

# Vue.js Fundamentals

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

# Vue Code Template

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Vue.js Code Template</title>
8      <script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>
9  </head>
10 <body>
11
12 </body>
13 </html>
```

# Vue Instance

- Vue Application은 Vue 함수를 사용하여 새로운 Vue Instance를 만드는 것부터 시작한다.
- Vue로 화면을 개발하기 위해 필수적으로 생성해야 하는 기본 단위이다.
- Vue 객체 생성은 **new** 연산자를 사용한다.

```
new Vue({  
  ...  
});
```

- 이처럼 생성한 Vue 객체는 Vue.js 프로그램의 최상위 객체로 동작한다.
- 이 객체를 일반적으로 Vue Instance라고 또는 *Root Component*라고 부른다.

## Vue Instance (Cont.)

### ■ Bootstrapping

- Vue.js는 Vue Template과 Instance의 속성을 Binding하고 DOM Event를 Vue Method로 전달하도록 하는 등 Instance를 생성할 때 필요한 작업

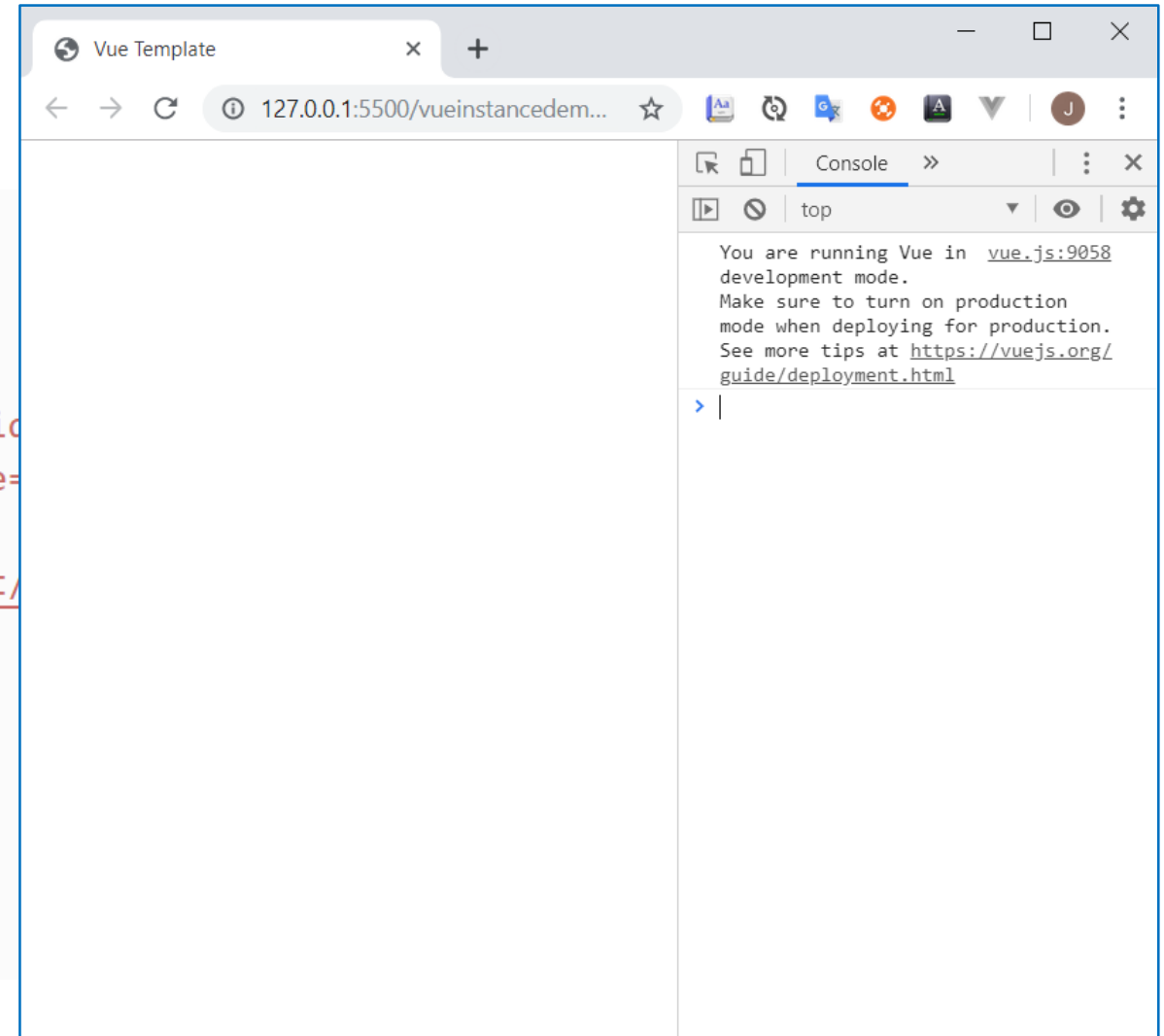
```
var vm = new Vue({  
    //Option 이나 Component 설정 등.  
});
```

- **Vue** **vm**의 변수 이름은 자유롭게 지정할 수 있지만 관례로 **vm**으로 지정한다.

# Vue Instance (Cont.)

## ■ vueinstancedemo.html

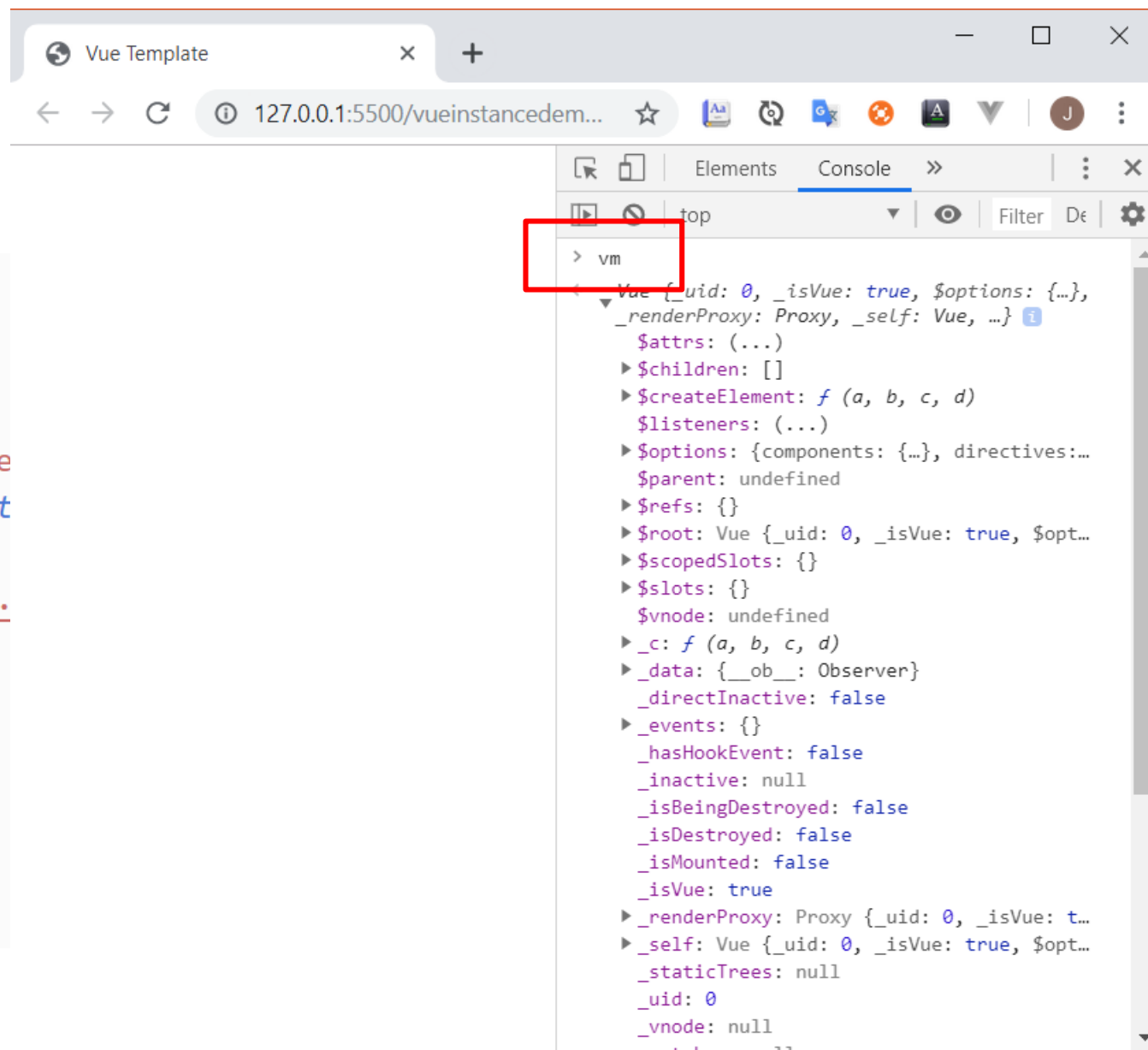
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-wid
6   <meta http-equiv="X-UA-Compatible" content="ie=
7   <title>Vue.js Code Template</title>
8   <script src="https://unpkg.com/vue@2.6.10/dist/
9 </head>
10 <body>
11   <script>
12     var vm = new Vue();
13   </script>
14 </body>
15 </html>
```



# Vue Instance (Cont.)

## ■ vueinstancedemo.html

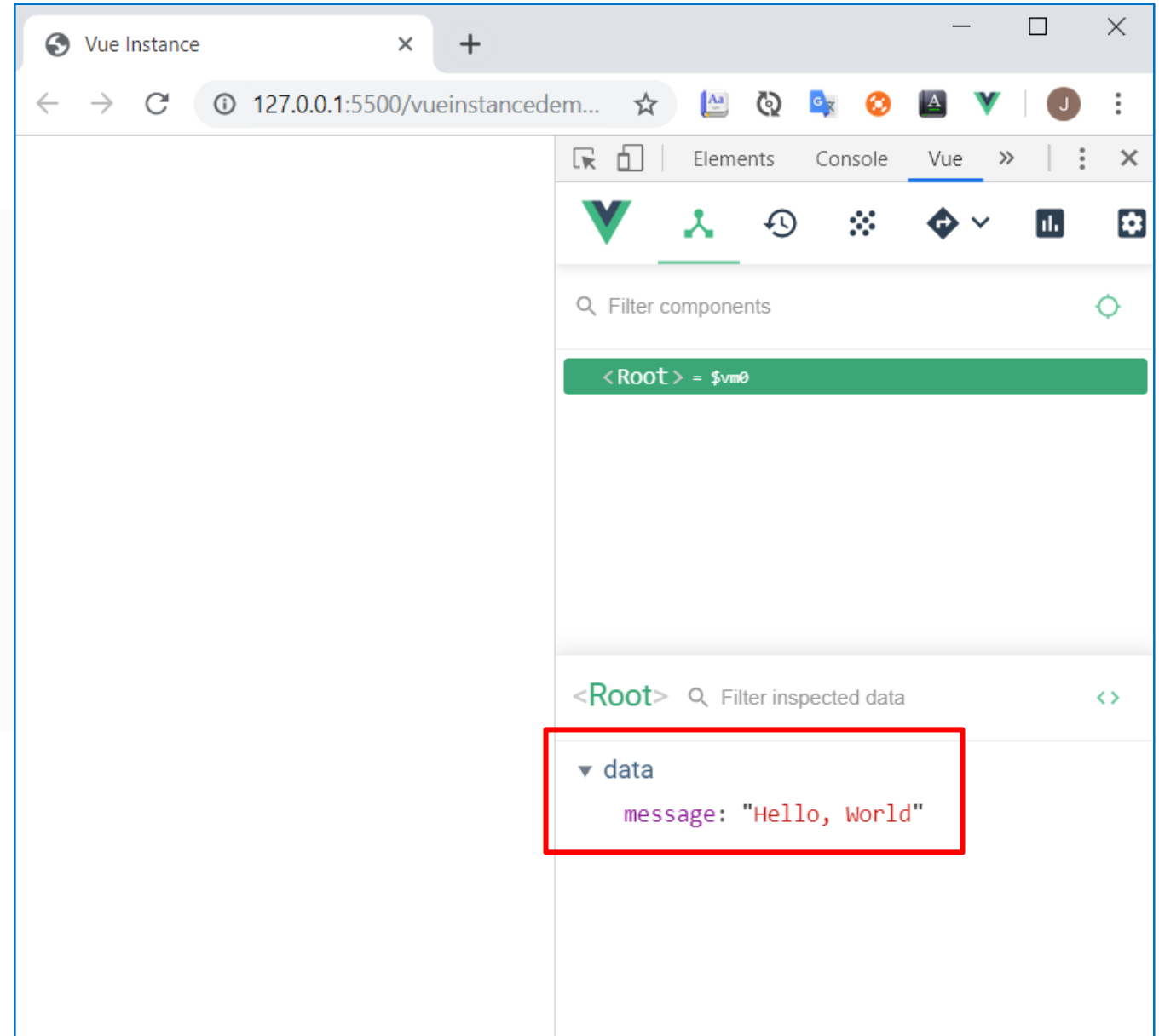
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <title>Vue.js Code Template</title>
8   <script src="https://unpkg.com/vue@2.6.12"></script>
9 </head>
10 <body>
11   <script>
12     var vm = new Vue();
13   </script>
14 </body>
15 </html>
```



# Vue Instance (Cont.)

## ■ vueinstancedemo.html

```
<div id="app"></div>
<script>
  var vm = new Vue({
    el : '#app',
    data : {
      message : 'Hello, World'
    }
  });
</script>
```



## Vue Instance (Cont.)

- Instance에서 사용할 수 있는 속성과 API
  - **el** : Instance가 그려지는 화면의 시작점(특정 HTML Tag)
  - **template** : 화면에 표시할 요소(HTML, CSS 등)
  - **data** : Vue의 반응성(Reactivity)이 반영된 Data 속성
  - **methods** : 화면의 동작과 Event Logic을 제어하는 Method
  - **created** : Vue의 Lifecycle과 관련된 속성
  - **watch** : data에서 정의한 속성이 변화했을 때 추가 동작을 수행할 수 있게 정의하는 속성

```
<script>
  var vm = new Vue({
    el : ,
    template : ,
    data : ,
    methods: ,
    created : ,
    watch : ,
  });
</script>
```

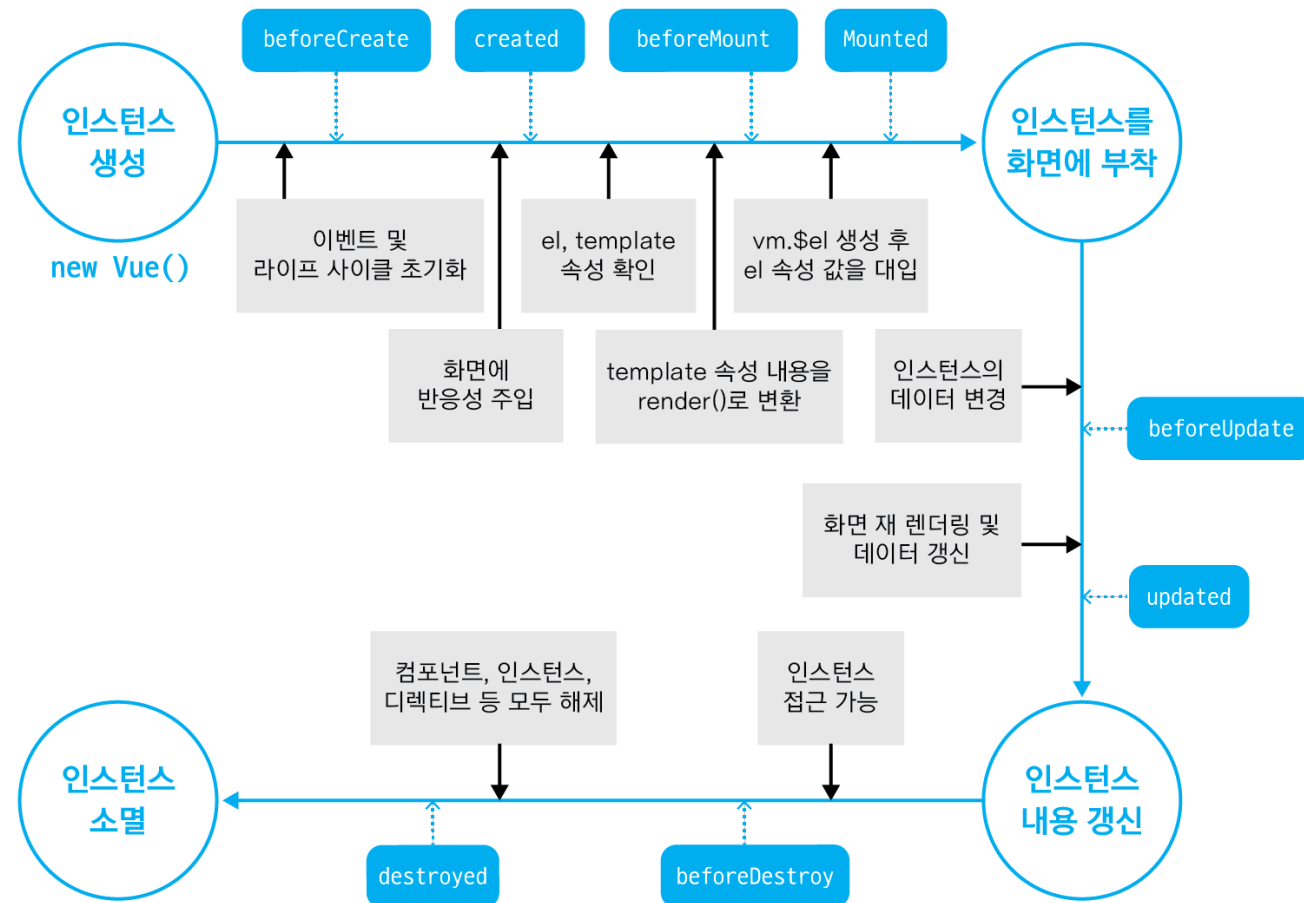


# Instance Lifecycle

- Vue의 Instance가 생성되어 소멸되기까지 거치는 과정
- Instance가 생성되고 나면 Library 내부적으로 다음과 같은 과정이 진행된다.
  - **data** 속성의 초기화 및 관찰(Reactivity 주입)
  - Vue Template Code Compile(Virtual DOM → DOM 변환)
  - Instance를 DOM에 부착

# Instance Lifecycle (Cont.)

## ■ Instance의 Lifecycle Diagram

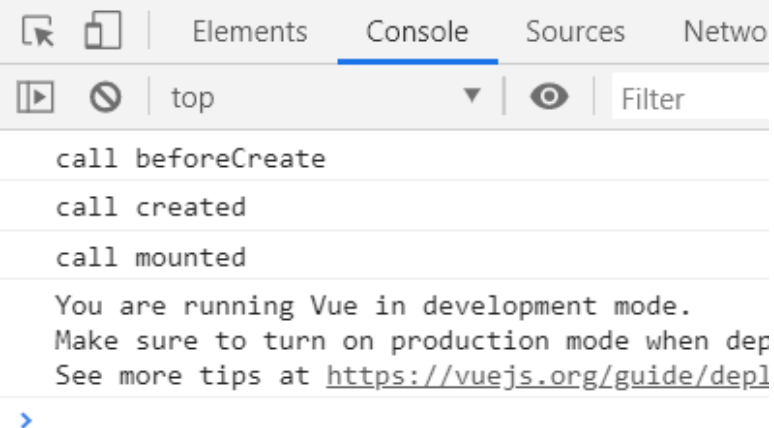


# Instance Lifecycle (Cont.)

```
<div id="app">
  {{ message }}
</div>

<script>
  var vm = new Vue({
    el : '#app',
    data : {
      message : 'Hello, Vue.js!!!'
    },
    beforeCreate : function() {
      console.log('call beforeCreate');
    },
    created : function(){
      console.log('call created');
    },
    mounted : function(){
      console.log('call mounted');
    },
    updated : function(){
      console.log('call updated');
    }
  });
</script>
```

Hello, Vue.js!!!



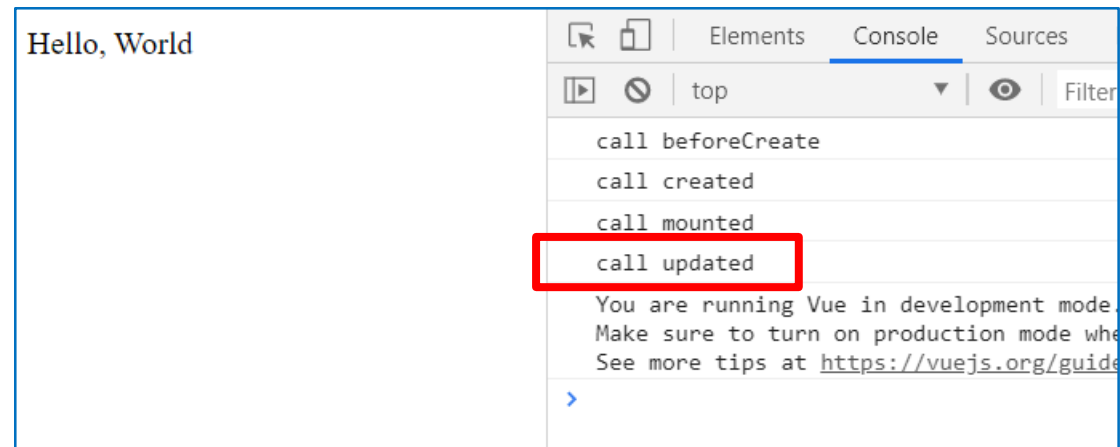
The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following logs:

- call beforeCreate
- call created
- call mounted

Below the logs, a message states: "You are running Vue in development mode. Make sure to turn on production mode when deploying to production. See more tips at <https://vuejs.org/guide/deployment>".

# Instance Lifecycle (Cont.)

```
var vm = new Vue({  
  el : '#app',  
  data : {  
    message : 'Hello, Vue.js!!!'  
  },  
  beforeCreate : function() {  
    console.log('call beforeCreate');  
  },  
  created : function(){  
    console.log('call created');  
  },  
  mounted : function(){  
    console.log('call mounted');  
    this.message = 'Hello, World';  
  },  
  updated : function(){  
    console.log('call updated');  
  }  
});
```



message값 변경

# Instance Lifecycle (Cont.)

## ■ Lifecycle Hook

- Vue의 Lifecycle을 이해해야 하는 이유는 바로 Lifecycle Hook 때문이다.
- Lifecycle Hook으로 Instance의 특정 시점에 원하는 Logic을 구현할 수 있다.
- 예를 들어, Component가 생성되고 바로 Data를 Server에서 받아오고 싶으면 **created**나 **beforeMount** Lifecycle Hook을 사용할 수 있다.
- 옆 Code는 Instance가 생성되고 바로 Axios로 HTTP GET Request를 보내서 Data를 받아오는 Code.

```
new Vue({  
  methods: {  
    fetchData() {  
      axios.get(url);  
    }  
  },  
  created: function() {  
    this.fetchData();  
  }  
})
```

# Instance Lifecycle (Cont.)

## ■ 자주 사용되는 Lifecycle Hook List

### ● created

#### ■ beforeCreate 다음 단계

- data 속성과 methods 속성이 정의되었기 때문에 **this.data** 또는 **this.fetchData()**와 같은 Logic들을 이용하여 **data** 속성과 **methods** 속성에 정의된 값에 접근하여 Logic을 실행할 수 있다.
- 하지만, 아직 Instance가 화면 요소에 부착되기 전이기 때문에 **template** 속성에 정의된 dom 요소에 접근할 수 없다.

### ● beforeMount

- **created** 단계 이후 **template** 속성에 지정한 Markup 속성을 **render()** 함수로 변환한 후 **el** 속성에 지정한 화면 요소(DOM)에 Instance를 부착하기 전에 호출되는 단계.
- **render()**가 호출되기 직전의 Logic 추가할 때 좋다.

# Instance Lifecycle (Cont.)

## ■ 자주 사용되는 Lifecycle Hook List

### ● Mounted

- **el** 속성에 지정한 화면 요소에 Instance가 부착된 후 호출되는 단계.
- **template** 속성에 정의한 화면 요소(DOM)에 접근할 수 있어서 화면 요소를 제어하는 Logic을 수행하기 좋은 단계.
- 하지만, DOM에 Instance가 부착되고 바로 호출되기 때문에 하위 Component나 외부 Library에 의해 추가된 화면 요소들이 최종 HTML Code로 변환되는 시점과 다를 수 있다.

### ● Destroyed

- Vue Instance가 소멸되고 난 후 호출되는 단계.
- Vue Instance에 정의한 모든 속성이 제거되고 하위에 선언한 Instance들 또한 모두 소멸한다.

# Vue Template

- Vue로 화면을 조작하는 방법
- Data Binding
- Directive



# Vue Template (Cont.)

## ■ Data Binding

- Vue Instance에서 정의한 속성들을 화면에 표시하는 방법
- 가장 기본적인 Data Binding은 콧수염 괄호(Mustache Tag)이다.

```
<div> {{ message }} </div>
new Vue( {
  data : {
    message : 'Hello Vue.js'
  }
});
```

- `div` Tag에 콧수염 괄호를 이용해서 Vue Instance의 `message` 속성을 연결했다.

# Vue Template (Cont.)

## ■ Data Binding

```
<div id="app">
  <h1> {{ message }} </h1>
  <input v-model="message">
</div>

<script>
  var vm = new Vue({
    el : '#app',
    data : {
      message : 'Greeting You!'
    }
  });
</script>
```

Greeting You!

Greeting You!

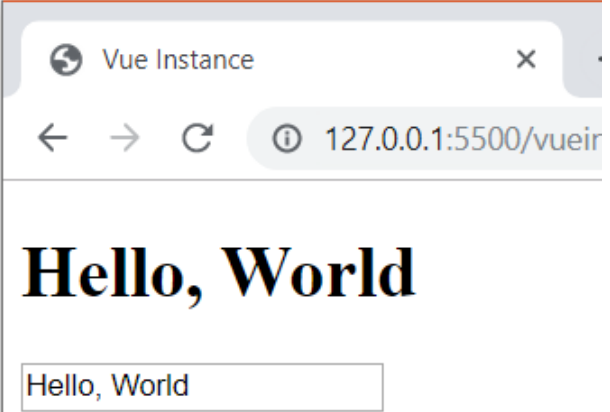
Vue.js가 좋네요.

Vue.js가 좋네요.

# Vue Template (Cont.)

## ■ jQuery와 비교

```
<script src="https://code.jquery.com/jquery-3.4.1.js"
  integrity="sha256-WpOohJOqMqqyKL9FccASB900KwACQJpFTUUBLTYOVvVU="
  crossorigin="anonymous"></script>
</head>
<body>
  <div id="app">
    <h1>Greeting You! </h1>
    <input id="message">
  </div>
  <script>
    $('#message').on('keyup', function(){
      var message = $('#message').val();
      $('#h1').text(message);
    });
  </script>
</body>
```



# Lab

- Hello, {{ name }}
- input tag 추가하고 이를 name과 Binding 하기
- User가 이름을 입력하거나 이름을 변경할 때마다 즉시 변경되어야 한다.

Hello Mr. Anderson

Enter your name:

```
{  
  "name": "Mr. Anderson"  
}
```

# Vue Template (Cont.)

## ■ Directive

- Vue로 화면의 요소를 더 쉽게 조작하기 위한 문법이다.
- 화면 조작에서 자주 사용되는 방식들을 모아 Directive 형태로 제공하고 있다.
- 예를 들어 다음과 같이 특정 속성 값에 따라 화면의 영역을 표시하거나 표시하지 않을 수 있다.

```
<div>  
    Hello <span v-if="show">Vue.js</span>  
</div>  
new Vue({  
    data : {  
        show : false  
    }  
})
```

# Vue Template (Cont.)

## ■ Directive

```
<ul>
  <li v-for="item in items"> {{ item }} </li>
</ul>
new Vue({
  data : {
    items : ['shirts', 'jeans', 'hats']
  }
})
```

# Vue Template (Cont.)

## ■ Directive

- **v-for** Directive를 활용하면 Data 속성의 개수만큼 화면의 요소를 반복하여 출력할 수 있다.
- 목록을 표시해야 할 때 유용하게 사용할 수 있는 기능이다.
- **v-bind**
- **v-on**
- **v-model**
- **v-else**

# Lab.

- User는 이름과 함께 성별을 입력한다.
- User가 남성(male)일 경우 Hello Mister {{ name }}
- User가 여성(female)일 경우 Hello Miss {{ name }}
- 성별이 남성이나 여성이 아닌 경우 경고 메시지를 'So you can't decide. Fine!'이라고 출력한다.

Hello, Mister Anderson.

Enter your gender:

male

Enter your name:

Anderson

```
{  
  "gender": "male",  
  "name": "Anderson"  
}
```



# Lab.

- 개인 정보를 저장해서 출력해보자.
- 개인정보에는 이름(name), 몸무게(weight), 키(height), 좋아하는 음식(favoriteFood)가 있다.
- v-for를 이용해서 index : key = value 형식으로 출력해보자.

This is me!

0: name = Kostas

1: height = 1.73m

2: weigh = 65kg

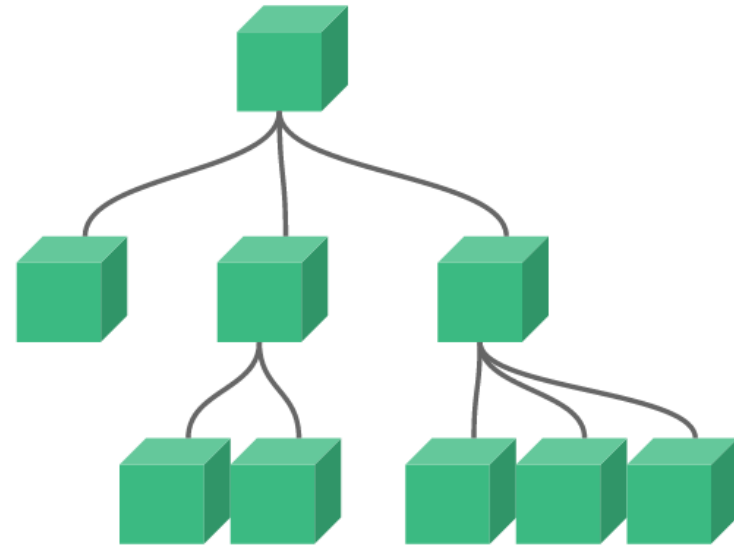
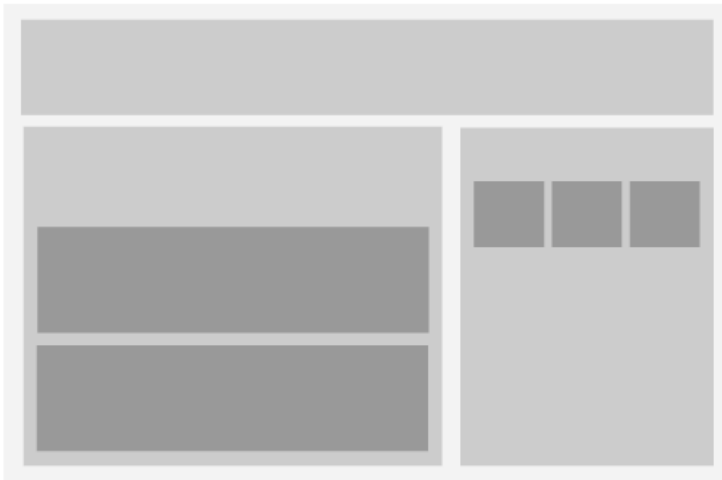
3: eyeColor = brown

4: favoriteFood = Ntolmas

```
{
  "quotes": {
    "name": "Kostas",
    "height": "1.73m",
    "weigh": "65kg",
    "eyeColor": "brown",
    "favoriteFood": "Ntolmas"
  }
}
```

# Vue Component

- 화면의 영역을 구분하여 개발할 수 있는 Vue의 기능
- Component 기반으로 화면을 개발하게 되면 Code의 재사용성이 올라가고 빠르게 제작할 수 있다.



## Vue Component (Cont.)

- Component 생성 코드 형식
  - Component를 생성하는 Code 형식은 다음과 같다.

```
Vue.component('Component 이름', {  
  // Component 내용  
});
```

## Vue Component (Cont.)

### ■ Component 생성 후 표시하기

- 간단한 App Header Component를 생성해 보자.

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

- 이제 등록된 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.)

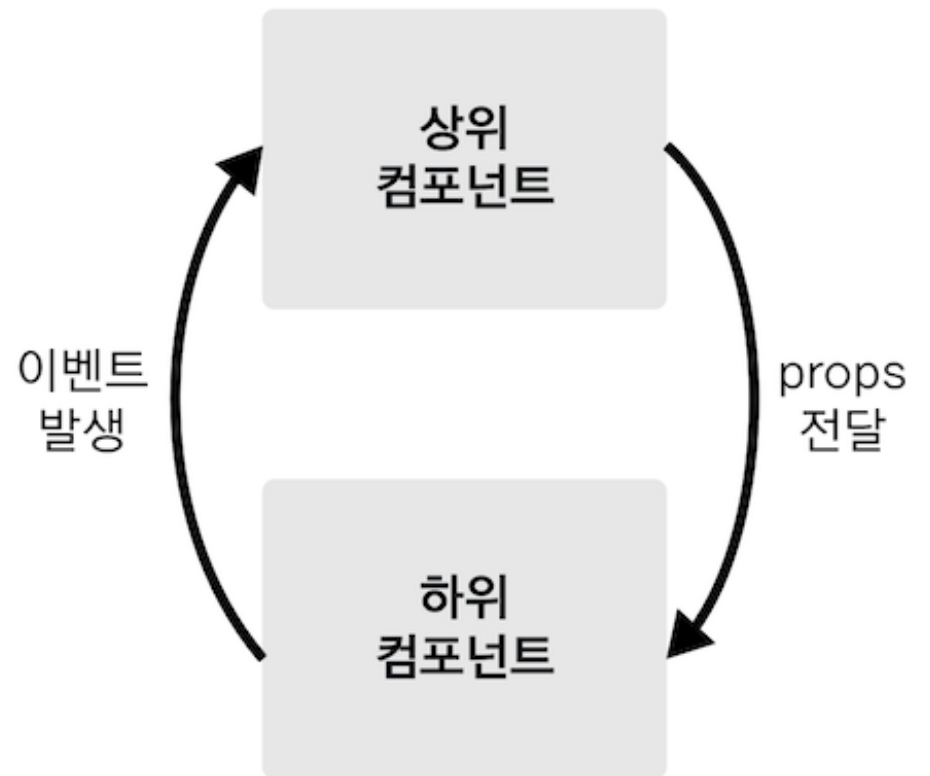
- Component 등록 방법 2가지
  - 전역 Component 등록하는 방법
    - 앞 Slide 방법
  - 지역 Component 등록하는 방법

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

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

# 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



## props 속성

- Component 사이에 Data를 전달할 수 있는 Component 통신 방법
- 상위 Component에서 하위 Component로 내려보내는 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.)

```
js
// 하위 컴포넌트 : childComponent
var childComponent = {
  props: ['propsdata'],
  template: '<p>{{ propsdata }}</p>'
}

// 상위 컴포넌트 : root 컴포넌트
new Vue({
  el: '#app',
  components: {
    'child-component': childComponent
  },
  data: {
    message: 'hello vue.js'
  }
})
```

```
html
<div id="app">
  <child-component v-bind:propsdata="message"></child-component>
  <!-- 위의 출력 결과는 hello vue.js -->
</div>
```



# Event Emit

- Event 발생은 Component의 통신 방법 중 하위 Component에서 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.)

```
// 하위 컴포넌트 : childComponent
var childComponent = {
  methods: {
    sendEvent: function() {
      this.$emit('update');
    }
  }
}

// 상위 컴포넌트 : root 컴포넌트
new Vue({
  el: '#app',
  components: {
    'child-component': childComponent
  },
  methods: {
    showAlert: function() {
      alert('event received');
    }
  }
})
```

js

```
<div id="app">
  <child-component v-on:update="showAlert"></child-component>
</div>
```

html

# Vue Router

- Vue Library를 이용하여 Single Page Application(SPA)를 구현할 때 사용하는 Library
- Vue Router 설치
  - CDN 방식

```
<script src=https://unpkg.com/vue-router/dist/vue-router.js>
```
  - NPM 방식

```
npm install vue-router
```

## Vue Router (Cont.)

### ■ Vue Router 등록

- Vue Router 설치 후 다음 Code와 같이 Router Instance를 하나 생성하고 Vue Instance에 등록한다.

```
//Router Instance 생성
var router = new VueRouter({
    //Router Option
})
//Instance에 Router Instance를 등록
new Vue({
    router : router
})
```

# Vue Router (Cont.)

## ■ Vue Router Option

- Vue Router 등록 후 Router에 Option을 정의한다.
- 대부분의 SPA Application에서는 다음과 같이 2개 Option을 필수로 지정한다.
  - routes : Routing할 URL과 Component 값 지정
  - mode : URL의 Hash 값 제거 속성

```
new VueRouter({  
  mode: 'history',  
  routes: [  
    { path: '/login', component: LoginComponent },  
    { path: '/home', component: HomeComponent }  
  ]  
})
```

- 위의 Code는 Routing 할 때 URL에 # 값을 제거하고, URL 값이 /login 과 /home 일 때 각각 LoginComponent와 HomeComponent를 뿌려준다.

# Vue Router (Cont.)

## ■ router-view

- Browser의 주소 창에서 URL이 변경되면, 앞에서 정의한 routes 속성에 따라 해당 Component가 화면에 뿌려진다.
- 이때 뿌려지는 지점이 Template의 <router-view> 이다.

```
<div id="app">  
  <router-view> </router-view>  
  <!-- LoginComponent 또는 HomeComponent -->  
</div>
```

- 앞 Slide에서 정의한 Routing Option 기준으로 /login은 LoginComponent를 /home은 HomeComponent를 화면에 표시한다.

# Vue Router (Cont.)

## ■ router-link

- 일반적으로 Web Page에서 Page 이동을 할 때는 사용자가 URL을 다 입력해서 이동하지 않는다.
- 이 때 화면에서 특정 Link를 Click해서 Page를 이동할 수 있도록 하는 Tag이다.

```
<router-link to="이동할 URL"> </router-link>
```

```
<div>
```

```
  <router-link to="/login"> </router-link>
```

```
</div>
```

# Axios

- Vue에서 권고하는 HTTP 통신 Library.
- Promise 기반의 HTTP 통신 Library
- 상대적으로 다른 HTTP 통신 Library에 비해 문서화가 잘 되어 있고 API가 다양하다.
- 설치
  - CDN 방식

```
<script src=https://unpkg.com/axios/dist/axios.min.js> </script>
```
  - NPM 방식

```
npm install axios
```



# Axios (Cont.)

## ■ 사용방법

- Library를 설치하고 나면 axios라는 변수에 접근할 수 있다.
- axios 변수를 이용하여 아래와 같이 HTTP GET 요청하는 Code를 작성한다.

```
<div id="app">
  <button v-on:click="fetchData">get data</button>
</div>
```

html

```
new Vue({
  el: '#app',
  methods: {
    fetchData: function() {
      axios.get('https://jsonplaceholder.typicode.com/users/')
        .then(function(response) {
          console.log(response);
        })
        .catch(function(error) {
          console.log(error);
        });
    }
  }
})
```

js