Vue.js - Lightweight Powerful View-layer Library

개요

- PWA 기반 Single Page App 을 제작하기 위한 화면 앞단 라이브러리 Vue.js 소개
- Vue 로 화면을 구성하기 위한 기본적인 개념을 학습하고 샘플을 활용하여 코드 제작
- Vue 의 주요 구성요소 (Component, Router, Resource, Templates) 학습 및 실습

목차

- Vue 소개
- MVVM 패턴이란?
- Vue 시작하기
- 실습 - Hello Vue
- Vue Instance
    - Instance Lifecycle

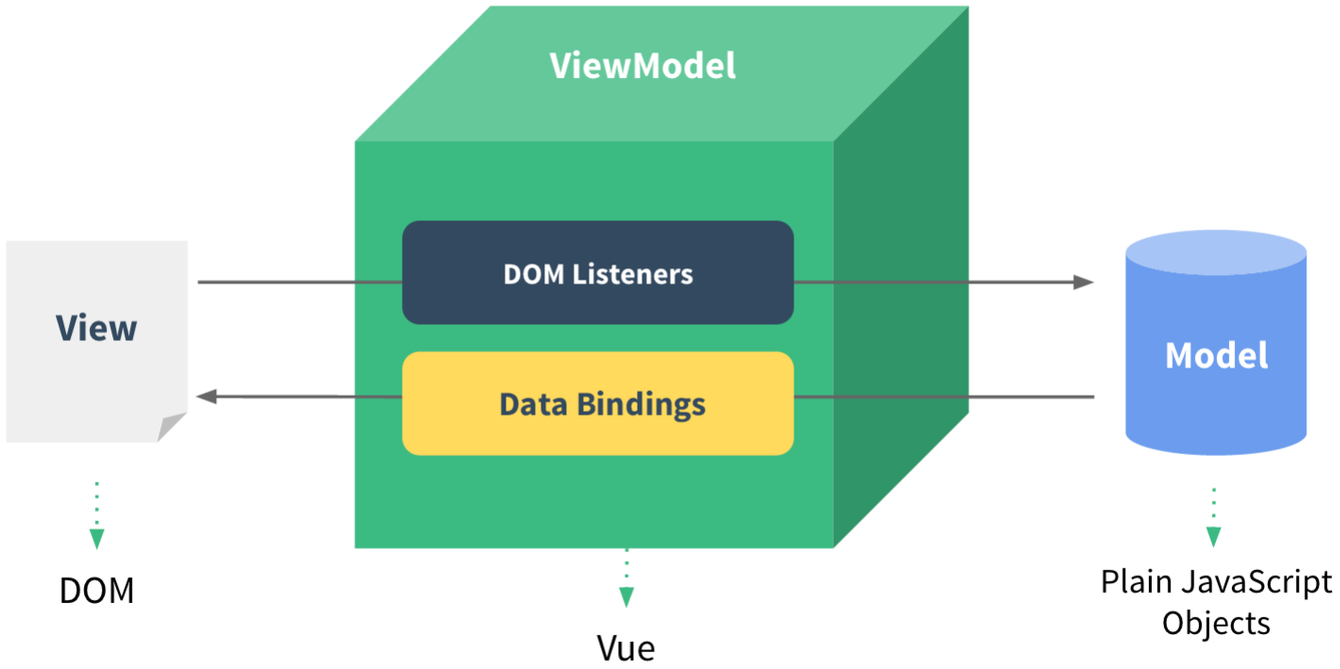
- Vue Components
  - Global & Local Component
  - 실습 #1 - Components
  - Props
  - 실습 #2 - Props
  - Event Bus
- Vue Routers
  - 실습 #4 - Routers
  - Nested Routers
  - 실습 #5 - Nested Routers
  - Named Views

- Vue Resource
- Vue Templates
    - Attritubes & JS Expression & Directives & Filter
    - 실습 #0 - Templates
    - Data Bindings
    - 실습 #3 - Components & Props & For
- Vue Single File Components
- Vue Development Workflow
    - 실습 #6 - Single File Component

- Vue Loader
- Glossory : Virtual DOM
- 실습 #8 - Todo App 제작

Vue 는 무엇인가?

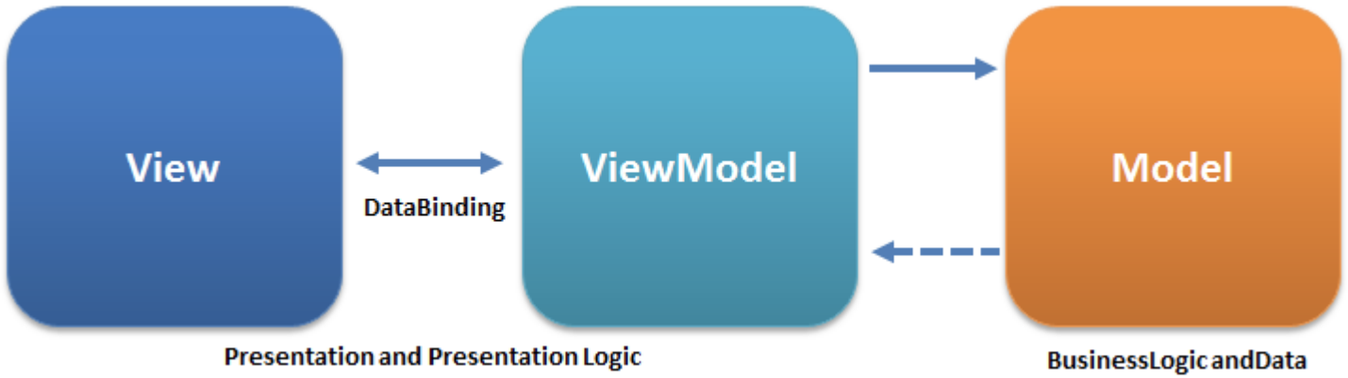
MVVM 패턴의 뷰모델(ViewModel) 레이어에 해당하는 화면(View)단 라이브러리



- **데이터 바인딩** 과 **화면 단위를 컴포넌트 형태로 제공하며**, **관련** API **를 지원**하는데에 궁극적인 목적이 있음
- Angular 에서 지원하는 2 way data bindings 을 동일하게 제공
- 하지만 Component **간 통신**의 기본 골격은 React 의 1 Way Data Flow (**부모** -> **자식**) 와 유사
- Virtual DOM **을 이용한 렌더링** 방식은 React와 비슷
- 다른 Front-End FW (Angular, React) 와 비교했을 때 상대적으로 가볍고 빠름.
- 간단한 Vue 를 적용하는데 있어서도 **러닝커브가 낮고**, **쉽게 접근 가능**

MVVM 패턴이란?

위키 에 명시된 것처럼, Backend 로직과 Client 의 마크업 & 데이터 표현단을 분리하기 위한 구조로 전통적인 MVC 패턴의 방식에서 기인하였다. 간단하게 생각해서 화면 앞단의 화면 동작 관련 로직과 뒷단의 DB 데이터 처리 및 서버 로직을 분리하고, 뒷단에서 넘어온 데이터를 Model 에 담아 View 로 넘어주는 중간 지점이라고 보면 되겠다.



Vue **시작하기**

다른 프런트엔드 프레임워크 Angular, React에 비해 **바로 시작하기가 정말 쉽다**는 점이 가장 큰 장점

```
<html>
  <head>
    <title>Vue Sample</title>
  </head>
  <body>
    <div id="app">
      {{ message }}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
    <script>
      new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue.js!'
        }
      })
    </script>
  </body>
</html>
```

실습 - Hello Vue 시작하기

앞의 시작하기 코드로 간단하게 Vue 를 이용한 Hello World 를 제작해보자.

Vue Instance

인스턴스는 Vue.js로 개발할 때 필요한 근간 & 토대(Foundation)

Vue.js 라이브러리를 로딩하고 나서 접근 가능한 Vue 생성자로 인스턴스를 생성

```
var vm = new Vue({
  // ...
})
```

위를 풀어서 얘기하면 `Vue` **라는 변수에 화면에서 사용할 옵션 (데이터, 속성, 메서드, 등등) 을 포함하여 화면의 단위를 생성한다** 는 의미

Vue Instance **생성자**

Vue 생성자를 만드는 방법은 아래와 같다.

```
// vm 은 ViewModel 을 뜻한다. (관행적인 코딩 컨벤션)
var vm = new Vue({
  // 인스턴스 옵션 속성
})
```

- Vue 객체를 생성할 때 아래와 같이 *data, template, el, methods, life cycle callback* 등의 인스턴스 옵션을 포함할 수 있다.

```
var vm = new Vue({
  template: ...,
  el: ...,
  methods: {

  },
  created: {

  }
  // ...
})
```

Vue Instance **라이프싸이클 초기화**

Vue 인스턴스가 생성될 때 `Vue({ })` 아래의 초기화 작업을 수행한다.

- 데이터 관찰 : 데이터의 변경을 감지하는 반응성(Vue Reactivity) 주입
- 템플릿 컴파일 : 가상돔(Virtual DOM) -> 실제 돔(DOM)으로 변환
- DOM 에 인스턴스 부착 : 템플릿의 내용을 실제 DOM에 연결
- 데이터 변경시 DOM 업데이트

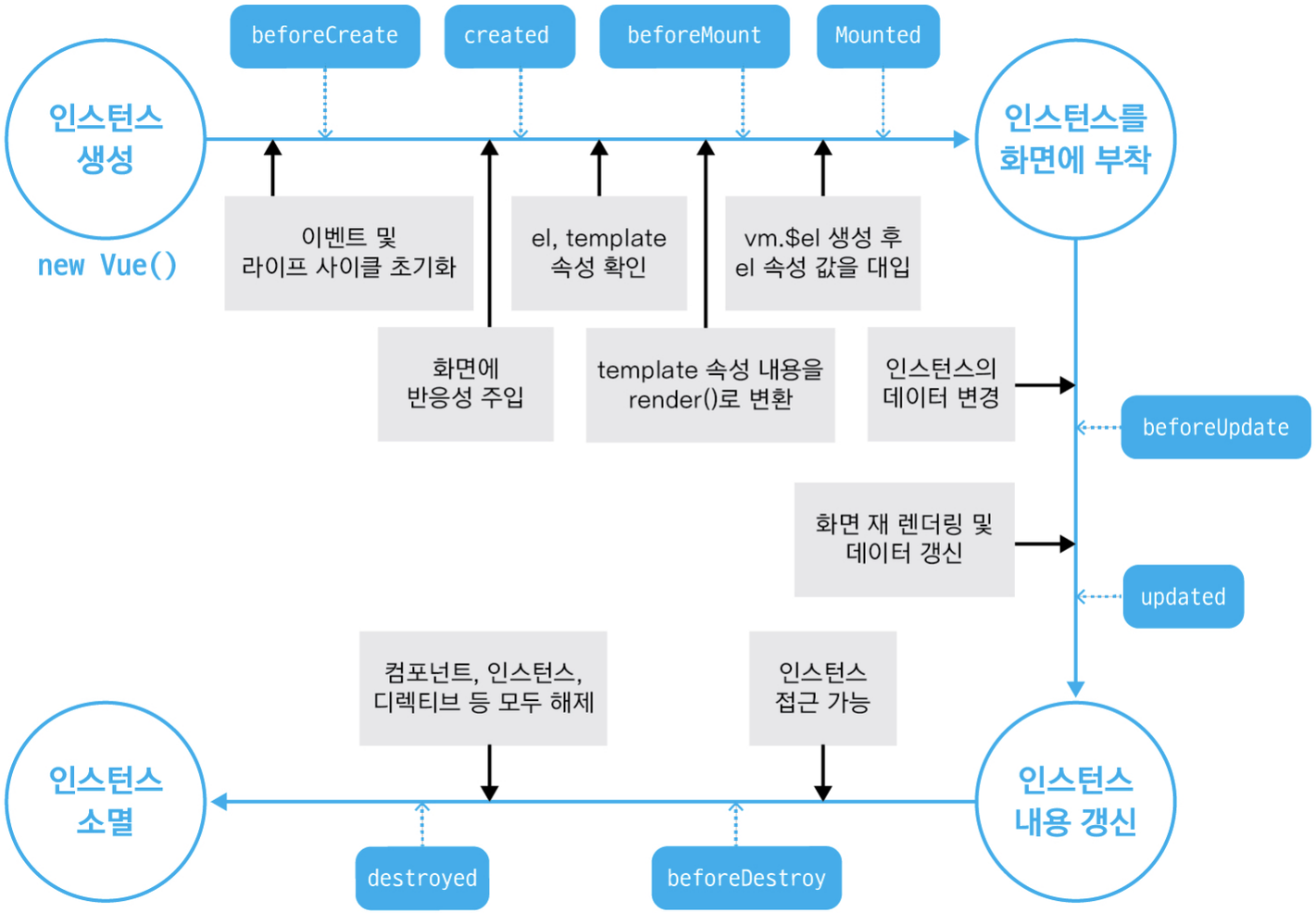
이 초기화 작업 외에도 개발자가 의도하는 커스텀 로직을 아래와 같이 추가할 수 있다.

```
var vm = new Vue({
  data: {
    a: 1
  },
  created: function () {
    // this는 vm을 가리킴
    console.log('a is: ' + this.a)
  }
})
```

이 외에도 라이프싸이클 단계에 따라 아래 메서드를 사용할 수 있다.

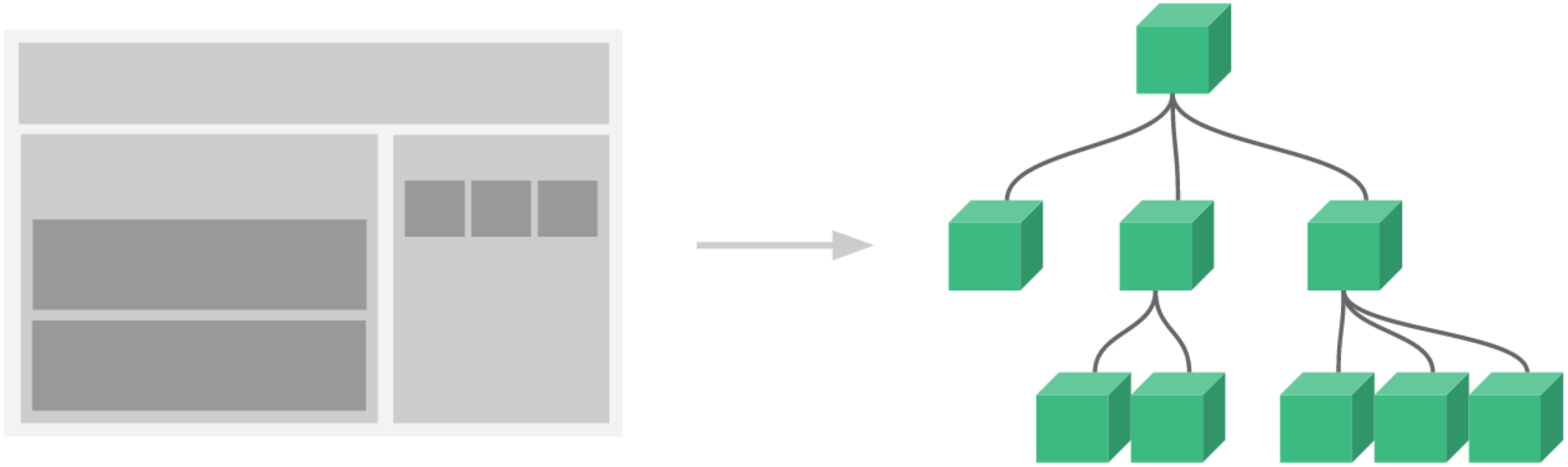
- `mounted`
- `updated`
- `destroyed`

위와 같이 초기화 메서드로 커스텀 로직을 수행하기 때문에 Vue.js**에는** Controller**가 없다**.


beforeCreate
created
beforeMount
Mounted
인스턴스
생성
new Vue()
인스턴스를
화면에 부착
이벤트 및
라이프 사이클 초기화
el, template
속성 확인
vm.$el 생성 후
el 속성 값을 대입
화면에
반응성 주입
template 속성 내용을
render()로 변환
인스턴스의
데이터 변경
beforeUpdate
화면 재 렌더링 및
데이터 갱신
updated
컴포넌트, 인스턴스,
디렉티브 등 모두 해제
인스턴스
접근 가능
인스턴스
소멸
인스턴스
내용 갱신
destroyed
beforeDestroy

Vue Components

- 화면에 비춰지는 뷰의 단위를 쪼개어 재활용이 가능한 형태로 관리하는 것이 컴포넌트



- 컴포넌트 등록은 아래와 같은 코드로 생성 가능하다.

```
<div id="app">
  <my-component></my-component>
</div>
```

```
// 등록
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
// Vue 인스턴스 생성
new Vue({
  el: '#app'
})
```

주의할 점 : Vue **인스턴스를 생성하기전에 꼭** Component **부터 등록**!

- 컴포넌트의 `data` 속성은 꼭 함수로 작성해야한다.

```
// 아래 컴포넌트는 콘솔에서 오류 발생
Vue.component('my-component', {
  data: {
    message: 'hello'
  }
})

var data = { text: 'hello' };
Vue.component('my-component', {
  data: function () {
    return data;
  }
  // 모든 컴포넌트가 같은 값을 공유하지 않게 아래와 같이 수정
  // data: function () {
  //   return {
  //     text: 'hello'
  //   }
  // }
})
```

Global or Local Component

- 컴포넌트를 뷰 인스턴스에 등록해서 사용할 때 다음과 같이 전역(global)으로 등록할 수 있다.

```
Vue.component('my-component', {
  // ...
})
```

- 지역(local)으로 등록하는 방법은 다음과 같다.

```
var cmp = {
  data: function () {
    return {
      // ...
    };
  }
  template: '<hr>',
  methods: {}
}

// 아래 Vue 인스턴스에서만 활용할 수 있는 로컬(지역) 컴포넌트 등록
new Vue({
  components: {
    'my-cmp' : cmp
  }
})
```

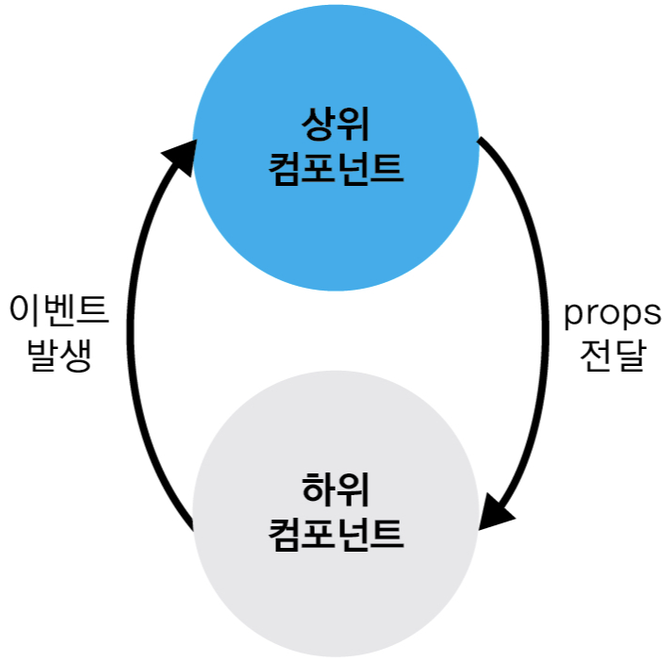
실습 #1 - 컴포넌트 등록

방금 배운 Global 컴포넌트 & Local 컴포넌트를 실제로 등록해보자

지역 & 컴포넌트 등록 예제

부모와 자식 컴포넌트 관계

- 구조상 상-하 관계에 있는 컴포넌트의 통신은
  - 부모 -> 자식 : props down
  - 자식 -> 부모 : events up



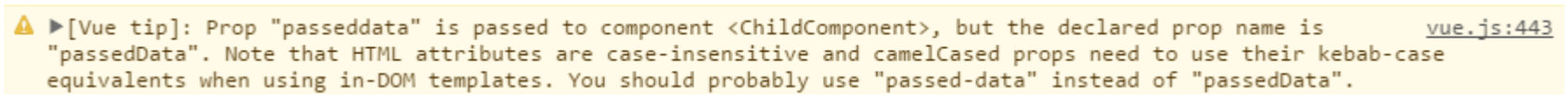
Props

- 모든 컴포넌트는 각 컴포넌트 고유의 유효범위를 갖는다.
    - ex) 특정 컴포넌트에서 다른 컴포넌트의 값을 바로 참조할 수 없음
- **상위에서 하위로 값을 전달하려면** props **속성을 사용한다**.

```
Vue.component('child-component', {
  props: ['propsdata'],
  template: '<p>{{ propsdata }}</p>'
});

var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue! from Parent Component',
  }
});
```

```
<div id="app">
  <child-component v-bind:propsdata="message"></child-component>
</div>
```

주의 할점 : js 에서 props 변수 명명을 카멜 기법으로 하면 html 에서 접근은 케밥 기법(`-`) 으로 가야한다.



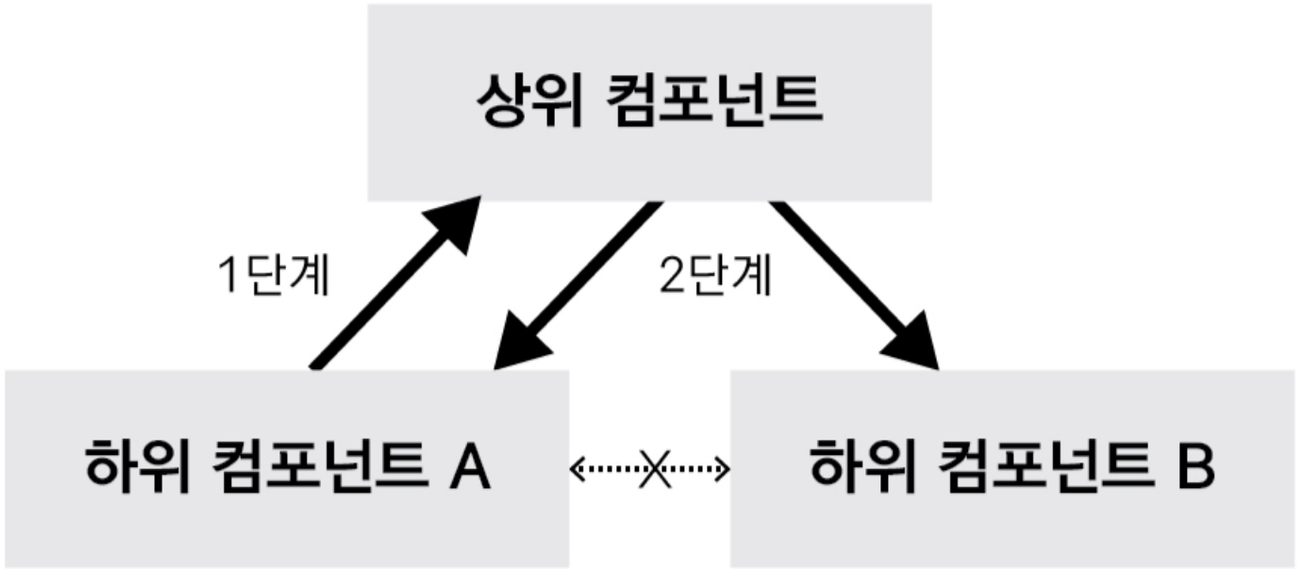
실습 #2 - Props

props 전달하기 예제

같은 레벨의 컴포넌트 간 통신

동일한 상위 컴포넌트를 가진 2개의 하위 컴포넌트 간의 통신은

- 하위 -> 상위 -> 다시 하위 2개



컴포넌트 간의 직접적인 통신은 불가능하도록 되어 있는게 Vue.js**의 기본 구조**

Event Bus - **컴포넌트 간 통신**

Non Parent - Child 컴포넌트 간의 통신을 위해 Event Bus 를 활용할 수 있다.

- Event Bus 를 위해 새로운 Vue 를 생성하여 아래와 같이 Vue Root Instance 가 위치한 파일에 등록

```
// Vue Root Instance 전에 꼭 등록 순서가 중요.
var eventBus = new Vue();

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

이벤트 발생(ES5)

- `$emit('이벤트 명', 인자)` 으로 이벤트 발생

```
methods: {
  showLog: function() {
    eventBus.$emit('refresh', 10);
  }
}
```

이벤트 수신(ES5)

- 해당 이벤트를 받을 컴포넌트에서 `$on('이벤트 명', 인자)` 로 이벤트 수신

```
created() {
  eventBus.$on('refresh', function (data) {
    console.log(data); // 10
  });
}
```

이벤트 발생(ES6)

- 이벤트를 발생시킬 컴포넌트에 `eventBus` import 후 `$emit` 으로 이벤트 발생

```
import { eventBus } from '../../main';

eventBus.$emit('refresh', 10);
```

이벤트 수신(ES6)

- 해당 이벤트를 받을 컴포넌트에도 동일하게 import 후 콜백으로 이벤트 수신

```
import { eventBus } from '../../main';

// 등록 위치는 해당 컴포넌트의 created 메서드에 등록
created() {
  eventBus.$on('refresh', function (data) {
    console.log(data); // 10
  });
}
```

- 참고 : eventBus 의 콜백함수 안에서 해당 소스의 메서드를 참고하려면 `self` 사용

```
methods: {
  callAnyMethod() {
    // ...
  }
}
created() {
  var self = this;
  eventBus.$on('refresh', function (data) {
    console.log(this); // this 는 빈 Vue 인스턴스를 접근
    self.callAnyMethod() // self 는 이 created 의 Vue 컴포넌트에 접근, 따라서 이 컴포넌트에
  });
}
```

Vue Routers

- Vue.js를 이용하여 싱글 페이지 애플리케이션(SPA)을 제작할 때 유용한 라우팅 라이브러리
- 뷰 코어 라이브러리 외에 Router 라이브러리를 공식 지원하고 있고 CDN 또는 NPM 으로 설치

```
// NPM 설치 방법
npm install vue-router --save
```

- Vue 라우터는 기본적으로 RootUrl'/#/'{Router name} 의 구조로 되어 있다.

```
example.com/#/user
```

- 여기서 URL의 `#` 태그 값을 제거하려면 history mode 사용

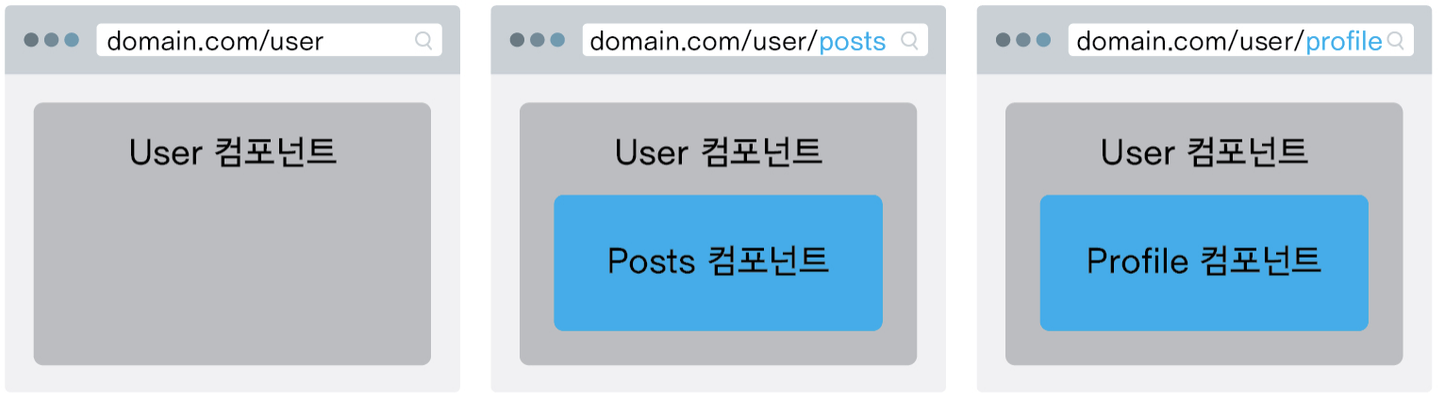
```
const router = new VueRouter({
  routes,
  // 아래와 같이 history 모드를 추가해주면 된다.
  mode: 'history'
})
```

실습 #4 - Vue 기본 Router

뷰 기본 라우터 동작 확인

Nested Routers

- 라우터로 화면 이동시 Nested Routers 를 이용하여 여러개의 컴포넌트를 동시에 렌더링 할 수 있다.
- 렌더링 되는 컴포넌트의 구조는 가장 큰 상위의 컴포넌트가 하위의 컴포넌트를 포함하는 `Parent - Child` 형태와 같다.



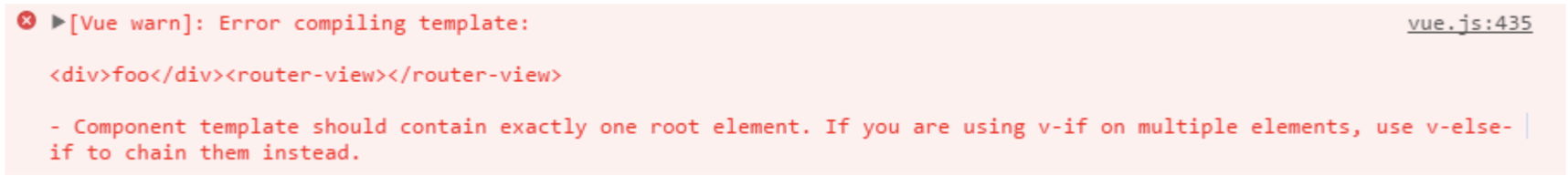
```
<!-- localhost:5000 -->
<div id="app">
  <router-view></router-view>
</div>

<!-- localhost:5000/home -->
<!-- parent component -->
<div>
  <p>Main Component rendered</p>
  <!-- child compoonent -->
  <app-header></app-header>
</div>
```

```
// localhost:5000/home 에 접근하면 Main 과 Header 컴포넌트 둘다 렌더링 된다.
{
  path : '/home',
  component: Main,
  children: [
    {
      path: '/',
      component: AppHeader
    },
    {
      path: '/list',
      component: List
    },
  ]
}
```

뷰 템플릿 작성시 주의 사항

- Vue 의 Template 에는 최상위 태그가 1개만 있어야 렌더가 가능하다.

```
var Foo = {
  template: `
    <div>foo</div>
    <router-view></router-view>
  ` // 에러 발생
};
```



- 여러 개의 태그를 최상위 태그 레벨에 동시에 위치시킬 수 없음
- 따라서 아래와 같이 최상위 Element 는 한개만 지정해야 한다.

```
var Foo = {
  // div 태그 안에 텍스트와 `router-view` 포함하여 정상 동작
  template: `
    <div>foo
      <router-view></router-view>
    </div>
  `
};
```

ES6의 Template String

실습 #5 - Nested Routers

네스티드 라우터 구현하기

Named Views

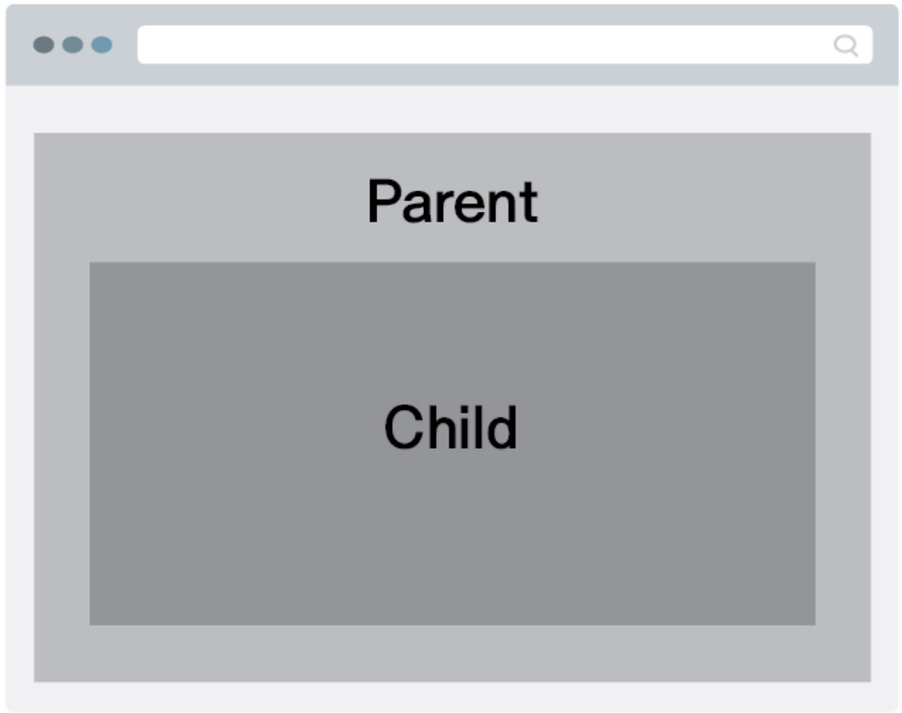
- 특정 URL로 이동했을 때 여러 개의 View(컴포넌트)를 동시에 렌더링 하는 라우팅 방법
- 각 컴포넌트에 해당하는 `name` 속성과 `router-view` 지정 필요

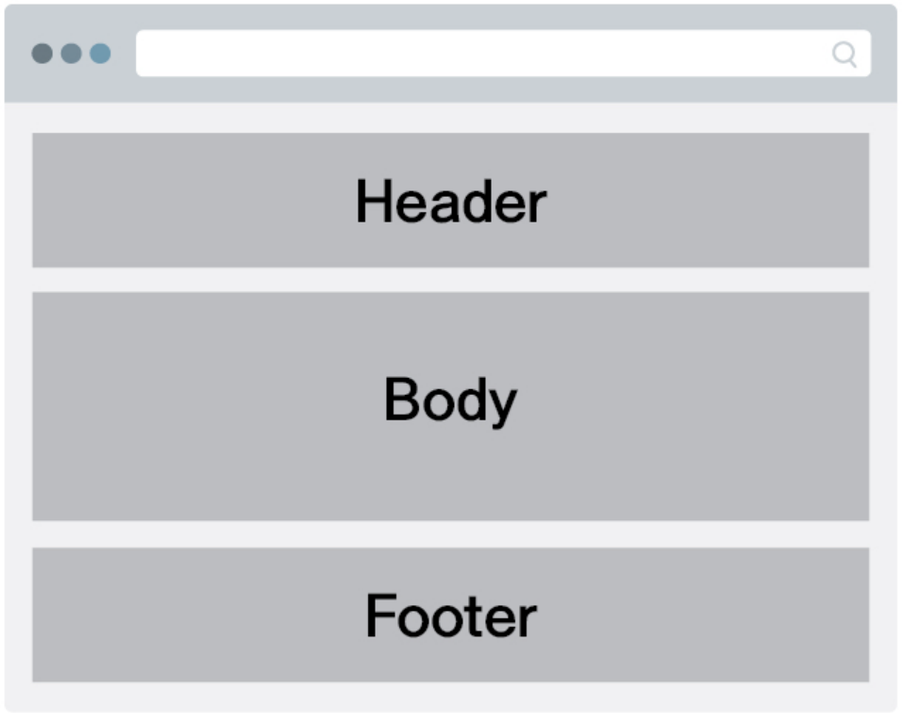
```
<div id="app">
  <router-view name="nestedHeader"></router-view>
  <router-view></router-view>
</div>
```

```
{
  path : '/home',
  // Named Router
  components: {
    nestedHeader: AppHeader,
    default: Body
  }
},
```

Nested View vs Named Views?

- 특정 URL 에서 1개의 컴포넌트에 여러 개의 하위 컴포넌트를 갖는 것을 Nested Routes
- 특정 URL 에서 여러 개의 컴포넌트를 쪼개진 뷰 단위로 렌더링 하는 것을 Named View





Vue Resource

뷰에서 HTTP 통신(ajax)을 하기 위해 지원하는 라이브러리

```
npm install vue-resource --save
```

위 명령어로 설치 후 Root Vue Instance 를 선언하는 js 파일에 아래와 같이 등록

```
// 뒤에서 배울 Single File Component 구조 기준
import VueResource from 'vue-resource';
...
Vue.use(VueResource);
```

사용법은 아래와 같다.

```
this.$http.get(url).then(successCallback, failCallback);
```

Vue Templates

뷰로 그리는 화면의 요소들, 함수, 데이터 속성을 모두 Templates 안에 포함된다.

- 뷰는 DOM 의 요소와 뷰 인스턴스를 매핑할 수 있는 HTML Template 을 사용.
- 템플릿이 실제 돔에 달라 붙을 때 가상 돔(Virtual DOM)을 실제 돔으로 변환
- 가상돔을 사용하는 이유 : DOM 조작을 최소화 하고 렌더링을 꼭 다시 해야만 하는 요소를 계산하여 성능 부하를 최소화.
- 원하면 render function 을 직접 구현하여 사용할 수 있음

- Attributes : HTML Attriubtes 를 Vue 의 변수와 연결할 때는 `v-bind` 를 이용.

```
<div v-bind:id="dynamicId"></div>
```

- JS Expressions : `{{ }}` 안에 다음과 같이 javascript 표현식도 가능하다.

```
<div>{{ number + 1 }}</div> <!-- O -->
<div>{{ message.split('').reverse().join('') }}</div> <!-- O -->
<div>{{ if (ok) { return message } }}</div> <!-- X -->
```

- Directives : `v-` 접두사를 붙인 뷰의 특별한 속성. `:` 을 붙여 인자를 받아 취급할 수 있다.

```
<p v-if="seen">Now you see me</p>
<!-- : 뒤에 선언한 href 인자를 받아 url 값이랑 매핑 -->
<a v-bind:href="url"></a>
<!-- click 이라는 이벤트를 받아 Vue 에 넘겨준다. -->
<a v-on:click="doSomething">
```

- Filters : 화면에 표시되는 텍스트의 형식을 편하게 바꿀 수 있도록 고안된 기능이며, `|` 을 이용하여 여러 개의 필터를 적용할 수 있다.

```
<!-- message 에 표시될 문자에 capitalize 필터를 적용하여 첫 글자를 대문자로 변경한다. -->
{{ message | capitalize }}
```

```
new Vue({
  // ...
  filters: {
    capitalize: function (value) {
      if (!value) return ''
      value = value.toString()
      return value.charAt(0).toUpperCase() + value.slice(1)
    }
  }
})
```

실습 #0 - Vue Templates

뷰 디렉티브 다뤄보기

Data Binding

- Template에 뷰 데이터를 바인딩 하는 방법은 크게 3가지가 있다.
  - Interpolation (값 대입)
  - Binding Expressions (값 연결)
  - Directives (디렉티브 사용)

Interpolation - **값 대입**

- 뷰의 가장 기본적인 데이터 바인딩 체계는 콧수염 괄호 `{{ }}` 를 따른다.

```
<span>Message: {{ msg }}</span>
<span>This will never change: {{* msg }}</span>
<div id="{{ id }}"></div>
```

Binding Expressions - **값 연결**

- `{{ }}` 를 이용한 데이터 바인딩을 할 때 자바스크립트 표현식을 사용할 수 있다.

```
<div>{{ number + 1 }}</div> <!-- O -->
<div>{{ message.split('').reverse().join('') }}</div> <!-- O -->
<div>{{ if (ok) { return message } }}</div> <!-- X -->
```

- 뷰 Filter도 콧수염 괄호를 사용하여 구현한다. 여러 개 체인 가능

```
{{ message | capitalize }}
{{ message | capitalize | upcapitalize}}
```

Directives

- 뷰에서 제공하는 특별한 속성이며 `-v` 의 prefix(접두사)를 갖는다.
- *자바스크립트 표현식, filter* 모두 적용된다.

```
<!-- login 의 결과에 따라 p 가 존재 또는 미존재 -->
<p v-if="login">Hello!</p>
<!-- 링크를 클릭하면 doSomething 메서드 실행 -->
<a v-on:click="doSomething">
```

Class Binding

- 화면 요소에 CSS 스타일링을 할 때도 v-bind 디렉티브를 사용한다.
- 기존 HTML class 속성과 v-bind:class를 함께 사용하여 동적으로 class를 할당할 수 있다.

```
<div class="static" v-bind:class="{ 'class-a': isA, 'class-b': isB }"></div>
<script>
  data: { isA: true, isB: false }
</script>
```

```
<!-- 결과 -->
<div class="static class-a"></div>
```

- 아래와 같이 Array 구문도 사용할 수 있다.

```
<div v-bind:class="[classA, classB]">
<script>
  data: {
    classA: 'class-a',
    classB: 'class-b'
  }
</script>
```

Single File Components with JSX(ES6)

- 앱의 복잡도가 증가할 때, `.vue` 라는 파일 단위 안에 html, js, css 를 관리할 수 있는 방법
- 복잡도가 커짐에 따라 야기될 수 있는 문제들
    i. **모든 컴포넌트에 고유의 이름**을 붙여야 함
    ii. js 파일에서 template 안의 html 의 **문법 강조가 되지 않음**
    iii. js 파일상에서 css **스타일링 작업이 거의 불가**
    iv. ES5 를 이용하여 계속 앱을 작성할 경우 Babel **빌드가 지원되지 않음**

- `.vue` 파일을 브라우저가 렌더할 수 있는 파일들로 변환하려면 webpack 의 vue-loader 또는 browserify 이용

```
<template>
  <!-- ... -->
</template>

<script>
  // ...
</script>

<style>
  /*...*/
</style>
```

참고 : ES5 **만 사용하는 경우** single file component **의 혜택을 볼 수 없음**

Vue Development Workflow

vue cli 로 간단한 webpack 설정이 되어 있는 프로젝트 생성이 가능하다.

```
npm install -global vue-cli
vue init webpack-simple
npm install
npm run dev
```

```
export default {
  // 이 안의 내용은 모두 Vue Instance 에 포함되어 생성된다.
}
```

실습 #6 - Single File Component

싱글 파일 컴포넌트 체계에서 컴포넌트 등록

- Local 컴포넌트 1개 등록
- Global 컴포넌트 1개 등록

Vue Loader

- `.vue` 형태의 파일을 javascript 모듈 형태로 변환해주는 webpack loader
- Vue Loader 로 인해 얻게 되는 장점들은
    i. ES6 **지원**
    ii. `<style>` 과 `<template>` 에 대한 각각의 webpack loader 지원. ex) sass, jade
    iii. **각** `.vue` **컴포넌트의 스코프로 좁힌** css **스타일링 지원**
    iv. webpack 의 모듈 번들링에 대한 지원과 의존성 관리가 제공
    v. 개발 시 hot reloading 지원

Glossory : Virtual DOM

- React 와 마찬가지로 빠른 화면 렌더링을 위해 사용하는 Virtual DOM 을 Vue 도 사용하고 있다.
- Virtual DOM 은 화면의 DOM 조작시 유용하게 사용되는 기술이다
- 화면의 DOM 을 추가하거나 삭제하는 등의 변경이 일어날 때 마다, 전체 DOM 을 Reflow 하는 것이 아니라, 가상의 DOM 을 이용하여 추가되거나 삭제될 DOM 의 모양을 잡아놓고 한번만 DOM Reflow 를 수행함으로서 화면의 부하를 줄여 빠르게 그릴 수 있는 장점이 있다.

실습 #8 - Todo App 제작

Vue.js 를 사용하여 Todo App 을 제작해보자.

- 데이터 저장소 : Local Storage 활용
- Single File Component 사용 여부는 개인 선택
- 제작 후 Firebase 로 호스팅 및 Github Repository 주소 제출. jangkeehyo@gmail.com
    - 코딩 버디 조를 활용하여 같이 의견 나누면서 제작해보는 것을 권고드립니다.

참고 자료

- todo in official home
- TODO it!

Advanced 레벨을 위한 학습 커리큘럼 제안

- Vue Reactivity System
- Render Function
- State Management
- Server Side Rendering

참고

- Vue Official Doc
- Vue Router Offical Doc
- Vue Router - Slide Share
- Learn ES2015 Guide
- React vs Vue
- webpack-simple, Github Repo
- webpack from first principles
- webpack advanced
- NHN Enter
- Vue Loader Docs

끝