

컴포넌트 심화



- ❑ 단일 파일 컴포넌트
- ❑ 컴포넌트에서의 스타일
- ❑ 슬롯
- ❑ 동적 컴포넌트
- ❑ 재귀 컴포넌트

단일 파일 컴포넌트



■ Single File Component

- 전역 수준 컴포넌트의 문제점
 - 빌드 단계가 없으므로 ES2015 등의 최신 문법을 사용할 수 없음
 - CSS를 지원하지 않음. CSS를 모듈화하고 빌드할 수 있는 기능이 없음
 - 컴포넌트의 템플릿이 작성될 때 HTML 파일안에 여러개의 <template />이 작성되어야 함.
- vue-loader 패키지가 단일 파일 컴포넌트를 지원
 - vue-loader가 .vue 파일을 파싱하고 다른 로더들을 활용해 하나의 모듈로 조합함.
 - css-loader를 활용하면 스타일 정보를 모듈화할 수 있음

단일 파일 컴포넌트



❑ 6장의 Todolist 앱을 새롭게 작성!!

❑ webpack-simple 템플릿 코드살펴보기

- 프로젝트 템플릿 생성 & 필요한 패키지 다운로드

```
vue init webpack-simple todolistapp  
cd todolistapp  
npm install
```

- src/App.vue 파일 구조

- 3가지 영역 : <template />, <script />, <style />
- <template />에는 id를 부여하지 않음
- <script /> 내부에서는 반드시 export!!
- 컴포넌트에서 사용할 스타일은 <style /> 내부에...

단일 파일 컴포넌트



- src/main.js : entry point!!

```
import Vue from 'vue'
import App from './App.vue'
new Vue({
  el: '#app',
  render: h => h(App)
})
```

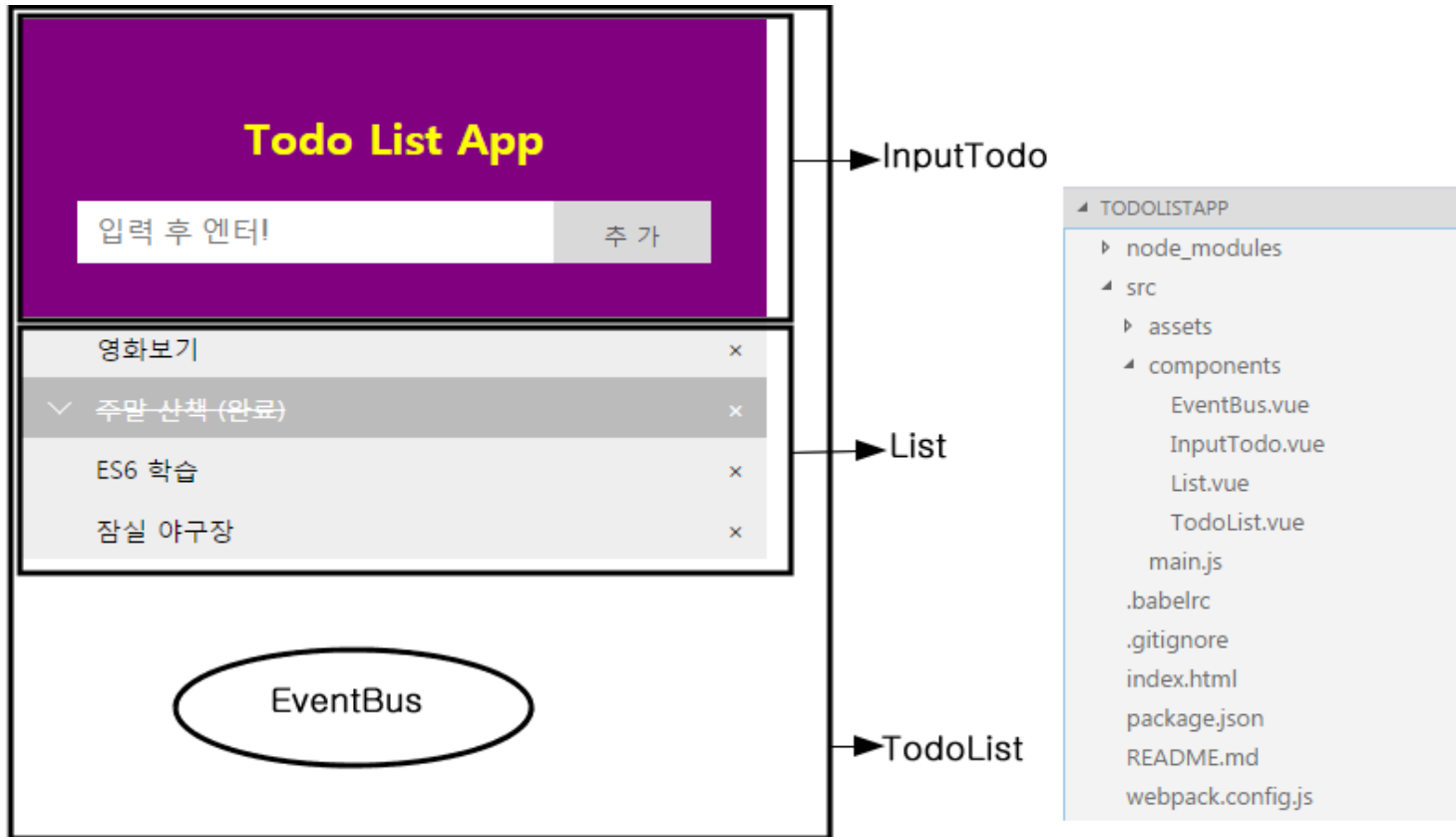
- index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>todolistapp</title>
</head>
<body>
  <div id="app"></div>
  <script src="./dist/build.js"></script>
</body>
</html>
```

단일 파일 컴포넌트



❖ 작성할 4개의 컴포넌트



단일 파일 컴포넌트



- src/components/EventBus.vue

```
<script type="text/javascript">  
import Vue from 'vue';  
  
var eventBus = new Vue();  
export default eventBus;  
</script>
```

단일 파일 컴포넌트



■ src/components/InputTodo.vue

```
<style>
.....(생략)
</style>
<template>
  <div>
    <input class="input" type="text" id="task" v-model.trim="todo"
      placeholder="입력 후 엔터!" v-on:keyup.enter="addTodo">
    <span class="addbutton" v-on:click="addTodo">추 가</span>
  </div>
</template>
<script type="text/javascript">
import EventBus from './EventBus.vue'

export default {
  name: 'input-todo',
  data: function() {
    return { todo: "" }
  },
  methods: {
    addTodo: function() {
      EventBus.$emit('add-todo', this.todo);
      this.todo = "";
    }
  }
}
</script>
```

단일 파일 컴포넌트



■ src/components/List.vue

```
<style>
.....(생략)
</style>
<template>
  <ul id="todolist">
    <li v-for="(a, index) in todolist"
      v-bind:class="checked(a.done)"
      v-on:click="doneToggle(index)">
      <span>{{ a.todo }}</span>
      <span v-if="a.done"> (완료)</span>
      <span class="close"
        v-on:click.stop="deleteTodo(index)">
        &#x00D7;</span>
    </li>
  </ul>
</template>
<script type="text/javascript">
import EventBus from './EventBus.vue'

export default {
  created : function() {
    EventBus.$on('add-todo', this.addToDo);
  },

```

```
data : function() {
  return {
    todolist : [
      { todo : "영화보기", done:false },
      { todo : "주말 산책", done:true },
      { todo : "ES6 학습", done:false },
      { todo : "잠실 야구장", done:false },
    ]
  },
},
methods : {
  checked : function(done) {
    if(done) return { checked:true };
    else return { checked:false };
  },
  addToDo : function(todo) {
    if (todo !== "") {
      this.todolist.push({ todo : todo, done:false });
    }
  },
  doneToggle : function(index) {
    this.todolist[index].done = !this.todolist[index].done;
  },
  deleteTodo : function(index) {
    this.todolist.splice(index,1);
  }
}
}
</script>
```


단일 파일 컴포넌트



■ src/components/ToDoList.vue

```
<style>
.....(생략)
</style>
<template>
  <div id="todolistapp">
    <div id="header" class="header">
      <h2>Todo List App</h2>
      <input-todo />
    </div>
    <list></list>
  </div>
</template>
<script type="text/javascript">
import InputTodo from './InputTodo.vue';
import List from './List.vue';

export default {
  name : 'todo-list',
  components : { InputTodo, List }
}
</script>
```

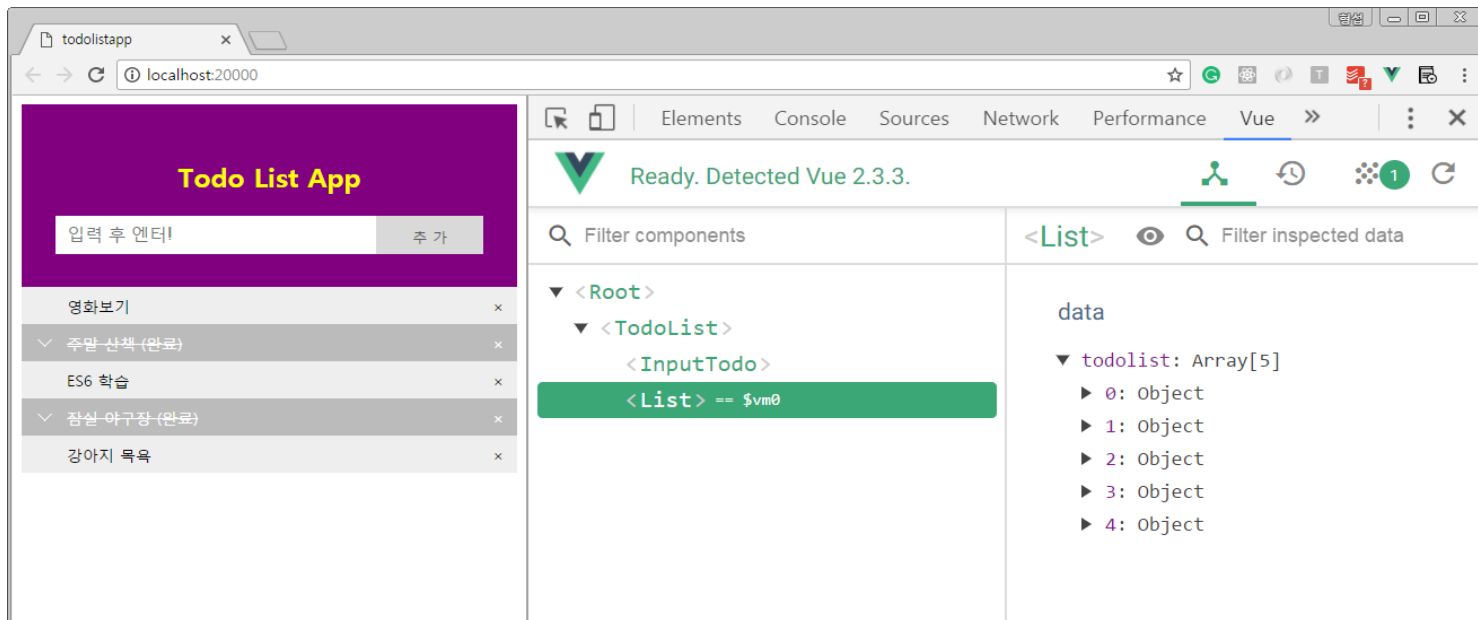
단일 파일 컴포넌트



■ src/main.js

```
import Vue from 'vue'
import TodoList from './components/TodoList.vue'

new Vue({
  el: '#app',
  render: h => h(TodoList)
})
```



컴포넌트에서의 스타일



■ 단순히 <style /> 을 이용한다면...

- 다른 컴포넌트에서 동일한 class 명을 사용하는 경우 충돌이 발생함.
- 특정 컴포넌트만의 스타일을 지정하려면?
 - 범위 CSS(Scoped CSS)
 - CSS 모듈(CSS Module)

컴포넌트에서의 스타일



■ 범위 CSS(Scoped CSS)

- styletest 프로젝트
- src/components/Child1.vue

```
<template>
  <div class="main">{{msg}}</div>
</template>
<script>
export default {
  name: 'child1',
  data () {
    return {
      msg: 'Child1'
    }
  }
}
</script>
<style>
  .main { border:solid 1px black; background-color:yellow; }
</style>
```

컴포넌트에서의 스타일



❑ 범위 CSS(이어서)

- src/components/Child2.vue

```
<template>
  <div class="main">{{msg}}</div>
</template>
<script>
export default {
  name: 'child2',
  data () {
    return {
      msg: 'Child2'
    }
  }
}
</script>
<style>
  .main { border:solid 1px black; background-color:aqua; }
</style>
```

- Child1과 Child2 컴포넌트에는 main 클래스로 서로 다른 배경색의 스타일이 정의되어 있음.

컴포넌트에서의 스타일



❖ 범위 CSS(이어서)

▪ src/App.vue

```
<template>
  <div id="app">
    <h2>{{msg}}</h2>
    <child1 />
    <child2 />
  </div>
</template>

<script>
import Child1 from './components/Child1.vue'
import Child2 from './components/Child2.vue'

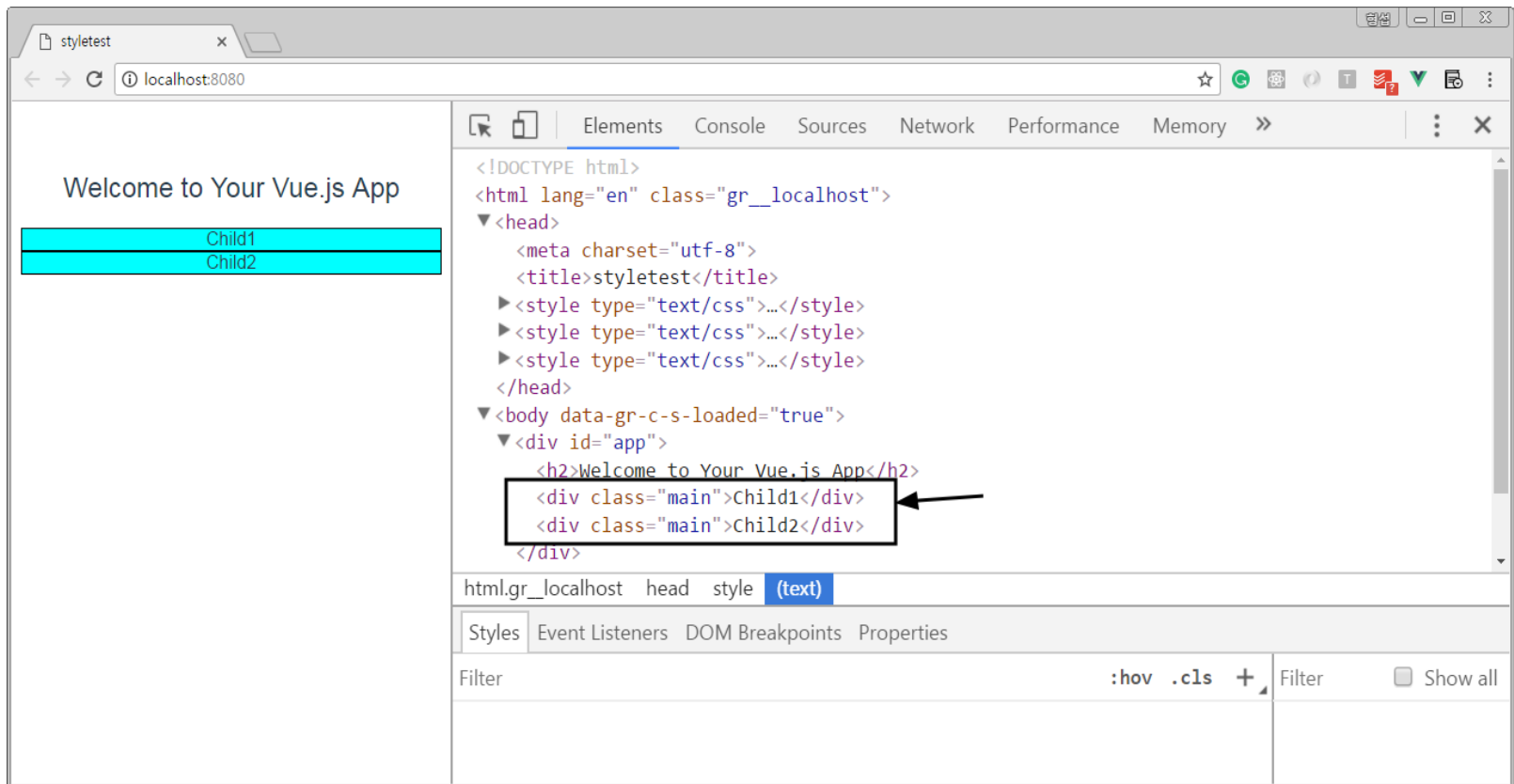
export default {
  name: 'app',
  components : { Child1, Child2 },
  data () {
    return { msg: 'Welcome to Your Vue.js App' }
  }
}
</script>
<style>
.....
</style>
```

컴포넌트에서의 스타일



범위 CSS(이어서)

- 실행 결과 : 스타일 중복!!

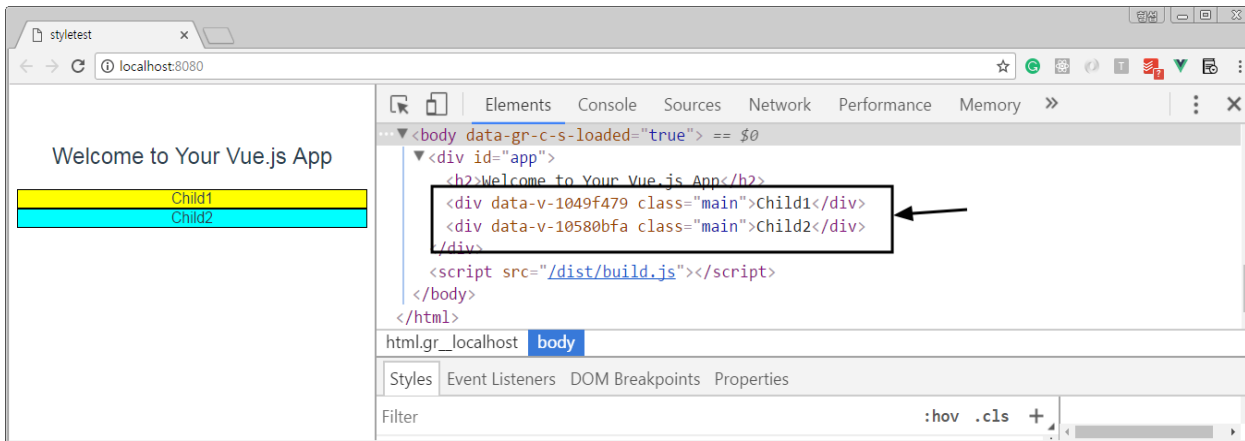


컴포넌트에서의 스타일



범위 CSS(이어서)

- 범위 CSS로 변경하면?
 - Child1, Child2 컴포넌트의 `<style />`을 `<style scoped>`로 변경한 후 확인
- 하나의 컴포넌트에 여러개의 `<style />`을 작성할 수 있음
 - 전역 CSS, 범위 CSS로 구분하여 적용할 수 있음



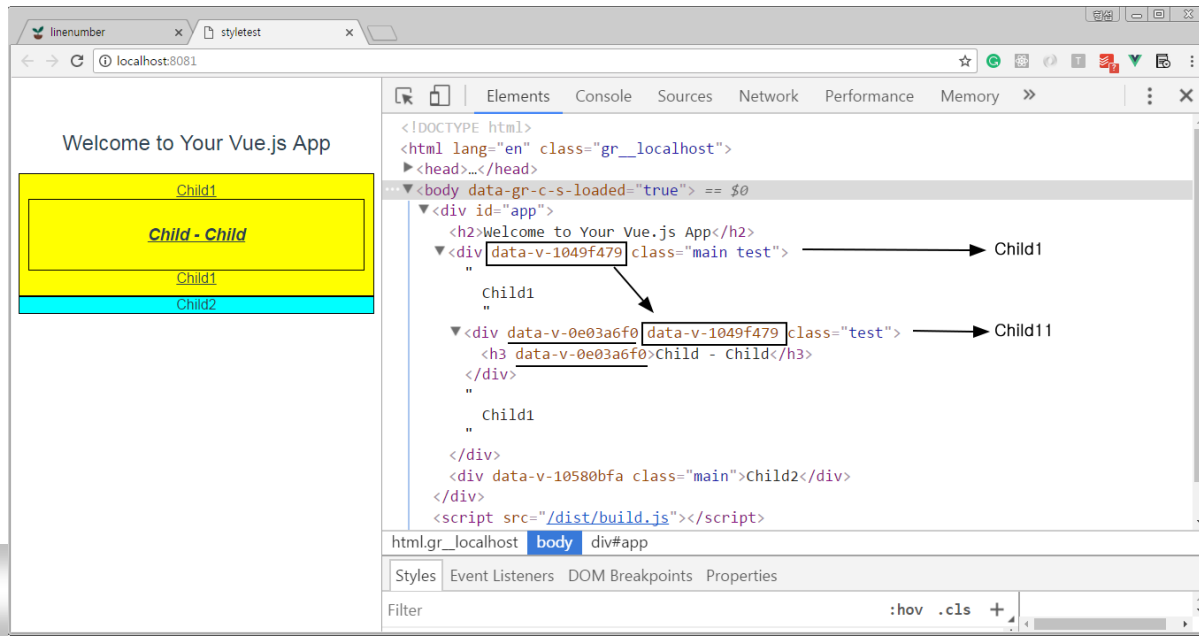
```
<style type="text/css">
  .main[data-v-1049f479] { border:solid 1px black; background-color:yellow;
}
```


컴포넌트에서의 스타일



❖ 범위 CSS 적용시 주의사항

- 범위 CSS는 특성 선택자를 이용하기 때문에 스타일의 적용속도가 느림
 - 반드시 속도가 빠른 선택자로 요소를 선택해 스타일을 적용하도록 해야 함.
- 부모 컴포넌트에 적용된 범위 CSS는 하위 컴포넌트에도 반영됨.
 - 부모 컴포넌트에서 범위 CSS를 적용하기 위해 생성되는 Attribute가 자식 컴포넌트의 루트요소에도 등록되기 때문!!
 - 예제 09-12~13 적용 후 실행 결과



컴포넌트에서의 스타일



■ CSS 모듈

- CSS 스타일을 마치 객체처럼 다룰 수 있도록 함.
- `$style` 이라는 계산형 속성을 통해서 직접 이용할 수 있음.
- `src/components/Modul1.vue`

```
<template>
  <div>
    <button :class="$style.hand"> CSS Module을 적용한 버튼 </button>
  </div>
</template>

<script>
export default {
  created() {
    console.log(this.$style);
  }
}
</script>

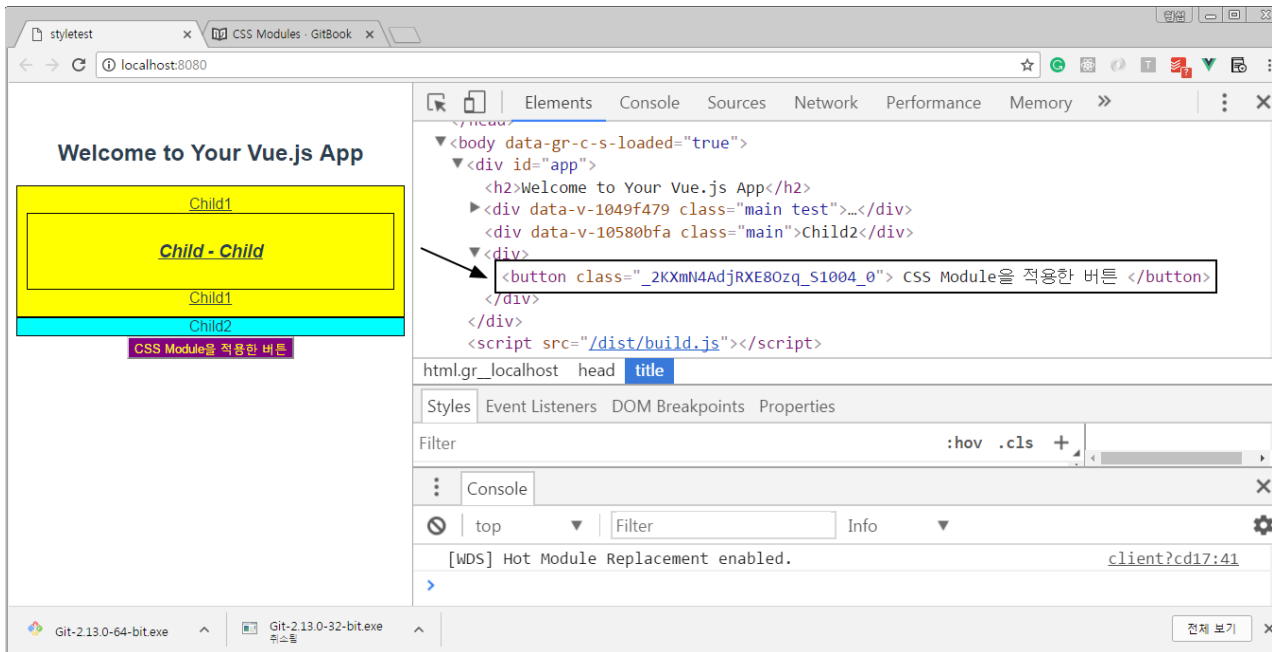
<style module>
  .hand { cursor:pointer; background-color:purple; color:yellow; }
</style>
```

컴포넌트에서의 스타일



■ CSS 모듈

■ 예제 09-15까지 적용 후 실행 결과



■ 적용해야할 클래스가 여러개라면 배열 문법을 이용해 한번에 적용 가능

```
<div v-bind:class="[$style.box, $style.border]">Hello World</div>
```



❖ props, event 를 이용한 컴포넌트간의 정보 교환

- 유용하긴 하지만 속성으로 HTML 문자열을 전달하는 것은 쉽지 않음
 - 슬롯은 이런 불편함을 해소할 수 있도록 함.
 - 부모컴포넌트에서 자식 컴포넌트를 사용할 때 지정한 콘텐츠가 슬롯으로 전달됨.

```
<template>
  <div class="container">
    <div class="header">{{headerText}}</div>
    <div class="content">
      <slot></slot> ←
    </div>
    <div class="footer">{{footerText}}</div>
  </div>
</template>

<script>
export default {
  props : [ 'headerText', 'footerText' ]
}
</script>
```

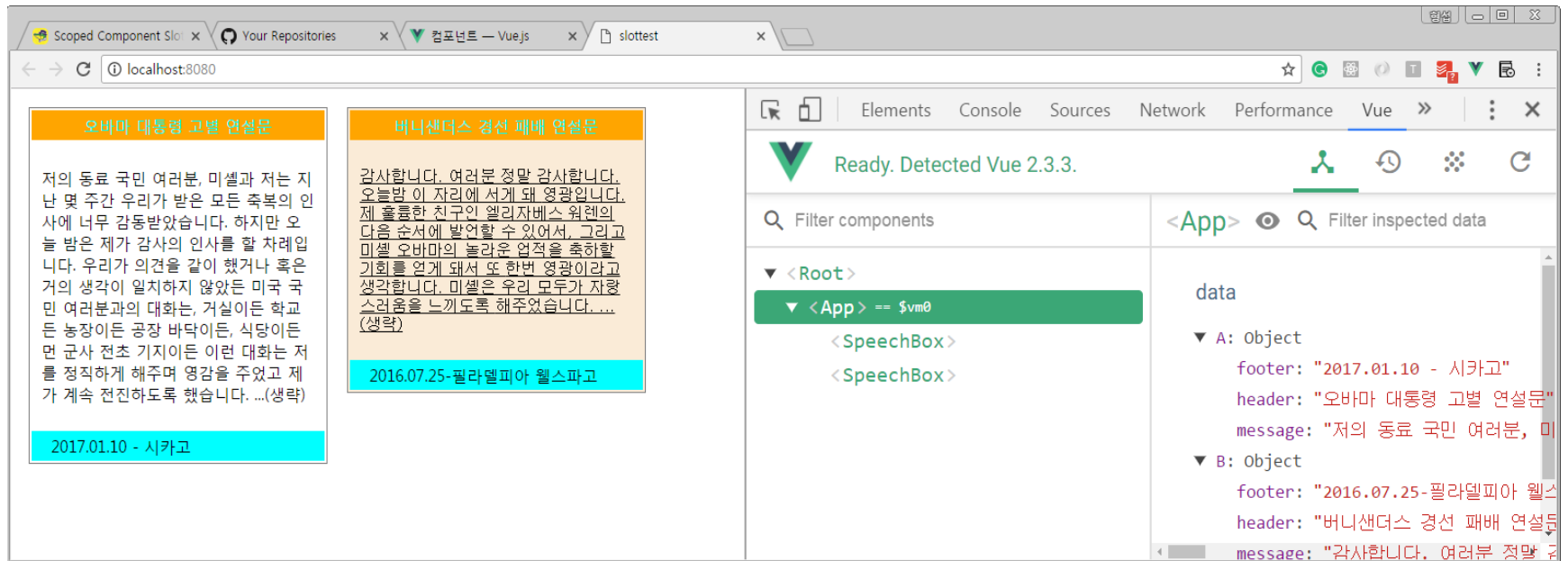
```
<speech-box :headerText="A.header" :footerText="A.footer">
  <div>
    <p>
      {{A.message}}
    </p>
  </div>
</speech-box>
```

슬롯



❖ 슬롯의 기본 사용법

■ 예제 09-16~17 실행 결과



■ 기억!

- 부모컴포넌트에서 자식 컴포넌트로 스타일도 함께 전달됨..
- 예제 09-17의 11,13행과 45, 46행 참조



■ 명명된 슬롯 : Named Slot

- 이름을 부여하여 여러 개의 슬롯을 사용할 수 있음
- 예제 09-18~20

<자식 컴포넌트>

```
<template>
  <div id="pagewrap">
    <header>
      <slot name="header"></slot>
    </header>
    <aside id="sidebar">
      <slot name="sidebar"></slot>
    </aside>
    <section id="content">
      <slot name="content"></slot>
    </section>
    <footer>
      <slot name="footer"></slot>
    </footer>
  </div>
</template>
```

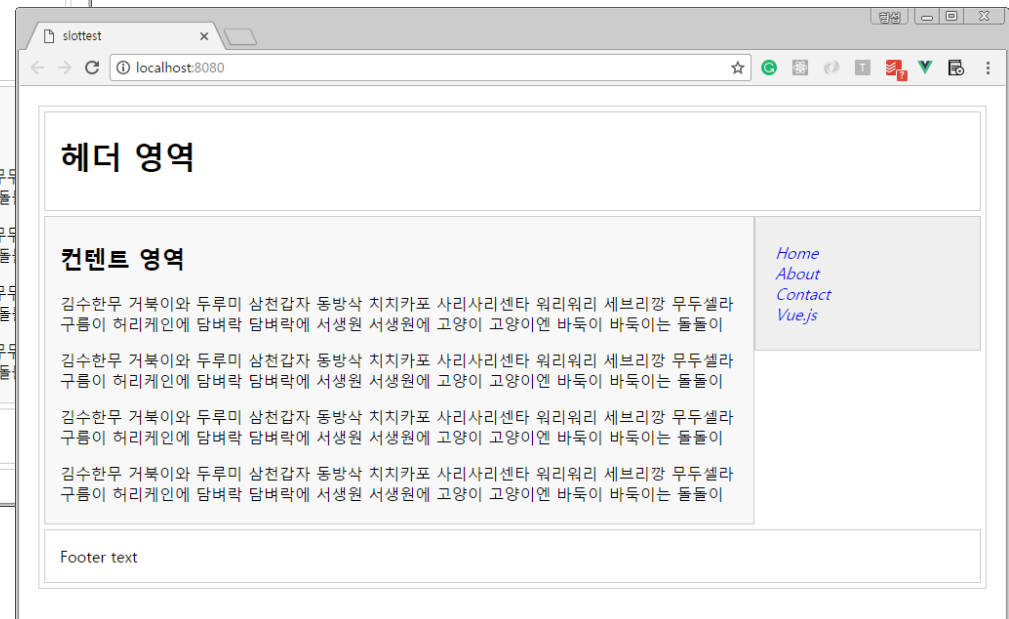
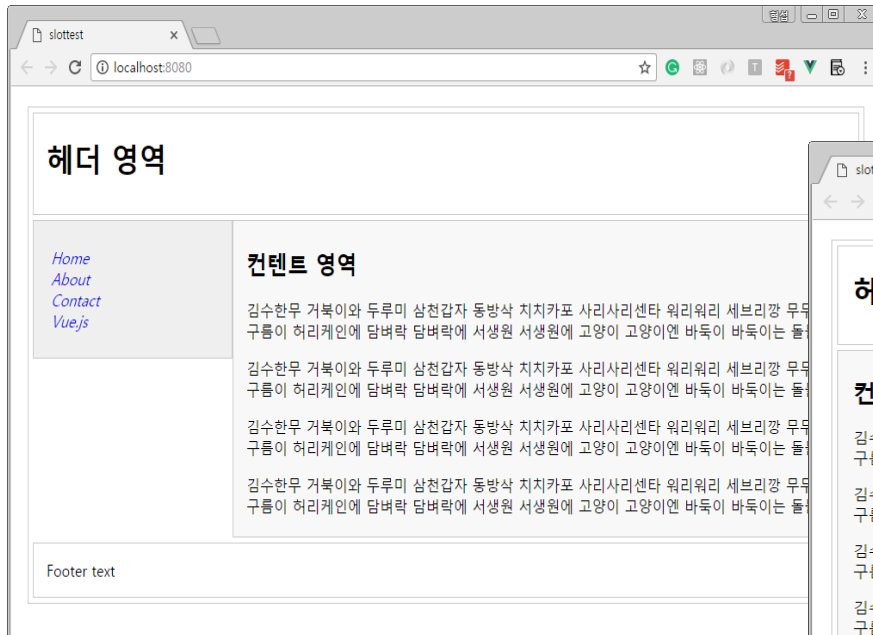
<부모 컴포넌트>

```
<template>
  <div id="app">
    <layout>
      <h1 slot="header">헤더 영역</h1>
      <div slot="sidebar">
        <ul class="menu">
          <li v-for="sidebar in sidebars">
            <a v-bind:href="sidebar.link">{{sidebar.menu}}</a>
          </li>
        </ul>
      </div>
      <div slot="content">
        <h2>컨텐츠 영역</h2>
        .....
      </div>
      <p slot="footer">Footer text</p>
    </layout>
  </div>
</template>
```



■ 실행 결과

- AppNamed.vue와 같은 부모 컴포넌트가 여러개 있고, 레이아웃을 변경해야 하는 상황이라면?
 - NamedSlot.vue 컴포넌트에서만 레이아웃을 변경하면 끝!





❧ 범위 슬롯(Scoped Slot)

- 범위 슬롯은 자식 -> 부모로 속성을 전달하여 부모 컴포넌트 측에서 출력할 내용을 커스터마이징!!
 - 기존 슬롯 : 부모 컴포넌트 -> 자식컴포넌트로 정보 전달
- 예제 09-21 : ScopedSlot.vue

```
<template>
  <div class="child">
    X : <input type="text" v-model="x" /><br />
    Y : <input type="text" v-model="y" /><br />
    <slot name="type1" :cx="x" :cy="y"></slot>
    <slot name="type2" :cx="x" :cy="y"></slot>
  </div>
</template>
<script>
export default {
  data() {
    return { x:4, y:5 };
  }
}
</script>
<style scoped>
  .child { padding:5px; border:solid 1px gray; }
</style>
```




❖ 범위 슬롯(이어서)

▪ 예제 09-22 : AppScoped.vue

```
<template>
  <div class="parent">
    <child>
      <template slot="type1" scope="p1">
        <div>{{p1.cx }} + {{p1.cy}} = {{ parseInt(p1.cx) + parseInt(p1.cy) }}</div>
      </template>
      <template slot="type2" scope="p2">
        <div>{{p2.cx }} 더하기 {{p2.cy}} 는 {{ parseInt(p2.cx) + parseInt(p2.cy) }}입니다.</div>
      </template>
    </child>
  </div>
</template>

<script>
import Child from './components/ScopedSlot.vue'
export default {
  components : { Child }
}
</script>

<style>
.parent { border:dashed 2px black; padding:5px; }
</style>
```

슬롯



■ 예제 09-21~23 실행 결과





Dynamic Component

- 화면의 동일한 위치에 여러 컴포넌트를 나타내고 싶을 때 사용할 수 있음
- <component> 요소 사용
- v-bind:is 를 이용해 어떤 컴포넌트를 그 위치에 나타낼지를 결정하면 됨.

예제

- 프로젝트 초기화

```
vue init webpack-simple dynamictest  
cd dynamictest  
npm install
```

- 메뉴로 이동시킬 화면을 제공하는 컴포넌트 작성
 - Home.vue, About.vue, Contact.vue
 - 예제 09-24 참조

동적 컴포넌트



■ src/App.vue : 예제 09-25

```
<template>
<div>
  <div class="header">
    <h1 class="headerText">(주) OpenSG</h1>
    <nav>
      <ul>
        <li>
          <a href="#" @click="changeMenu('home')">Home</a>
        </li>
        <li>
          <a href="#" @click="changeMenu('about')">About</a>
        </li>
        <li>
          <a href="#" @click="changeMenu('contact')">Contact</a>
        </li>
      </ul>
    </nav>
  </div>

  <div class="container">
    <component v-bind:is="currentView"></component>
  </div>
</div>
</template>
```

동적 컴포넌트



■ src/App.vue : 예제 09-25 (이어서)

```
<script>
import Home from './components/Home.vue';
import About from './components/About.vue';
import Contact from './components/Contact.vue';

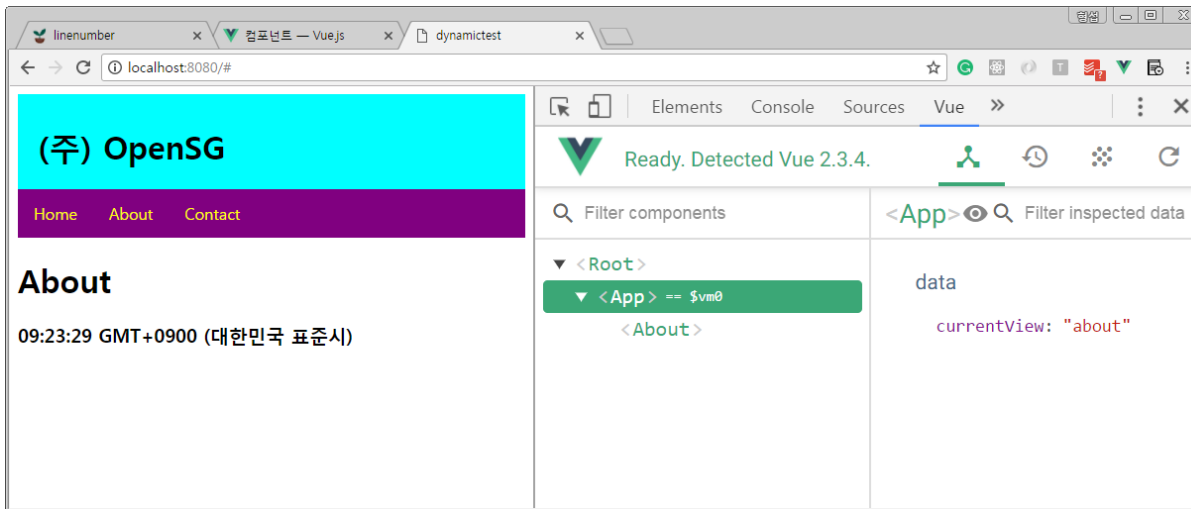
export default {
  components : { Home, About, Contact },
  data() {
    return { currentView : 'home' }
  },
  methods : {
    changeMenu(view) {
      this.currentView = view;
    }
  }
}
</script>

<style scoped>
.header { background-color:aqua; padding: 10px 0px 0px 0px; }
.headerText { padding: 0px 20px 0px 20px; }
ul { list-style-type: none; margin: 0; padding: 0;
  overflow: hidden; background-color: purple; }
li { float: left; }
li a { display: block; color: yellow; text-align: center;
  padding: 14px 16px; text-decoration: none; }
li a:hover { background-color: aqua; color:black; }
</style>
```

동적 컴포넌트



■ 예제 09-24~25 실행 결과



- 메뉴를 변경할때마다 시간이 바뀜 --> 매번 렌더링한다는 의미
- 정적 콘텐츠일때는 매번 실행할 필요가 없음 -> <keep-alive> 요소로 해결



■ <keep-alive> 적용

```
[ 예제 09-25의 11행을 다음과 같이 수정 ]
<keep-alive include="about,home">
  <component v-bind:is="currentView"></component>
</keep-alive>
```

```
[ Home, About, Contact 컴포넌트에서 name 옵션 부여 ]
<template>
  <div>
    <h1>Home</h1>
    <h3>{{ (new Date()).toString() }}</h3>
  </div>
</template>
<script>
export default {
  name : "home"
}
</script>
```

- 포함, 배제를 위해 컴포넌트는 name 옵션을 지정해야 함.
- include, exclude 특성을 이용

재귀 컴포넌트



■ Recursive Component

- 템플릿에서 자기자신을 호출하는 컴포넌트
- 반드시 name 옵션이 지정되어야 함.

■ 예제 : Tree 컴포넌트

- 예제 09-27 : src/components/Tree.vue

```
<template>
  <ul>
    <li v-for="s in subs" v-bind:class="s.type">
      {{s.name}}
      <tree :subs="s.subs"></tree>
    </li>
  </ul>
</template>

<script>
export default {
  name : 'tree',
  props : [ 'subs' ]
}
</script>
```


재귀 컴포넌트



■ 예제 09-28 : src/component/About.vue

```
<template>
  <div>
    <h1>About</h1>
    <h3>{{ (new Date()).toString() }}</h3>
    <h4>조직도</h4>
    <tree :subs="orgcharts"></tree>
  </div>
</template>
<script>
import Tree from './Tree.vue';

export default {
  name : "about",
  components : { Tree },
  data : function() {
    return {
      orgcharts : [
        {
          name : "(주) OpenSG", type:"company",
          subs : [
            { name: "SI 사업부", type:"division",
              subs : [
                { name: "SI 1팀", type:"team" },
                { name: "SI 2팀", type:"team" }
              ]
            }
          ]
        },

```

```
        { name: "BI 사업부", type:"division",
          subs : [
            { name: "BI 1팀", type:"team" },
            { name: "BI 2팀", type:"team" },
            { name: "BI 3팀", type:"team" }
          ]
        },
        { name: "솔루션 사업부", type:"division",
          subs : [
            { name: "ESM팀", type:"team" },
            { name: "MTS팀", type:"team" },
            { name: "ASF팀", type:"team" }
          ]
        },
        { name: "총무팀", type:"team" },
        { name: "인사팀", type:"team" }
      ]
    }
  ],
}
</script>
<style>
.....(생략)
</style>
```

재귀 컴포넌트



■ 예제 09-27~28 실행 결과

The screenshot displays a web browser window with the URL `localhost:8080/#` and a Vue.js application. The application has a cyan header with the text **(주) OpenSG** and a purple navigation bar with links [Home](#), [About](#), and [Contact](#). The main content area shows the **About** page, which includes the time **12:14:11 GMT+0900 (대한민국 표준시)** and an organizational chart.

The organizational chart is structured as follows:

- (주) OpenSG
 - SI 사업부
 - SI 1팀
 - SI 2팀
 - BI 사업부
 - BI 1팀
 - BI 2팀
 - BI 3팀
 - 솔루션 사업부
 - ESM팀
 - MTS팀
 - ASF팀
 - 총무팀
 - 인사팀

The Vue DevTools component inspector is open on the right, showing the component tree. The tree structure is as follows:

- ▼ <App>
 - <Home> inactive
 - ▼ <About>
 - ▼ <Tree>
 - ▼ <Tree>
 - ▼ <Tree>
 - <Tree>
 - <Tree>
 - ▶ <Tree>
 - ▶ <Tree>
 - <Tree>
 - <Tree>
 - <Contact> inactive

The right panel of the component inspector contains the text: "Select a component instance to inspect."