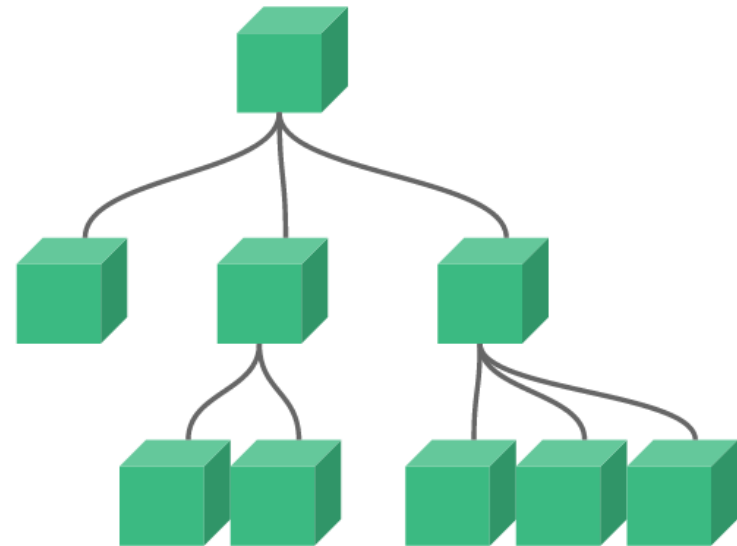
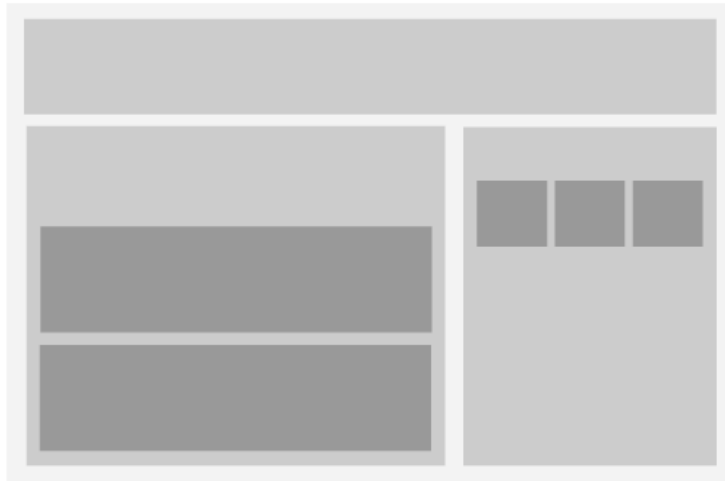


# Vue Component

**Bok, Jong Soon**  
**javaexpert@nate.com**  
**<https://github.com/swacademy/Vue.js>**

# Vue Component

- 화면의 영역을 구분하여 개발할 수 있는 Vue의 기능
- 재사용 가능한 Vue Instance이다.



# Vue Component (Cont.)

## ■ Component 장점

- 재사용성이 향상되므로 개발 효율성이 좋아진다.
- 이미 사용하는 Component를 재사용하므로 품질이 보장된다.
- 적절히 분할할 Component가 느슨하게 결합하므로 유지 보수성이 향상된다.
- Capsule化를 통해 개발 작업에서 신경 써야할 부분을 최소한으로 제한할 수 있다.

# Vue Component (Cont.)

## ■ Component 생성 코드 형식

- 하나의 Component의 구성은 **template, script, style**로 구성될 수 있다.
- 즉, 각각 HTML, JavaScript, CSS로 대응되는 개념이다.
- 최소한 한 개의 **<template>** 은 있어야 한다.
- Component를 생성하는 Code 형식은 다음과 같다.

```
Vue.component('Component 이름', {  
  template : 'HTML Code들',  
  props : ,  
});
```

# Vue Component (Cont.)

## ■ Component 생성 후 표시하기

- 간단한 App Header Component를 생성해 보자.

```
Vue.component('app-header', {  
  template : '<h1>Header Component</h1>'  
});
```

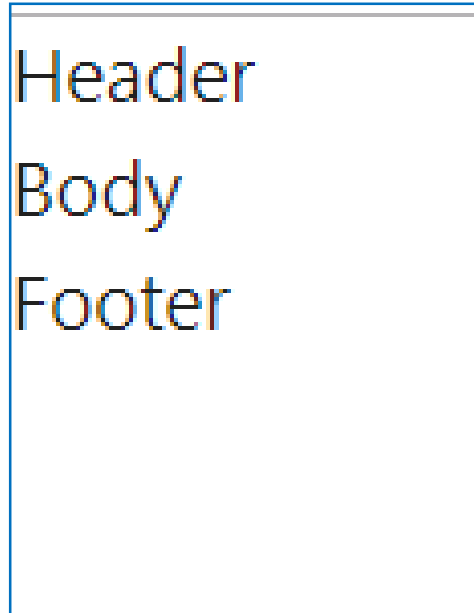
- 이제 등록한 Component를 화면에서 표시하려면 아래와 같이 Component Tag(Component 이름)을 추가한다.

```
<div id="app">  
  <app-header></app-header>  
</div>
```

- 그러면 아래와 같이 표시된다.

```
<div id="app">  
  <h1>Header Component</h1>  
</div>
```

# Vue Component (Cont.)



```
<div id="app">
  <my-header></my-header>
  <my-body></my-body>
  <my-footer></my-footer>
</div>
<script>
  Vue.component('my-header', {
    template : '<div>Header</div>'
  });
  Vue.component('my-body', {
    template : '<div>Body</div>'
  });
  Vue.component('my-footer', {
    template : '<div>Footer</div>'
  });
  let vm = new Vue({
    el : '#app'
  })
</script>
```

# Lab : @vue/cli를 이용하여 Vue Component 구성하기



# Vue Component (Cont.)

- Component 등록 방법 2가지
  - 전역 Component 등록하는 방법
    - 앞 Slide 방법
  - 지역 Component 등록하는 방법

```
var appHeader = {  
  template: '<h1>Header Component</h1>'  
}
```

```
new Vue({  
  components: {  
    'app-header': appHeader  
  }  
})
```





---



# Lab : Global Component 만들기

# Lab - 전역 Component 만들기

```
8     <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
9 </head>
10 <body>
11     <ul id="example">
12         <list-item></list-item>
13     </ul>
14
15     <script>
16         Vue.component('list-item', {
17             template : '<li>foo</li>'
18         })
19
20         new Vue({
21             el : '#example'
22         })
23     </script>
24 </body>
```

← → ↻ ⓘ 127.0.0.1:5500/vuecomponent.html

- foo

# Lab - 전역 Component 만들기 (Cont.)

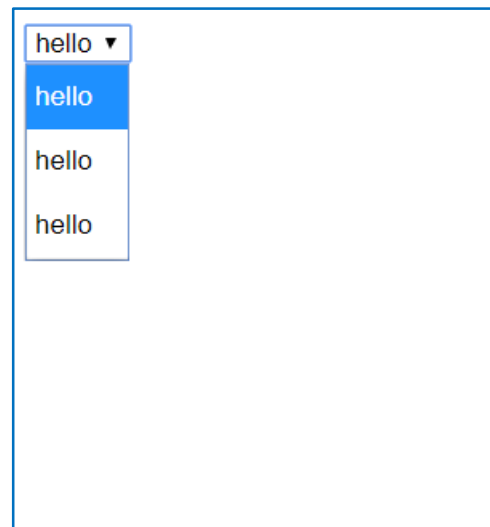
```
8     <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
9 </head>
10 <body>
11     <ul id="example">
12         <list-item></list-item>
13     </ul>
14
15     <script>
16         Vue.component('list-item', {
17             template : '<li>foo {{ contents }} </li>',
18             data : function(){
19                 return {contents : 'bar'}
20             }
21         })
22
23         new Vue({
24             el : '#example'
25         })
26     </script>
27 </body>
```

← → ↻ ⓘ 127.0.0.1:5500/vuecomponent.html

- foo bar

# Lab - 전역 Component 만들기 (Cont.)

```
8      <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
9  </head>
10 <body>
11   <div id="app">
12     <select-component></select-component>
13   </div>
14
15   <script id="selectTemplate">
16     <select>
17       <option-component></option-component>
18       <option-component></option-component>
19       <option-component></option-component>
20     </select>
21   </script>
22   <script>
23     Vue.component('select-component', {
24       template: '#selectTemplate'
25     });
26     Vue.component('option-component', {
27       template : '<option>hello</option>'
28     })
29     new Vue({
30       el : '#app'
31     })
32   </script>
33 </body>
```



## Lab - 전역 Component 만들기 (Cont.)

```
8      <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
9  </head>
10 <body>
11     <div id="fruits-list">
12         <fruits-list-title></fruits-list-title>
13         <fruits-list-description></fruits-list-description>
14         <fruits-list-table></fruits-list-table>
15     </div>
16
17     <script>
18         Vue.component('fruits-list-title', {
19             template : '<h1>과일 목록</h1>'
20         })
21         Vue.component('fruits-list-description', {
22             template: '<p>각 계절 대표적 과일의 목록</p>'
23         })
```

# Lab - 전역 Component 만들기 (Cont.)

```
24 Vue.component('fruits-list-table', {
25   template: `
26     <table>
27       <tr>
28         <th>계절</th>
29         <th>과일</th>
30       </tr>
31       <tr>
32         <td>봄</td>
33         <td>딸기</td>
34       </tr>
35       <tr>
36         <td>여름</td>
37         <td>수박</td>
38       </tr>
39       <tr>
40         <td>가을</td>
41         <td>포도</td>
42       </tr>
43       <tr>
44         <td>겨울</td>
45         <td>귤</td>
46       </tr>
47     </table>`
48 })
```

```
50 new Vue({
51   el : '#fruits-list'
52 })
53 </script>
54 </body>
```

← → ↻ ⓘ 127.0.0.1:5500/vuecomponent.html

## 과일 목록

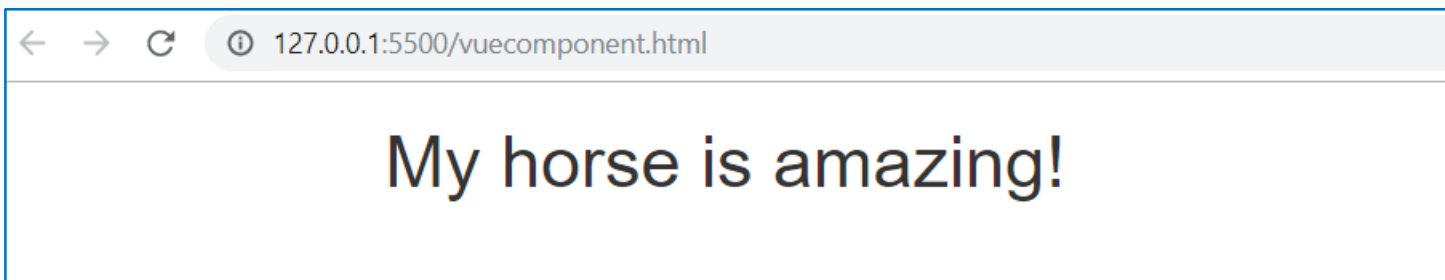
각 계절 대표적 과일의 목록

### 계절 과일

봄 딸기  
여름 수박  
가을 포도  
겨울 귤

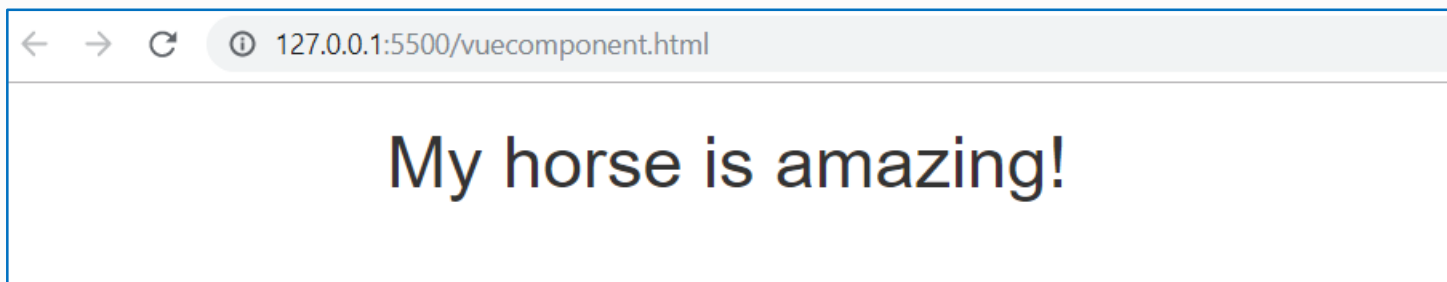
# Lab - 전역 Component 만들기

```
8   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
9   <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
10  </head>
11  <body>
12    <div class="container">
13      <story></story>
14    </div>
15
16    <script>
17      Vue.component('story', {
18        template: '<h1>My horse is amazing!</h1>'
19      });
20
21      new Vue({
22        el : '.container'
23      })
24    </script>
25  </body>
```



# Lab - 전역 Component 만들기

```
8   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
9   <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
10  </head>
11  <body>
12    <div class="container">
13      <story></story>
14    </div>
15    <template id="story-template">
16      <h1>My horse is amazing!</h1>
17    </template>
18    <script>
19      Vue.component('story', {
20        template: '#story-template'
21      });
22
23      new Vue({
24        el: '.container'
25      })
26    </script>
27  </body>
```







---

# Lab : Local Component 만들기



# Lab – 지역 Component 만들기

```
8   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
9   <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
10  </head>
11  <body>
12    <div class="container">
13      <story v-bind:story="{plot: 'My horse is amazing.', writer: 'Mr. Weebl'}">
14      </story>
15      <story v-bind:story="{plot: 'Narwhals invented Shish Kebab.', writer: 'Mr. Weebl'}">
16      </story>
17      <story v-bind:story="{plot: 'The dark side of the Force is stronger.', writer: 'Darth Vader'}">
18      </story>
19    </div>
20    <template id="story-template">
21      <h1>{{ story.writer }} said "{{ story.plot }}"</h1>
22    </template>
23    <script>
24      Vue.component('story', {
25        props: ['story'],
26        template: "#story-template"
27      });
28
29      new Vue({
30        el: '.container'
31      })
32    </script>
33  </body>
```

127.0.0.1:5500/vuecomponent.html

Mr. Weebl said "My horse is amazing."

Mr. Weebl said "Narwhals invented Shish Kebab."

Darth Vader said "The dark side of the Force is stronger."



# Lab : @vue/cli를 이용한 Global, Local Component 생성하기





---



# Lab : Component 와 Instance와의 관계

# template을 만드는 그 외의 방법들

- template property의 값 형태로 HTML tag를 문자열로 작성하는 방법 외에 다양한 방법이 있다.
- text/x-template
- render()
- 단일 File Component
- inline-template
- JSX

# text/x-template

- `<script>` tag에 **text/x-template** type을 사용하면, Browser에 DOM이라고 인식되지 않아 **template**으로 활용가능.

## 과일 목록

- Mango
- Apple
- Grape

```
<div id="app">
  <my-component-h1></my-component-h1>
  <my-component-contents></my-component-contents>
</div>
<script type="text/x-template" id="child-template1">
  <h1>과일 목록</h1>
</script>
<script type="text/x-template" id="child-template2">
  <ul>
    <li>Mango</li>
    <li>Apple</li>
    <li>Grape</li>
  </ul>
</script>
<script>
  Vue.component('my-component-h1', {
    template : '#child-template1'
  })
  Vue.component('my-component-contents', {
    template : '#child-template2'
  })
  new Vue({
    el : '#app'
  })
</script>
```

DOM으로 인식되지 않는다는 것은 text/x-template은 Browser에서 인식하지 못하는 MIME type이라는 뜻. 그래서 Browser에서는 무시되고, Vue.js만 인식한다.

# render()

- template의 약점 중 한 가지는 프로그램적으로 작성하기 어렵다는 점.
- Component에서 code를 사용할 수 있도록 **render()** option 제공.

2019-12-11

```
<div id="app">
  <input-date-with-today></input-date-with-today>
</div>
<script>
  Vue.component('input-date-with-today', {
    render : function(createElement){
      return createElement(
        'input',
        {
          attrs : {
            thype : 'date',
            value : new Date().toISOString().substring(0, 10)
          }
        }
      )
    }
  })
  new Vue({
    el : '#app'
  })
</script>
```

# 단일 File Component

- Component 단위로 JavaScript, HTML, CSS의 Set
- 확장자는 **.vue**의 단일 File.
- 어느 정도 규모 이상의 Project를 개발시 사용
- **vue-cli** 또는 **@vue/cli**로 개발




# inline-template

- 특별한 속성인 **inline-template**을 사용자 정의 tag에 지정하면, 내부에 작성한 HTML을 template으로 사용할 수 있다.


## 과일 목록

- Mango
- Banana
- Apple
- Grape

```
<div id="app">
  <my-component inline-template>
    <div>
      <h1>과일 목록</h1>
      <ul>
        <li v-for="fruit in fruits">{{ fruit }}</li>
      </ul>
    </div>
  </my-component>
</div>
<script>
  Vue.component('my-component', {
    data : function(){
      return {
        fruits : ['Mango', 'Banana', 'Apple', 'Grape']
      }
    }
  })
  new Vue({
    el : '#app'
  })
</script>
```



# Lab : Vue에서 Component template을 정의하는 7가지 방법



# template이 DOM으로 인식되는 경우

- HTML Code가 Browser에 의해 해석 → JavaScript 실행되며, 이 때 Vue.js가 template을 찾고 처리한다.
- 이런 순서로 인해 발생할 수 있는 문제들
  - Mount 요소 #app 내부에 직접 작성한 template
  - inline-template으로 작성한 template
  - 이러한 template 내부에서 사용하는 Slot Contents
- 주의할 점
  - Component 명명 규칙 → Kebab Case 사용할 것
  - HTML 내포 가능한 요소 규칙
    - `<table>`, `<select>` 와 같은 몇 가지 요소는 내부에 가질 수 있는 요소 제한 있음.
    - 이럴 경우 `is="컴포넌트이름"`으로 해결

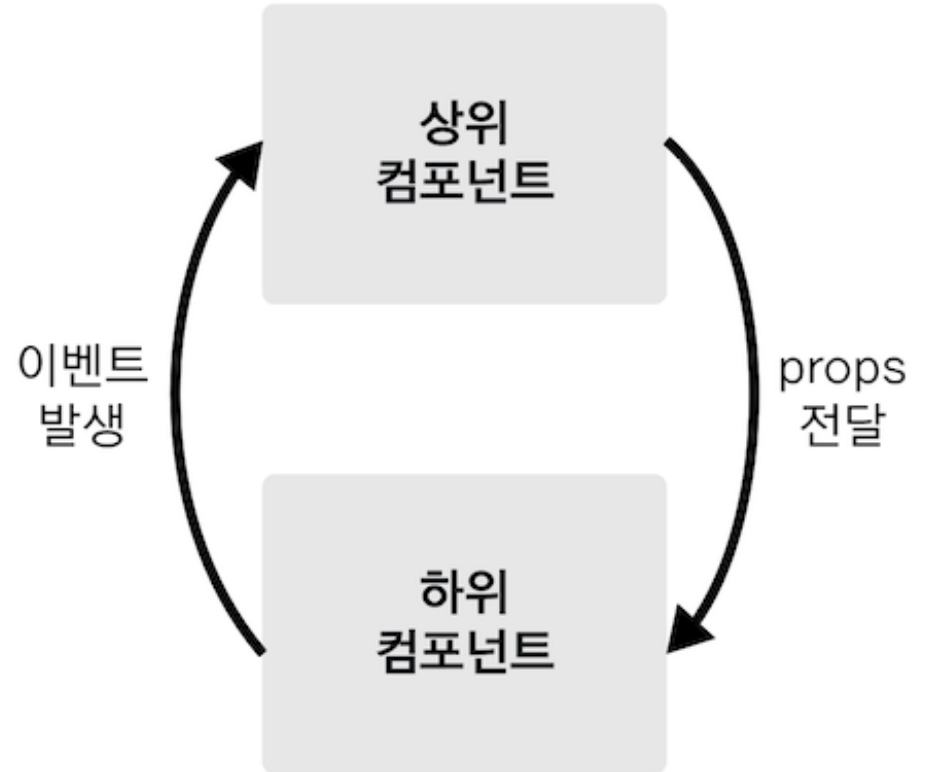
```
<table> <tr is="my-row"></tr> </table>
```

# template이 DOM으로 인식되지 않는 경우

- 다음과 같은 3가지 경우는 '문자열 template'을 사용할 경우에는 DOM으로 인식되지 않기 때문에 HTML 제약을 받지 않는다.
  - `<script type="text/x-template">`
  - JavaScript에서 `template` option으로 template을 직접 작성한 경우
  - `'.vue'` 단일 File Component
- 대체로 특별한 이유가 없다면 위에서 설명한 문자열 template을 이용하는 것이 좋다.
- 이렇게 문자열 template을 사용하면 HTML을 비교적 자유로운 형태로 작성할 수 있다.

# Component Communication

- Vue Component는 각각 고유한 Data 유효 범위를 갖는다.
- 따라서, Component간에 Data를 주고 받기 위해서는 다음과 같은 규칙을 따라야 한다.
  - 상위에서 하위로 Data를 내려줌 → **props** 속성
  - 하위에서 상위로 Event를 올려줌 → **Event** 발생  
→ **\$emit, \$on**
  - 형제간 → **EventBus**



# props 속성

- Component 사이에 Data를 전달할 수 있는 Component 통신 방법
- 상위 Component → 하위 Component로 보내는 Data 속성
- Code 형식
  - props 속성을 사용하기 위해서는 하위 Component의 Component 내용과 상위 Component의 Template에 각각 Code를 추가해줘야 한다.

//하위 Component 내용

```
var childComponent = {  
  props : ['props 속성 명']  
}
```

<!--상위 Component의 Template -->

```
<div id="app">
```

```
  <child-component v-bind:props 속성 명="상위 Component의 data 속성">
```

```
  </child-component>
```

```
</div>
```

## props 속성 (Cont.)

- 부모 Component에서 전달한 Data를 미리 정의 → **template**에서 속성 형태로 자식 Component에 전달.

```
{  
  props : {  
    부모로부터 전달받은 속성명 : {  
      type : String 혹은 Object type,  
      default : 기본값,  
      required : 필수 여부,  
      validator : 유효성 검사 함수  
    }  
  }  
}
```

## props 속성 (Cont.)

```
<div id="app">
  <child-component v-bind:propsdata="message"></child-component>
  <!-- 위의 출력 결과는 hello vue.js -->
</div>
<script>
  // 하위 Component : childComponent
  let childComponent = {
    props : ['propsdata'],
    template : '<p>{{propsdata}}</p>'
  }

  // 상위 Component : Root Component
  new Vue({
    el : '#app',
    components : {
      'child-component' : childComponent
    },
    data : {
      message : 'hello vue.js'
    }
  })
</script>
```



## props 속성 (Cont.)

이것은 pen 입니다.

```
<div id="app">
  <child-component v-bind:item-name="myItem"></child-component>
</div>
<script>
  Vue.component('child-component', {
    props : {
      itemName : {
        type : String
      }
    },
    template : '<p>이것은 {{ itemName }} 입니다.</p>'
  })

  new Vue({
    el : '#app',
    data : { myItem : 'pen' }
  })
</script>
```

A diagram with two red arrows. One arrow starts at the 'myItem' value in the 'v-bind:item-name' attribute of the 'child-component' tag and points to the 'itemName' property in the 'props' object of the 'child-component' definition. The second arrow starts at the 'myItem' value in the 'data' object of the main Vue instance and points to the 'myItem' property in the 'data' object of the same instance.

# props 속성 (Con

1. 배
2. 딸기
3. 포도

```
<div id="app">
  <ol>
    <fruits-item-name v-for="fruit in fruitsItems"
      :key="fruit.name" :fruits-item="fruit"></fruits-item-name>
    </ol>
  </div>
<script>
  Vue.component('fruits-item-name', {
    props : {
      fruitsItem : { //template 안에서는 kebab case 사용
        type : Object, //객체 여부
        required : true //이 Component에서는 반드시 필수
      },
    },
    template : '<li>{{fruitsItem.name}}</li>'
  })
  new Vue({
    el : '#app',
    data : { //부모 Component에서는 배열이지만 v-for를 사용해서 객체로 만들어 전달
      fruitsItems : [
        {name : '배'},
        {name : '딸기'},
        {name : '포도'}
      ]
    }
  })
})
```

# Event Emit

- Event 발생은 Component의 통신 방법 중 하위 Component → 상위 Component로 통신하는 방법이다.
- Code 형식
  - 하위 Component의 Method나 Lifecycle Hook과 같은 곳에 다음과 같은 Code를 추가한다.

//하위 Component의 내용

**this.\$emit('Event 이름')**

- 그리고 나서 해당 Event를 수신하기 위해 상위 Component의 Template에 다음과 같이 구현한다.

<!-- 상위 Component의 Template -->

**<div id="app">**

**<child-component v-on:Event 이름="상위 Component의 실행할 Method 이름 또는 연산">**

**</child-component>**

**</div>**

# Event Emit (Cont.)

Event 호출하기

127.0.0.1:5500 says

Event Received

```
<div id="app">
  <child-component v-on:update="showAlert"></child-component>
</div>
<script>
  let childComponent = {
    template : '<button @click="sendEvent">Event 호출하기</button>',
    methods : {
      sendEvent : function(){
        this.$emit('update');
      }
    }
  }
  new Vue({
    el : '#app',
    components : {
      'child-component' : childComponent
    },
    methods : {
      showAlert : function(){
        alert('Event Received');
      }
    }
  })
</script>
```



The diagram illustrates the event emission process. A red arrow originates from the `sendEvent` method within the `childComponent` object, pointing to the `showAlert` method in the parent component's `methods` object. Another red arrow points from the `showAlert` method to the `v-on:update="showAlert"` attribute on the `child-component` tag in the HTML, indicating the event listener.

# Lab : Component간 통신

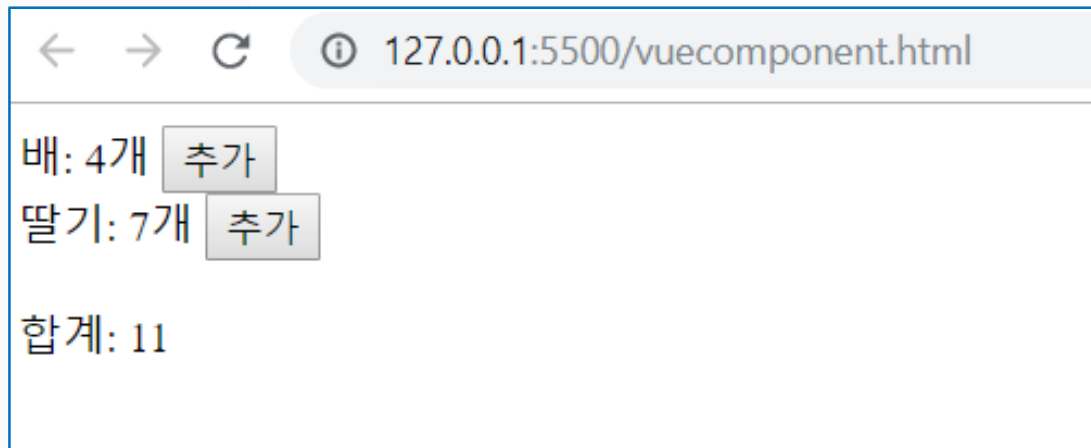
```
8     <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
9 </head>
10 <body>
11     <div id="fruits-counter">
12         <div v-for="fruit in fruits">
13             {{fruit.name}}: <counter-button v-on:increment="incrementCartStatus()"></counter-button>
14         </div>
15         <p>합계: {{total}}</p>
16     </div>
```

## Lab : Component간 통신 (Cont.)

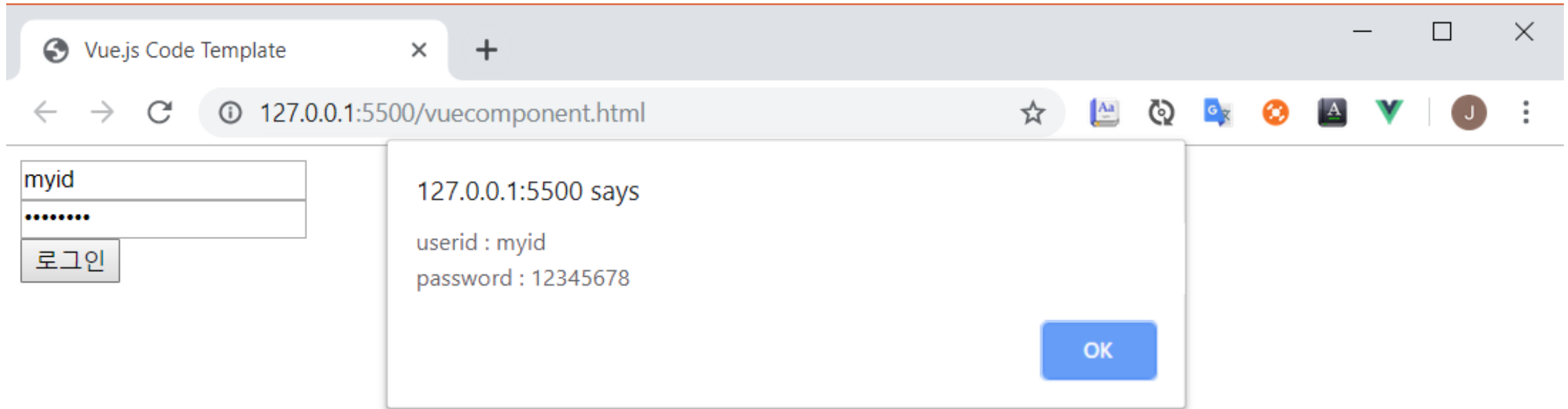
```
18     <script>
19         var counterButton = Vue.extend({
20             template: '<span>{{counter}}개 <button v-on:click="addToCart">추가</button></span>',
21             data: function () {
22                 return {
23                     counter: 0
24                 }
25             },
26             methods: {
27                 addToCart: function () {
28                     this.counter += 1
29                     this.$emit('increment') // increment 커스텀 이벤트 발생
30                 }
31             },
32         })
```

## Lab : Component간 통신 (Cont.)

```
34     new Vue({
35       el: '#fruits-counter',
36       components: {
37         'counter-button': counterButton
38       },
39       data: {
40         total: 0,
41         fruits: [
42           { name: '배' },
43           { name: '딸기' }
44         ]
45       },
46       methods: {
47         incrementCartStatus: function () {
48           this.total += 1
49         }
50       }
51     })
52 </script>
```



# Lab : Login Form Component





# Lab (Cont.)

```
8     <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
9 </head>
10 <body>
11     <div id="login-example">
12         <user-login></user-login>
13     </div>
14
15     <!-- template -->
16     <script type="text/x-template" id="login-template">
17         <div id="login-template">
18             <div>
19                 <input type="text" placeholder="로그인 ID" v-model="userid">
20             </div>
21             <div>
22                 <input type="password" placeholder="패스워드" v-model="password">
23             </div>
24             <button @click="login">로그인</button>
25         </div>
26     </script>
```

## Lab (Cont.)

```
28 <script>
29   Vue.component('user-login', {
30     template : '#login-template',
31     data : function(){
32       return {
33         userid:'',
34         password : ''
35       }
36     },
37     methods : {
38       login:function(){
39         auth.login(this.userid, this.password)
40       }
41     }
42   })
43
44   let auth = {
45     login:function(id, pass){
46       alert('userid : ' + id + '\n' + 'password : ' + pass);
47     }
48   }
49
50   new Vue({
51     el : '#login-example'
52   });
53 </script>
```



---



# Lab : Component간 Data 전달하기

# EventBus를 이용한 형제간 Component 통신

- 상위-하위 Component가 아닌 Component끼리 Data를 전달할 경우, **this**를 사용한 **props**와 사용자 정의 **Event(\$emit)**로 통신할 수 없다.
- 이러한 경우에는 Vue Instance의 **EventBus**를 사용한다.
- **EventBus**는 Publisher-Subscriber Pattern이라고 볼 수 있다.
- 한 쪽 Component가 Event를 발행(Publishing)하면 다른 Event가 그 Event를 구독(Subscribing)하는 구조이다.

# EventBus를 이용한 형제간 Component 통신 (Cont.)

- EventBus 전용 Instance 생성 – EventBus의 초기화

```
let bus = new Vue()
```

- Component A의 **methods**에서

```
bus.$emit('event-name') //Event를 Publishing하는 쪽에서
```

- Component B의 **created** hook에서

```
bus.$on('event-name', function() { //Event를 Subscribing하는 쪽에서
```

```
  // bus-event Event가 발생할 때 처리
```

```
  // 익명 함수 내부의 this는 bus Instance.
```

```
}
```

# EventBus를 이용한 형제간 Component 통신 (Cont.)

```
<div id="app">
  <publisher></publisher>
  <subscriber></subscriber>
</div>
```


```
let bus = new Vue({
  data : {
    count : 0
  }
})
```

Count Up


bus : 11

```
Vue.component('publisher', {
  template : '<p><button @click="add">Count Up</button></p>',
  methods : {
    add(){
      bus.$emit('add')
    }
  }
})

Vue.component('subscriber', {
  template : '<p>bus : {{ bus.count }} </p>',
  computed: {
    bus: function () {
      return bus.$data
    }
  },
  created : function(){
    bus.$on('add', function(){
      this.count++
    })
  }
})
```



# Lab : EventBus를 이용한 Component간 Data 전달하기



# Slot

- Slot은 Vue.js에서 제공하는 강력한 기능 중 하나
- Slot을 사용해서 Component를 재사용할 수 있게 만든다.
- Slot은 상위가 되는 Component 쪽에서 하위가 되는 Component의 Template 일부를 주입하는 기능이다.
- Vue.js에서 props를 이용해서 상위 Component에서 하위 Component로 Data를 전달할 수 있지만, 복잡한 Code가 포함된 Data를 전달하는 것은 쉽지 않다.
- 이 경우 Slot을 이용하면 쉽게 전달할 수 있다.
- Component에 Contents를 넣거나, Template의 일부를 원하는 형태로 변경하고 싶을 때 사용한다.
- Unnamed Slot vs Named Slot



# Slot (Cont.)

## ■ Unnamed Slot

- Default Slot
- 가장 기본적인 Slot으로 상위 Component의 일부를 하위 Component에 주입할 수 있다.
- 상위의 **p**, **div**, **img** 같은 DOM Element들을 하위 Component에 정의된 **<slot></slot>** tag 위치로 대체할 수 있다.
- 그러나, 하위 Component에 **<slot>**이 없으면 상위에서 HTML 을 전달해도 하위로 전달되지 않는다.

```
<div>
  <child-component>
    <p>여행이란?</p>
  </child-component>
</div>
```

상위 Component

```
<div>
  <slot></slot>
</div>
```

하위 Component

# Slot (Cont.)

Child Component
Vue.js의 강력한 기능인 Slot을 배워보자.
Vue.js의 강력한 기능인 Slot을 배워보자.

```
<body>
  <div id="app">
    <child-component>
      <p>Vue.js의 강력한 기능인 slot을 배워보자.</p>
    </child-component>
  </div>
  <script>
    Vue.component('child-component', {
      template : `
        <div>
          <p>Child Component</p>
          <slot></slot>
          <slot></slot>
        </div>`
    })
    new Vue({
      el : '#app'
    })
  </script>
</body>
```

# Slot (Cont.)

## ■ Named Slot

- 일반적으로 Vue Component는 단일 Slot만 사용해도 충분하지만, 종종 여러 개의 Slot을 배치해야 할 수도 있다.
- Slot의 기본 사용 방법은 slot element를 작성하는 것인데 Slot이 다수 배치되면 Vue Component가 어떤 Slot을 참조해야 하는지 알 수 없기 때문에 Slot에 이름을 부여한다.
- Slot에 이름이 지정되면 상위 Instance Template의 Vue Component tag 아래에 위치하는 Element가 특정 Slot의 위치에 추가하도록 제어할 수 있다.

`<slot name='slot_name'></slot>`

# Lab : Slot 다루기



# Slot (Cont.)

여기에 페이지 제목이 위치한다.

Main Contents 위치.

하나 더 있다.

여기에는 연락처 정보가 위치한다.

```
11 <div id="app">
12   <app-layout>
13     <h1 slot="header">여기에 페이지 제목이 위치한다.</h1>
14     <p>Main Contents 위치.</p>
15     <p>하나 더 있다.</p>
16     <p slot="footer">여기에는 연락처 정보가 위치한다.</p>
17   </app-layout>
18 </div>
19 <script>
20   Vue.component('app-layout',{
21     template : `
22 <div class='container'>
23   <header>
24     <slot name='header'></slot>
25   </header>
26   <main>
27     <slot></slot>
28   </main>
29   <footer>
30     <slot name='footer'></slot>
31   </footer>
32 </div>`
33   })
34   new Vue({
35     el : '#app'
36   })
```