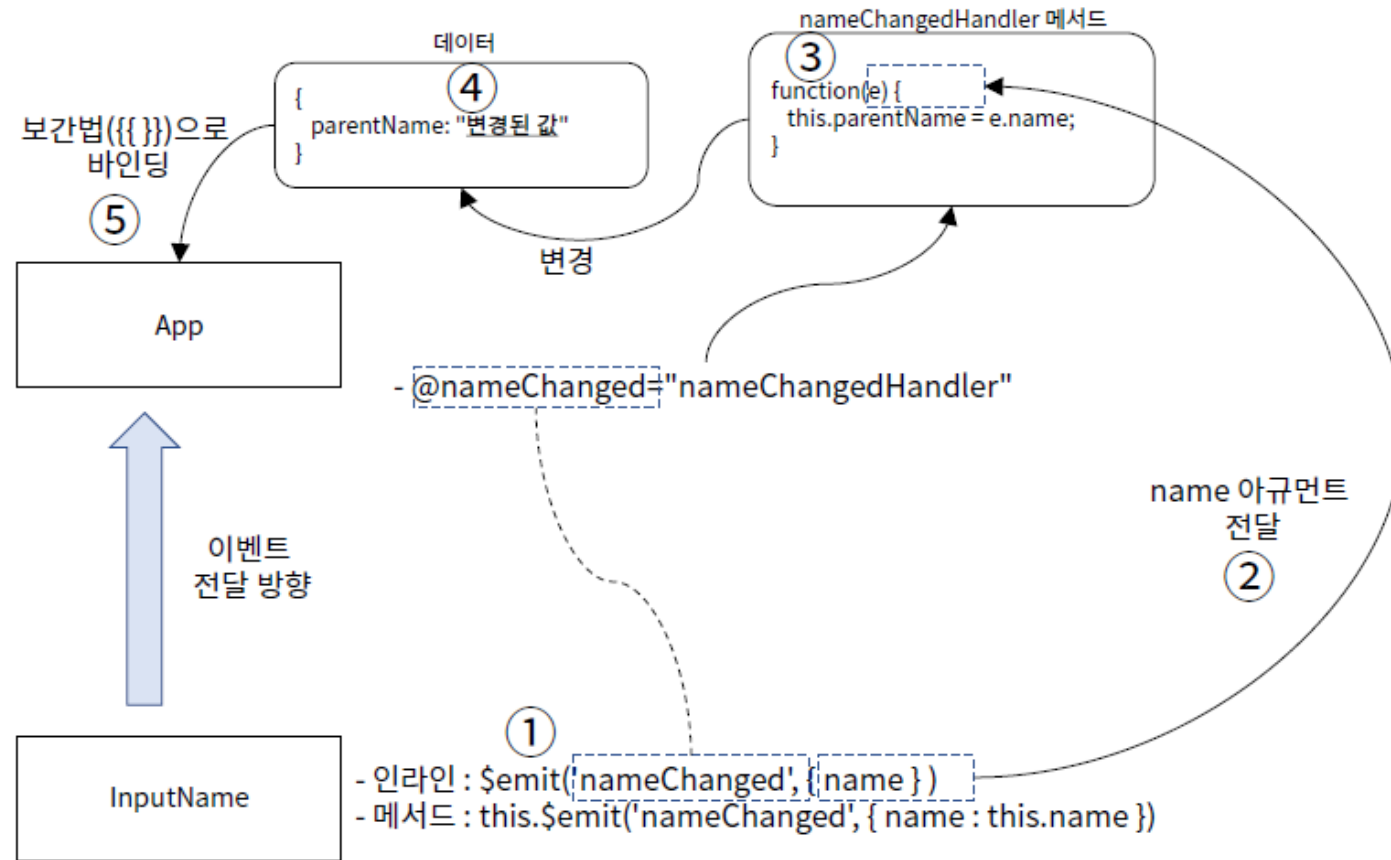
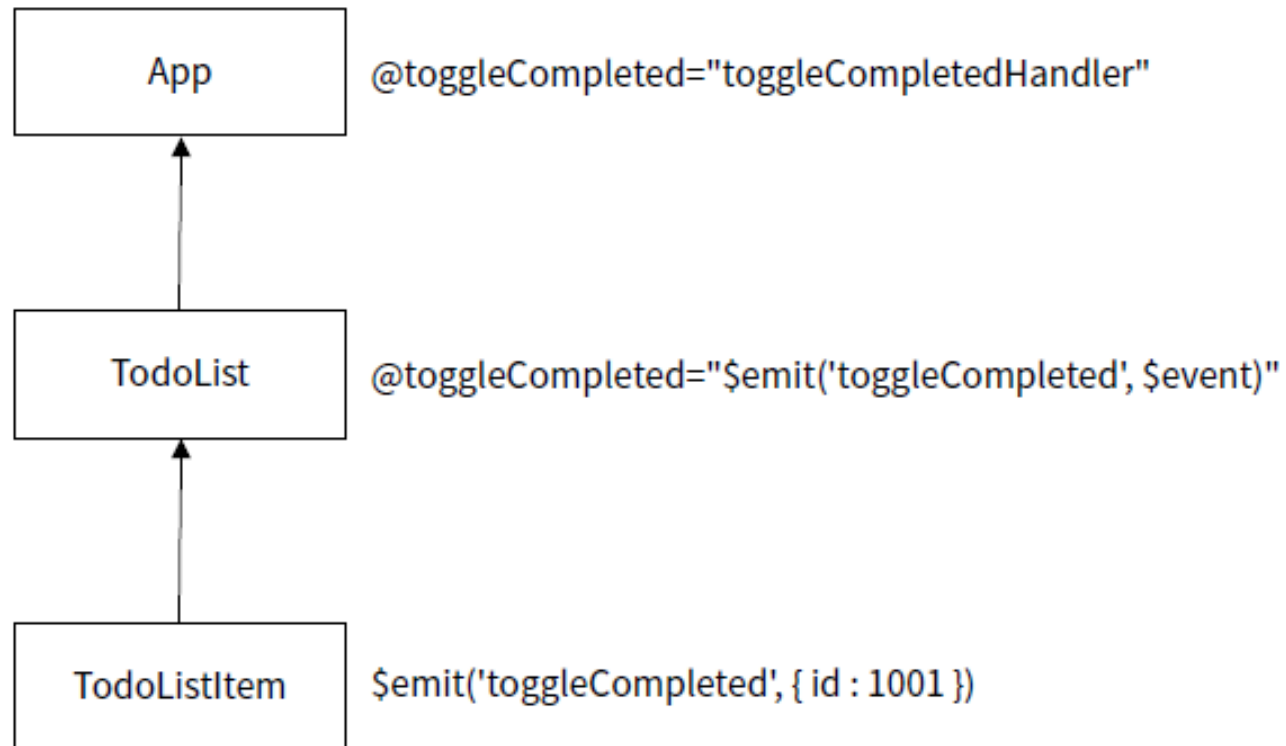


그림 07-19 : 예제 07-19~21의 구조

❖ 컴포넌트 계층 구조가 복잡하다면?

- 중간 계층에서 받아서 부모로 전달해야 함



❖이벤트 유효성 검증 적용 방법

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

또는

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

❖사용 권장 이유

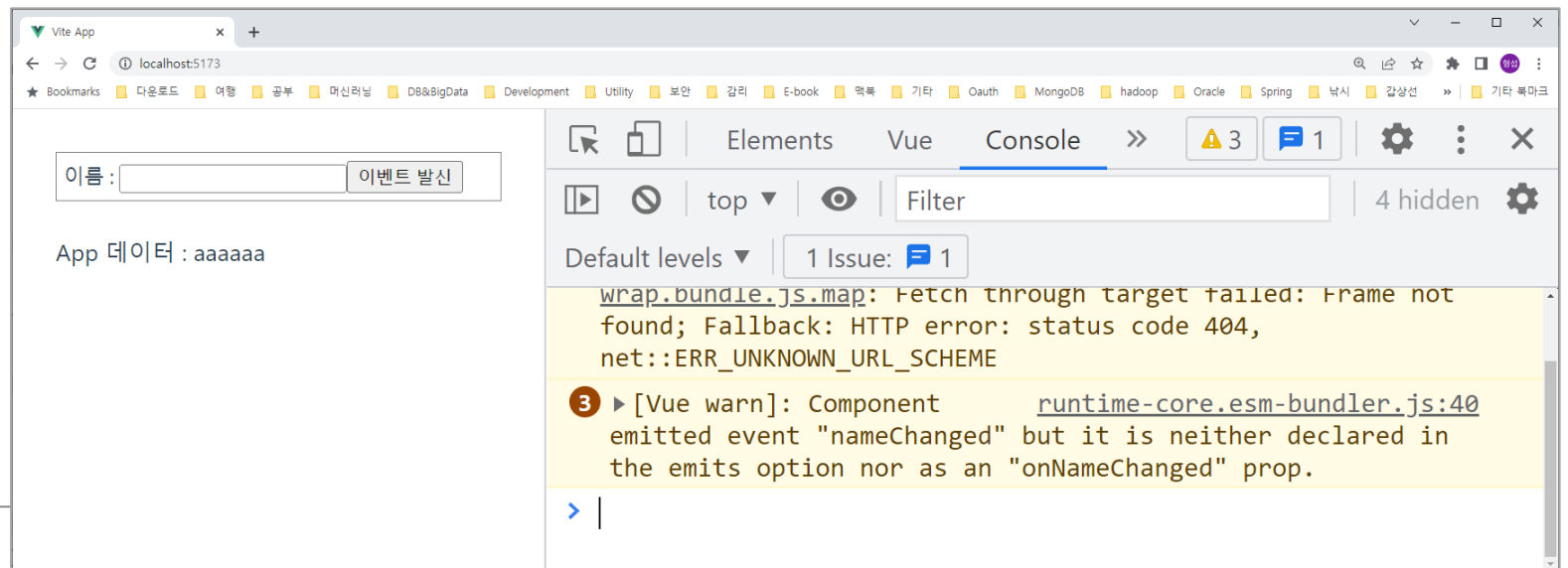
- emits 옵션만 보면 어떤 이벤트를 발신하는지 알 수 있음
- 컴포넌트 작동 방식에 대한 명확한 문서화를 위해서도 권장

5.2 이벤트 유효성 검증

❖의도적으로 발신하는 이벤트명 변경 --> 오류 발생 확인

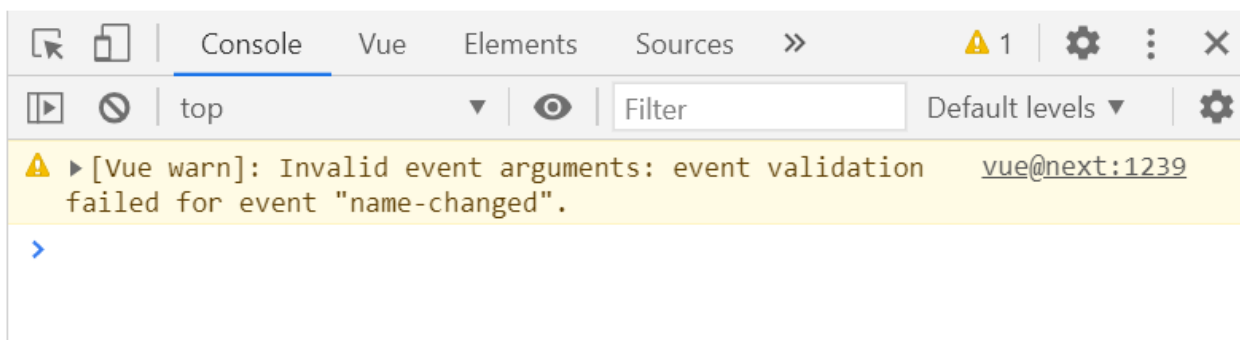
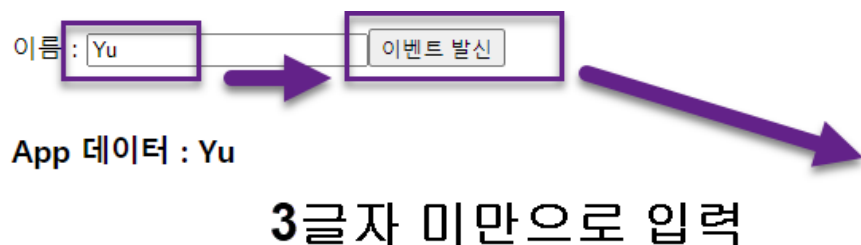
예제 07-22 : src/components/InputName.vue 변경

```
01: <template>
02:   .....
03: </template>
04:
05: <script>
06:   export default {
07:     name: "InputName",
08:     emits : [ "nameChanged1"],
09:     data() {
10:       return {
11:         name: ""
12:       };
13:     },
14:   }
15: </script>
```



❖ 전달하는 이벤트 아규먼트 검증

```
01: export default {  
02:   name: "InputName",  
03:   //emits : [ "nameChanged1"],  
04:   emits: {  
05:     nameChanged: (e) => {  
06:       return e.name && typeof(e.name) === "string" && e.name.trim().length >= 3  
07:         ? true : false  
08:     },  
09:     .....  
10: }
```



6. 이벤트 에미터 사용

❖ 복잡한 컴포넌트 구조

- 부모 - 자식 - 손자 경로를 따라 이벤트를 일일이 전달해야 함
- 이런 경우 이벤트 에미터를 사용할 수 있음
 - 형제1- 형제2 관계
 - 부모- 자식- 손자-증손자 관계

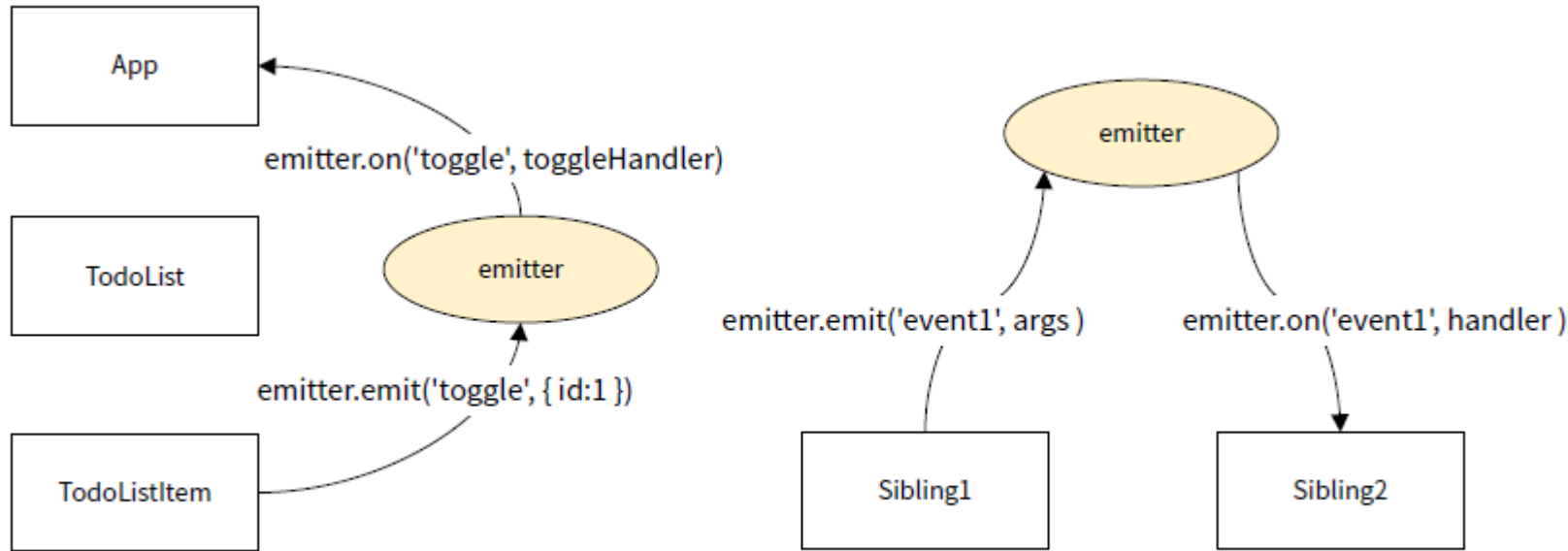


그림 07-23 : 이벤트 에미터 적용 방법

6. 이벤트 에미터 사용

❖ 이벤트 에미터 라이브러리

- mitt : <https://github.com/developit/mitt>

```
// CTRL+C를 눌러서 실행 중이던 개발 웹서버를 중단하고 다음 명령어를 실행하여 mitt를 설치합니다.  
npm install --save mitt
```

❖ src/main.js 변경

```
1 import { createApp } from 'vue'  
2 import App from './App5.vue'  
3 import mitt from 'mitt';  
4  
5 import './assets/main.css'  
6  
7 const emitter = mitt();  
8 emitter.on('*', (type, e) => console.log(` ## 이벤트 로그 : ${type} -> `, e) )  
9 const app = createApp(App)  
10 app.config.globalProperties.emitter = emitter;  
11 app.mount("#app");
```

- 이벤트 에미터를 이용해서 이벤트 발신, 수신

❖ 예제 07-24

예제 07-24 : src/components/Sender.vue

```
01: <template>
02:   <div style="border:solid 1px gray; margin:5px; padding:5px;">
03:     <h2>Sender</h2><hr/>
04:     <button @click="sendMessage">이벤트 발신</button>|
05:   </div>
06: </template>
07:
08: <script>
09:   export default {
10:     name: "Sender",
11:     methods: {
12:       sendMessage() {
13:         this.emitter.emit("message", Date.now() + "에 발신된 메시지");
14:       }
15:     }
16:   }
17: </script>
```

❖예제 07-25

예제 07-25 : src/components/Receiver.vue

```
01: <template>
02:   <div style="border:solid 1px gray; margin:5px; padding:5px;">
03:     <h2>Receiver</h2>
04:     <hr />
05:     <h2>전송된 텍스트 : {{textMessage}}</h2>
06:   </div>
07: </template>
08:
09: <script>
10:   export default {
11:     name: "Receiver",
12:     created() {
13:       this.emitter.on("message", this.receiveHandler);
14:     },
```

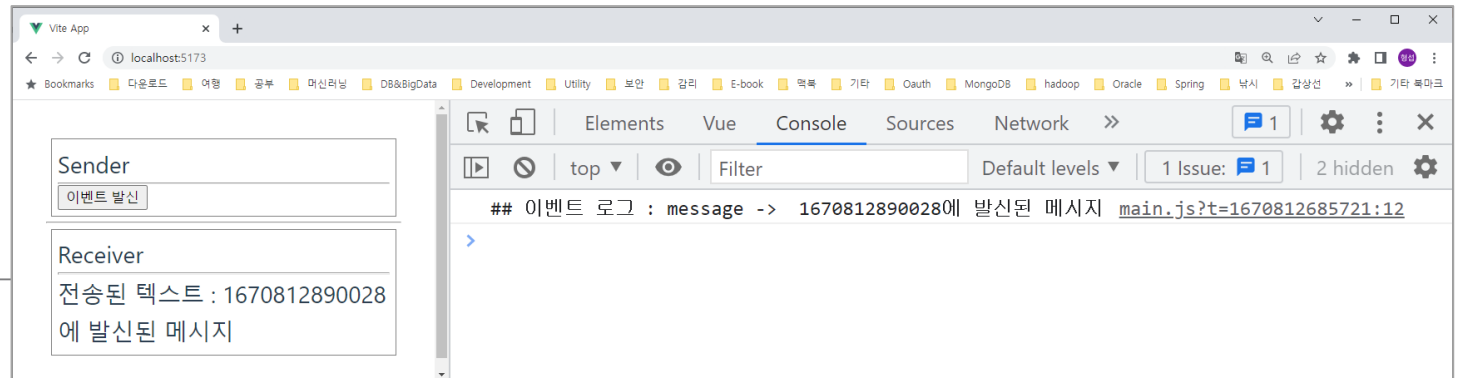
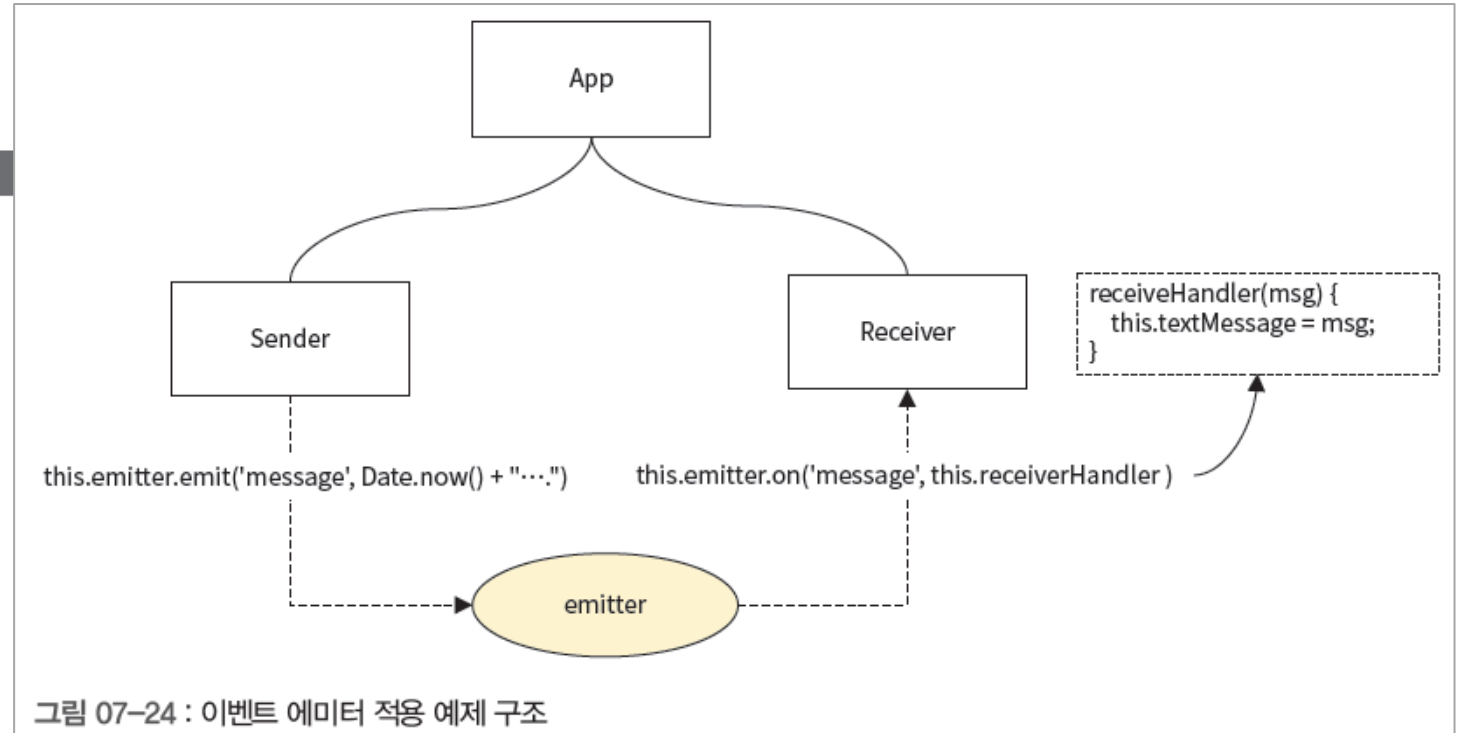
```
15:     data() {
16:       return {
17:         textMessage: ""
18:       }
19:     },
20:     methods: {
21:       receiveHandler(msg) {
22:         this.textMessage = msg;
23:       }
24:     },
25:   }
26: </script>
```

6. 이벤트 에미터 사용

❖예제 07-26

예제 07-26 : src/App5.vue

```
01: <template>
02:   <div>
03:     <Sender />
04:     <hr />
05:     <Receiver />
06:   </div>
07: </template>
08:
09: <script>
10:   import Receiver from './components/Receiver.vue'
11:   import Sender from './components/Sender.vue'
12:   export default {
13:     name : "App5",
14:     components : { Sender, Receiver }
15:   }
16: </script>
```



6. 이벤트 에미터 사용

❖ 이벤트 에미터 사용은 공식적으로 권장하는 방법은 아님



이벤트 에미터를 사용하는 방법은 공식적으로 권장하는 방법은 아닙니다. Vue 공식 문서에서는 전역 상태 관리 기능을 제공하는 라이브러리로 Vuex 또는 Pinia를 이용할 것을 권장합니다. 하지만 필자는 간단한 구조의 애플리케이션이라면 이벤트 에미터를 사용하는 것이 나쁘지 않다고 생각합니다.

7. TodoList 예제 리팩토링

❖ 6장에서 작성했던 TodoList 앱 예제를 단일 파일 컴포넌트 기반으로 변경

:: Todolist App

할일을 여기에 입력!

추가

자전거 타기

삭제

딸과 공원 산책 (완료)

삭제

일요일 애견 카페

삭제

Vue 원고 집필 (완료)

삭제

마트에서 장보기

삭제

❖컴포넌트 분할 기준

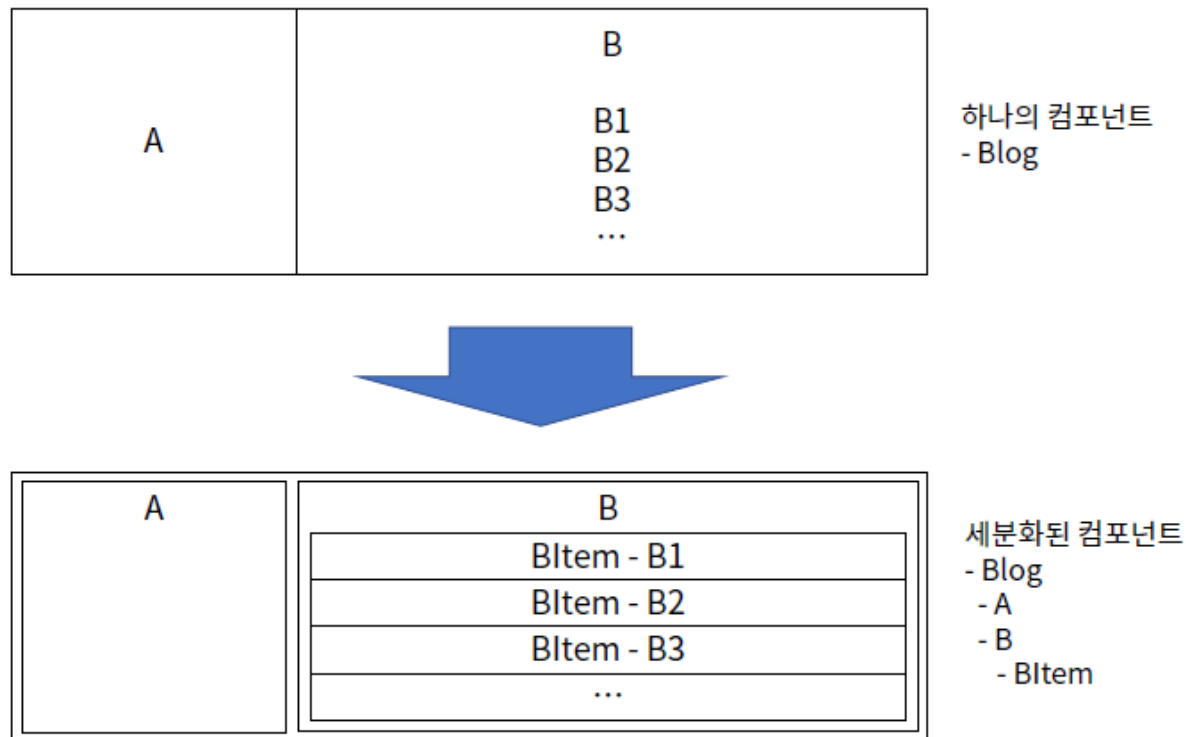
- 재사용성
- 테스트 용이성
- 디버깅 편의성

+

- 렌더링 최적화

❖Vue에서의 렌더링할지 말지 결정 단위

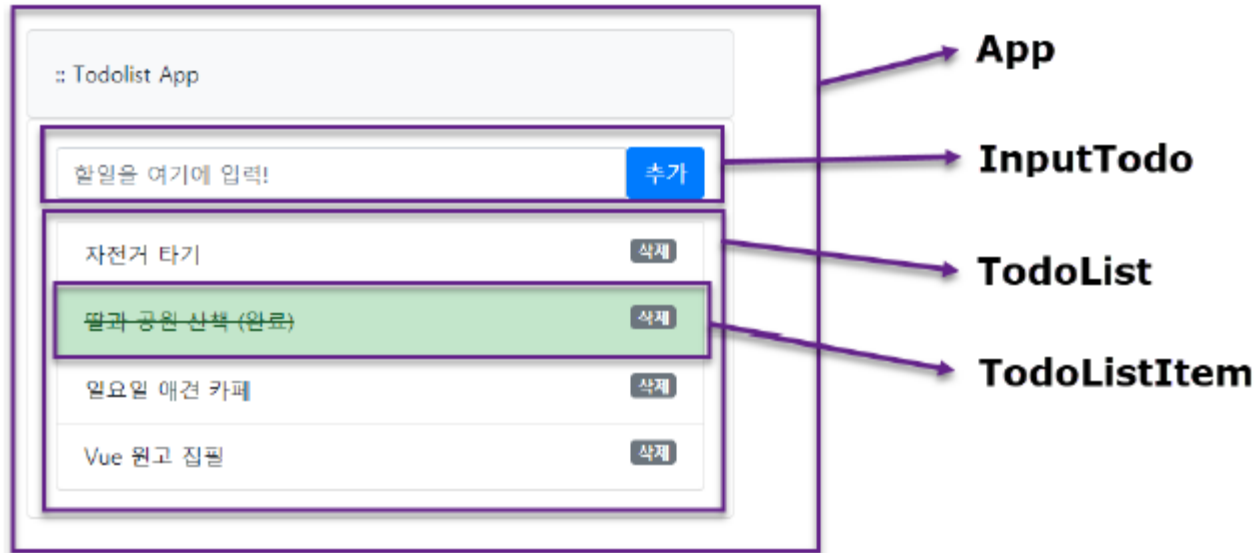
- 컴포넌트!!
- 컴포넌트의 입자가 너무 크면?
 - 작은 데이터 하나만 변경되어도 컴포넌트 전체가 다시 렌더링됨
- 한번에 변경되는 데이터를 렌더링하는 UI 단위로 컴포넌트를 분할하는 편이 좋음



데이터가 변경되는 단위로 컴포넌트를 세분화하면 해당 컴포넌트만 다시 렌더링하고 나머지 컴포넌트는 다시 렌더링하지 않게 되어 보다 좋은 렌더링 성능을 제공할 수 있게 됩니다. 특히 반복 렌더링하는 부분은 반드시 별도의 컴포넌트를 작성하세요.

7.1 컴포넌트 분할과 정의

❖ TodoList 앱 화면의 컴포넌트 분할



❖ 화면 단위의 데이터와 메서드 도출

- 최상위 App 컴포넌트에 정의될 것임
 - 데이터는 속성을 이용해 자식 컴포넌트로 전달
 - 반드시 이렇게 해야하는 것은 아니지만 데이터에 대한 변경 추적을 용이하게 함
- 데이터
 - 관리가 필요한 중요한 데이터
 - 데이터의 생명주기 관리가 필요한 경우

```
{
  todoList : [
    { id: 1, todo:"자전거 타기", completed: false },
    { id: 2, todo:"딸과 공원 산책", completed: true },
    { id: 3, todo:"일요일 애견 카페", completed: false },
    { id: 4, todo:"Vue 원고 집필", completed: false },
  ]
}
```

- InputTodo 컴포넌트에서의 todo는 중앙집중화된 상태관리 (X)
 - 사용자의 입력을 임시로 받아내기 위한 용도이므로

❖ 메서드와 수신 이벤트 목록 정의

[메서드 목록]

- addTodo(todo)
- deleteTodo(id)
- toggleCompleted(id)

[수신 이벤트 목록]

- add-todo : 전달 인자-todo
- delete-todo : 전달 인자-id
- toggle-completed : 전달 인자-id

❖ 컴포넌트 목록

* App 컴포넌트

- data : todoList 배열
- methods : addTodo, deleteTodo, toggleCompleted 메서드
- 수신 이벤트 : add-todo, delete-todo, toggle-completed

* InputTodo 컴포넌트

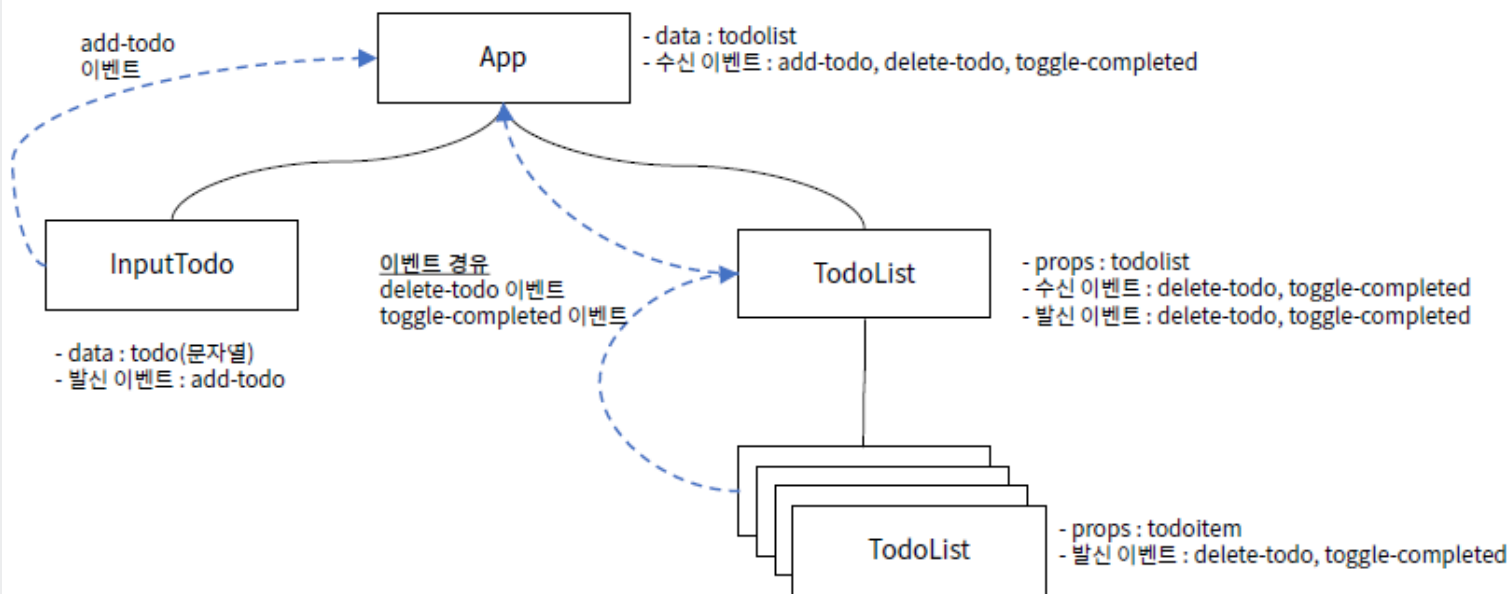
- data : todo 문자열 값
- 발신 이벤트 : add-todo

* TodoList 컴포넌트

- props : todoList 배열(todoList)
- 발신이벤트 : delete-todo(경유), toggle-completed(경유)
- 수신이벤트 : delete-todo(경유), toggle-completed(경유)

* TodoListItem 컴포넌트

- props : todoItem(todoList 배열의 값 하나)
- 발신 이벤트 : delete-todo, toggle-completed



❖프로젝트 생성

- npm init vue todolist-app
- cd todolist-app
- npm install
- npm install bootstrap@5
- src/components, src/assets 디렉터리 내의 모든 파일 삭제

❖파일 생성

```
src/assets/main.css  
src/components/ToDoList.vue  
src/components/InputTodo.vue  
src/components/ToDoListItem.vue
```

❖src/main.js 변경, src/assets/main.css 추가

예제 07-27 : src/main.js 변경

```
import { createApp } from 'vue'
import App from './App.vue'
import 'bootstrap/dist/css/bootstrap.css'
import './assets/main.css'

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

예제 07-28 : src/assets/main.css 추가

```
body { margin: 0; padding: 0; font-family: sans-serif; }
.title { text-align: center; font-weight:bold; font-size:20pt; }
.todo-done { text-decoration: line-through; }
.container { padding:10px 10px 10px 10px; }
.panel-borderless { border: 0; box-shadow: none; }
.pointer { cursor:pointer; }
```


❖예제 07-29 : src/App.vue

```
1 <template>
2   <div id="app" class="container">
3     <div class="card card-body bg-light">
4       <div classe="title">:: Todolist App</div>
5     </div>
6     <div class="card card-default card-borderless">
7       <div class="card-body">
8         <InputTodo @add-todo="addTodo" />
9         <TodoList :todoList="todoList" @delete-todo="deleteTodo"
10          @toggle-completed="toggleCompleted" />
11       </div>
12     </div>
13   </div>
14 </template>
15
16 <script>
17   import TodoList from './components/TodoList.vue'
18   import InputTodo from './components/InputTodo.vue'
19
20   let ts = new Date().getTime()
21
22   export default {
23     name: "App",
24     components : { InputTodo, TodoList },
```

```
25   data() {
26     return {
27       todoList : [
28         { id: ts, todo:"자전거 타기", completed: false },
29         { id: ts+1, todo:"딸과 공원 산책", completed: true },
30         { id: ts+2, todo:"일요일 애견 카페", completed: false },
31         { id: ts+3, todo:"Vue 원고 집필", completed: false },
32       ]
33     },
34   },
35   methods: {
36     addTodo(todo) {
37       if (todo.length >= 2) {
38         this.todoList.push({ id: new Date().getTime(),
39           todo: todo, completed: false });
40       }
41     },
42     deleteTodo(id) {
43       let index = this.todoList.findIndex((item) => id === item.id);
44       this.todoList.splice(index, 1);
45     },
46     toggleCompleted(id) {
47       let index = this.todoList.findIndex((item) => id === item.id);
48       this.todoList[index].completed = !this.todoList[index].completed;
49     }
50   }
51 }
52 </script>
```

7.2 속성과 이벤트를 조합한 리팩토링

❖예제 07-30 : src/components/InputTodo.vue

```
1 <template>
2   <div class="row mb-3">
3     <div class="col">
4       <div class="input-group">
5         <input id="msg" type="text" class="form-control" name="msg"
6           placeholder="할일을 여기에 입력!" v-model.trim="todo"
7           @keyup.enter="addTodoHandler" />
8         <span class="btn btn-primary input-group-addon"
9           @click="addTodoHandler">추가</span>
10      </div>
11    </div>
12  </div>
13 </template>
14
```

```
15 <script>
16   export default {
17     name : "InputTodo",
18     data() {
19       return { todo : "" }
20     },
21     emits : ["add-todo"],
22     methods : {
23       addTodoHandler() {
24         if (this.todo.length >= 3) {
25           this.$emit('add-todo', this.todo);
26           this.todo = "";
27         }
28       }
29     },
30   },
31 </script>
```

7.2 속성과 이벤트를 조합한 리팩토링

❖예제 07-31 : src/components/ToDoList.vue

```
1 <template>
2   <div class="row">
3     <div class="col">
4       <ul class="list-group">
5         <ToDoListItem v-for="todoItem in todoList" :key="todoItem.id"
6           :todoItem="todoItem" @delete-todo="$emit('delete-todo', $event)"
7           @toggle-completed="$emit('toggle-completed', $event)" />
8       </ul>
9     </div>
10  </div>
11 </template>
12
13 <script>
14   import ToDoListItem from './ToDoListItem.vue'
15
16   export default {
17     name : "ToDoList",
18     components : { ToDoListItem },
19     props : {
20       todoList : { type : Array, required:true }
21     },
22     emits : ["delete-todo", "toggle-completed"],
23   }
24 </script>
```

7.2 속성과 이벤트를 조합한 리팩토링

❖예제 07-32 : src/components/ToDoListItem.vue

```
1 <template>
2   <li class="list-group-item"
3     :class="{ 'list-group-item-success': todoItem.completed } "
4     @click="$emit('toggle-completed', todoItem.id)" >
5     <span class="pointer" :class="{ 'todo-done': todoItem.completed }">
6       {{ todoItem.todo }} {{ todoItem.completed ? "(완료)" : "" }}
7     </span>
8     <span class="float-end badge bg-secondary pointer"
9       @click.stop="$emit('delete-todo', todoItem.id)">삭제</span>
10  </li>
11 </template>
12
13 <script>
14   export default {
15     name : "ToDoListItem",
16     props : {
17       todoItem : { type : Object, required: true }
18     },
19     emits : ["delete-todo", "toggle-completed"],
20   }
21 </script>
```

7.2 속성과 이벤트를 조합한 리팩토링

❖ 실행 결과

The screenshot displays a web browser window with the address bar showing 'localhost:5173'. The browser's bookmark bar includes various categories like 'Downloads', 'Travel', 'Public', etc. The main content area shows a 'Vite App' running. The app's interface includes a text input field with the placeholder '할일을 여기에 입력!' and a blue '추가' button. Below this is a list of tasks, each with a '삭제' button. The tasks are: '자전거 타기', '딸과 공원 산책 (완료)', '일요일 애견 카페', 'Vue 원고 집필 (완료)', and 'Vue 인강 녹화'. The 'Vue 원고 집필 (완료)' task is highlighted in green.

The Vue DevTools component inspector is open, showing the component tree. The selected component is '<App>' with a duration of 10.1 ms. The component tree shows the following structure:

```
<App>
  <InputTodo>
    <TodoList>
      <TodoListItem key=16>
      <TodoListItem key=16>
      <TodoListItem key=16>
      <TodoListItem key=16>
      <TodoListItem key=16>
```

The right panel shows the state of the selected component, specifically the 'data' property. The state is:

```
data
  todoList: Array[4]
    0: Reactive
    1: Reactive
    2: Reactive
    3: Reactive
```

❖ 직전까지의 예제에 이벤트 에미터 적용

- 템플릿에 이벤트를 수신하는 코드(v-on, @)가 모두 사라짐
- App 컴포넌트의 created 생명주기 메서드에서 이벤트 수신 코드 작성
- 각 컴포넌트에서 emits 옵션 모두 삭제

❖ mitt 라이브러리 추가

- npm install --save mitt
- src/main.js 변경

```
1  import { createApp } from 'vue'
2  import App from './App.vue'
3  import 'bootstrap/dist/css/bootstrap.css'
4  import './assets/main.css'
5  import mitt from 'mitt'
6
7  const emitter = mitt();
8
9  const app = createApp(App)
10 app.config.globalProperties.emitter = emitter
11 app.mount('#app')
```


❖ 예제 07-34 : src/App.vue 변경

```
<div class="card-body">  
  <InputTodo />  
  <TodoList :todoList="todoList" />  
</div>
```

```
export default {  
  name: "App",  
  components : { InputTodo, TodoList },  
  created() {  
    this.emitter.on("add-todo", this.addTo);  
    this.emitter.on("delete-todo", this.deleteTodo);  
    this.emitter.on("toggle-completed", this.toggleCompleted);  
  },  
}
```


7.3 이벤트 에미터 적용하기

❖예제 07-35 : src/components/InputTodo.vue 변경

```
15 <script>
16   export default {
17     name : "InputTodo",
18     data() {
19       return { todo : "" }
20     },
21     methods : {
22       addTodoHandler() {
23         if (this.todo.length >= 3) {
24           this.emitter.emit("add-todo", this.todo);
25           this.todo = "";
26         }
27       }
28     },
29   }
30 </script>
```

7.3 이벤트 에미터 적용하기

❖예제 07-36 : src/components/ToDoList.vue 변경

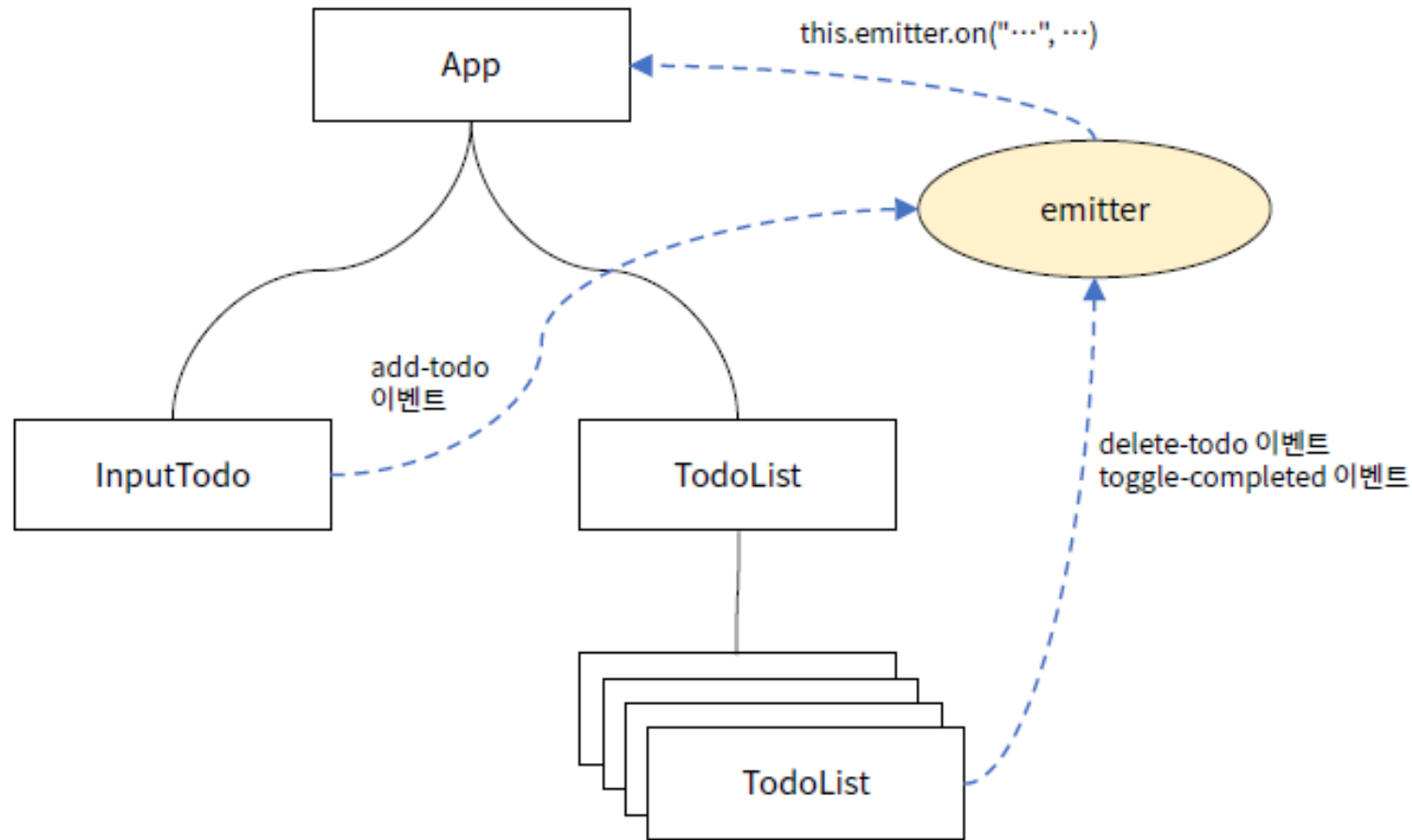
```
1 <template>
2   <div class="row">
3     <div class="col">
4       <ul class="list-group">
5         <ToDoListItem v-for="todoItem in todoList" :key="todoItem.id"
6           :todoItem="todoItem" />
7       </ul>
8     </div>
9   </div>
10 </template>
11
12 <script>
13   import ToDoListItem from './ToDoListItem.vue'
14
15   export default {
16     name : "ToDoList",
17     components : { ToDoListItem },
18     props : {
19       todoList : { type : Array, required:true }
20     }
21   }
22 </script>
```

7.3 이벤트 에미터 적용하기

❖예제 07-37 : src/components/ToDoListItem.vue 변경

```
1 <template>
2   <li class="list-group-item"
3     :class="{ 'list-group-item-success': todoItem.completed } "
4     @click="emitter.emit('toggle-completed', todoItem.id)" >
5     <span class="pointer" :class="{ 'todo-done': todoItem.completed }">
6       {{todoItem.todo}} {{ todoItem.completed ? "(완료)" : "" }}
7     </span>
8     <span class="float-end badge bg-secondary pointer"
9       @click.stop="emitter.emit('delete-todo', todoItem.id)">삭제</span>
10  </li>
11 </template>
12
13 <script>
14   export default {
15     name : "ToDoListItem",
16     props : {
17       todoItem : { type : Object, required: true }
18     }
19   }
20 </script>
```

❖ 이벤트 에미터 적용 예제 구조



이제까지 컴포넌트에 대한 내용을 살펴보았습니다. 컴포넌트의 세세한 기능을 이해하는 것도 필요하지만 속성과 이벤트를 이용한 컴포넌트 간의 정보 전달 방법을 이해하고, 적절히 화면 UI를 컴포넌트 단위로 분할하여 설계해 나가는 능력이 중요하다는 점을 잊지 않았으면 합니다.