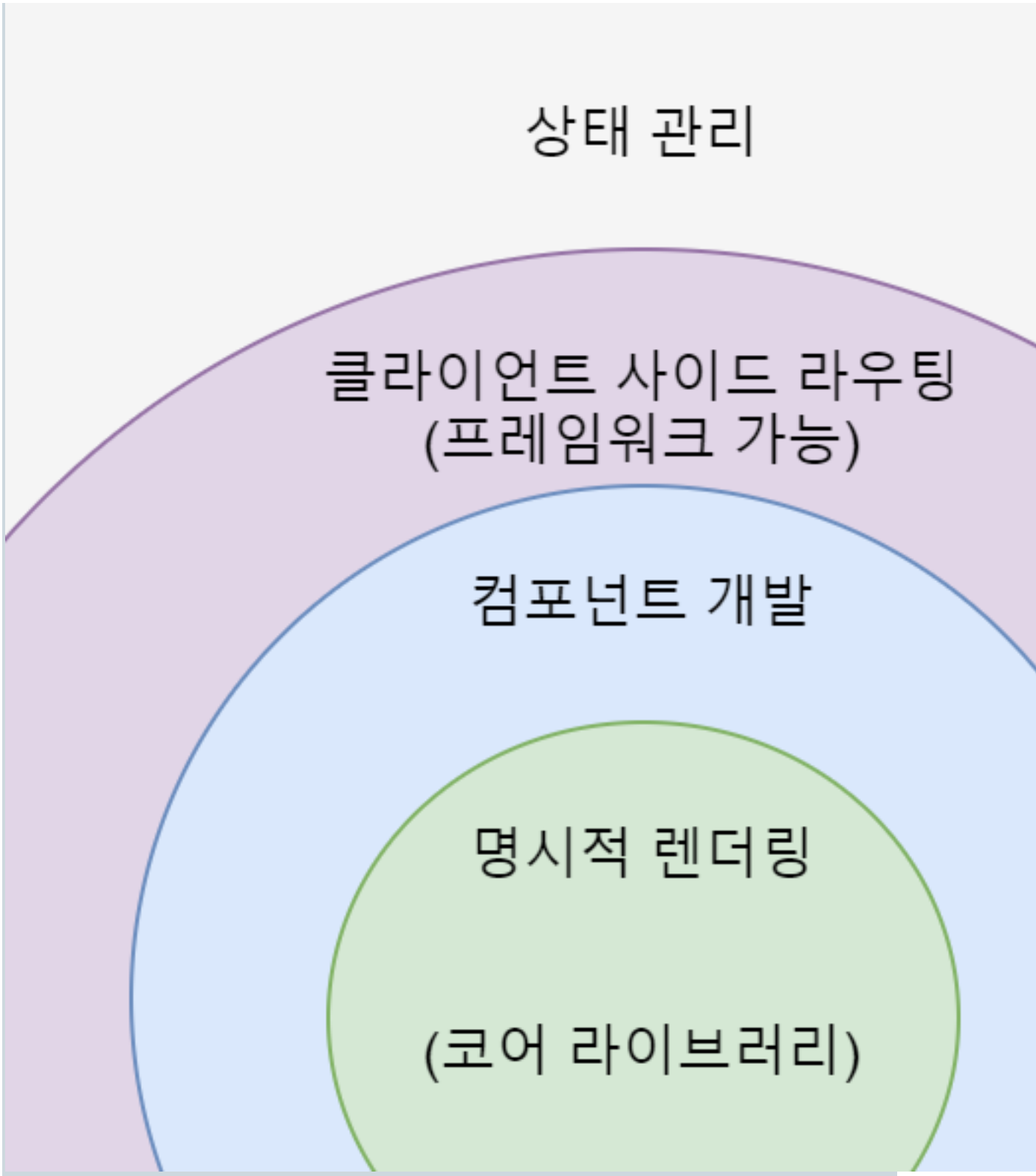
Front-End - Vue.js 란? 특징 및 장점?

오늘 포스팅할 내용은 프론트엔드 프레임워크 중에 하나인 vue.js 이다. 필자도 처음 다루어보는 프론트엔드이기 때문에 간단히 vue.js 가 무엇인지 알아보자.

Vue.js 란?

vue.js 는 웹 페이지 화면을 개발하기 위한 프론트엔드 프레임워크이다. vue.js 는 여타 다른 프론트엔드 프레임워크보다 배우기 쉽다는 장점이 있다. 리엑트와 앵귤러라는 프레임워크의 장점들을 쏙 빼와서 더욱 빠르고 가볍게 만든 프레임워크라고 한다.



vue.js 는 위 그림과 같은 구조를 가지고 있다. 코어 라이브러리는 화면단 데이터 표현에 관한 기능들을 중점적으로 지원하지만 프레임워크의 기능인 라우터, 상

태관리, 테스팅 등을 쉽게 결합할 수 있는 형태로도 제공된다. 즉, 단순 라이브러리의 기능 외에 프레임워크 역할도 수행하게 되는 것이다. 그렇다면 이러한 vue.js 의 장점은 무엇일까?

Vue.js 장점

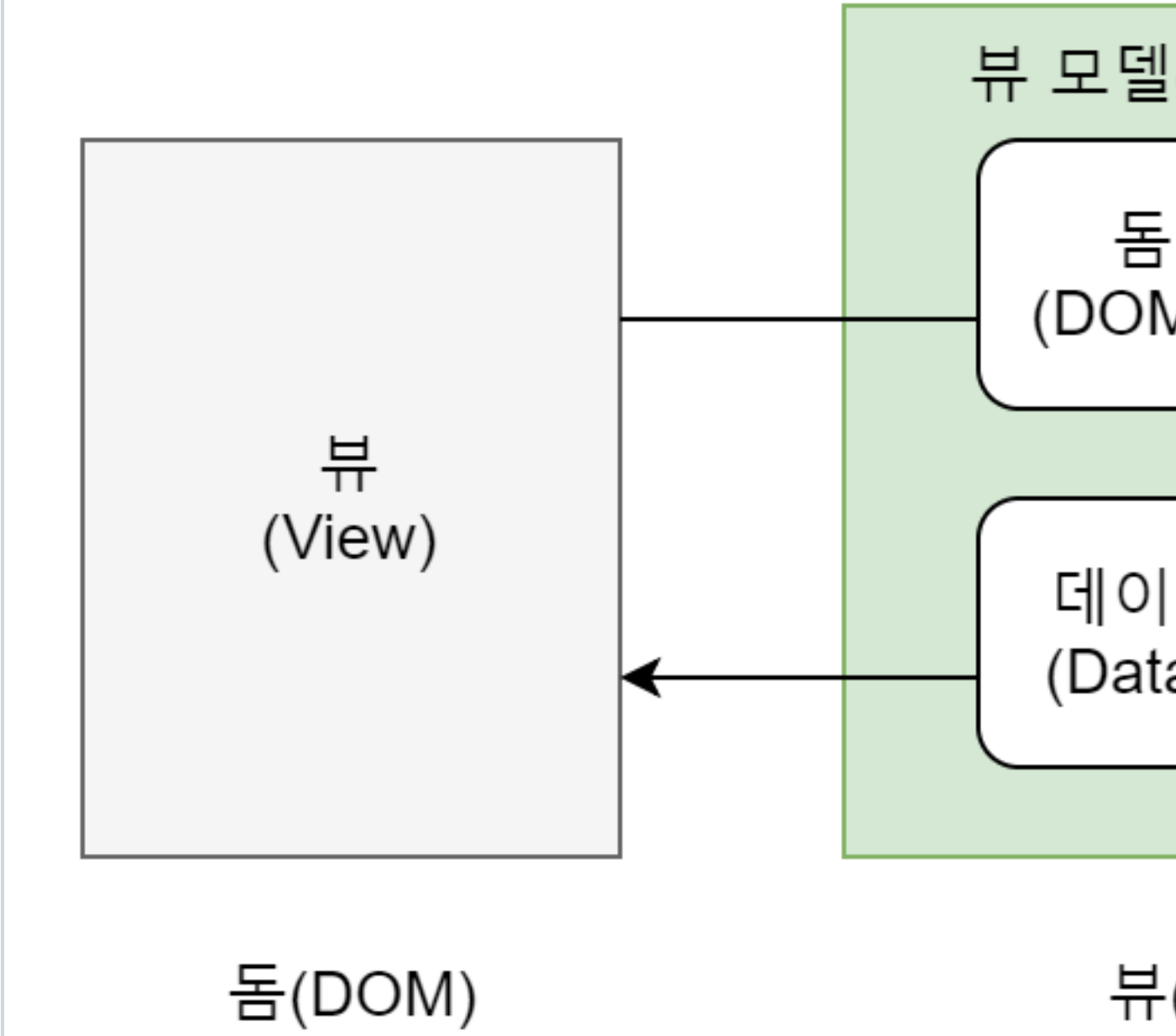
뷰 같은 경우는 앵귤러의 데이터 바인딩 특성과 리액트의 가상 돔 기반 렌더링 특징을 모두 가지고 있다.

1.배우기 쉽다.
2.리액트와 앵귤러에 비해 성능이 우수하고 빠르다.
3.리액트의 장점과 앵귤러의 장점을 가지고 있다.

<Vue.js 의 특징>

MVVC 패턴

뷰(Vue.js)는 UI 화면 개발 방법 중 하나인 MVVM 패턴의 뷰 모델에 해당하는 화면단 라이브러리이다.



위 그림에서 보듯, MVVM 패턴이란 화면을 모델(Model) - 뷰(View) - 뷰 모델(ViewModel)로 구조화하여 개발하는 방식을 의미한다. 이러한 방식으로 개발하는 이유는 화면의 요소들을 제어하는 코드와 데이터

제어 로직을 분리하여 코드를 더 직관적으로 이해할 수 있고, 추후 유지 보수가 편해진다.

용어	설명
뷰(View)	사용자에게 보이는 화면
돔(DOM)	HTML 문서에 들어가는 요소(태그,클래스,속성 등)의 정보를 담고 있는 데이터 트리
돔 리스너(DOM Listener)	돔의 변경 내역에 대해 즉각적으로 반응하여 특정 로직을 수행하는 장치
모델(Model)	데이터를 담는 용기. 보통은 서버에서 가져온 데이터를 자바스크립트 객체 형태로 저장
데이터 바인딩(Data Binding)	뷰(View)에 표시되는 내용과 모델의 데이터를 동기화

뷰 모델(ViewModel)	뷰와 모델의 중간 영역. 돔 리스너와 데이터 바인딩을 제공하는 영역

그렇다면 MVVM 구조의 처리 흐름은 어떻게 될까?

Vue js 란?

위와 같은 구글 검색 창이 있다고 생각해보자.(실제 구글은 뷰로 개발되지는 않았음. 단순 참고용)

여기서 뷰는 사용자에게 비춰지는 구글 검색화면 전체를 의미한다. 그리고 돔은 구글로고, 검색창, 키보드, Google 검색 버튼 등의 HTML 요소들을 뜻한다. 만약 검색어를 입력하고 Google 검색 버튼을 클릭하면 어떤일이 일어날까?

Vue js 란?

전체 이

검색결과 약 28,700

3. Vue.js란? -

https://wikidocs.

2018. 5. 11. - **Vue.j**

주기; Vue 인스턴스

자바스크립트

https://meetup.to

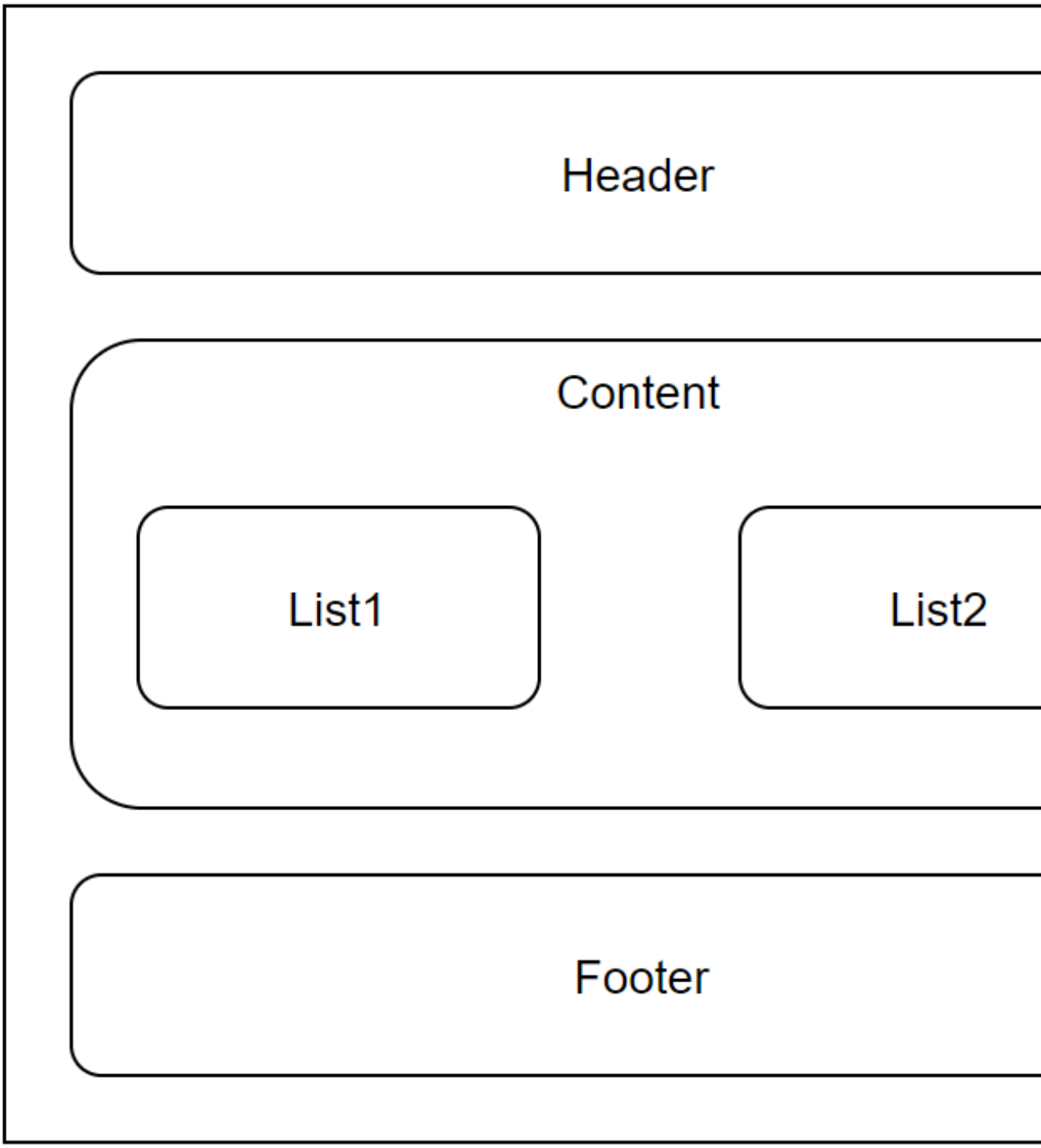
2016. 11. 8. - **Vue.j**

임워크이다. 앞서 소

위와 같은 검색 결과 화면을 보여줄 것이다. 즉, Google 검색 버튼을 클릭하였을 때, 돔 리스너에서 버튼의 클릭을 감지하고 버튼이 동작하면 검색 결과를 보여주는 로직이 실행되는 것이다. 즉, 이 처리 과정에서 데이터 바인딩이 관여하며 검색 결과에 해당하는 데이터를 모델에서 가져와 화면에 나타내 준다.

컴포넌트 기반 프레임워크

뷰가 가지는 또 하나의 큰 특징은 바로 컴포넌트 기반 프레임워크라는 것이다.



위 그림과 같이 컴포넌트란 레고 블록을 조합하는 것과 비슷하다. 즉, 화면을 왼쪽 같이 구성을 하면 오른

쪽과 같은 컴포넌트 트리 구조를 가지게 되는 것이다. 이러한 컴포넌트 기반의 프레임워크를 사용함으로써 코드의 재사용성이 향상되며 직관적인 화면 구조를 갖게 되는 것이다.

리액트와 앵귤러의 장점을 가진 프레임워크
뷰는 앵귤러의 양방향 데이터 바인딩과 리액트의 단방향 데이터 흐름의 장점을 모두 결합한 프레임워크이다. 양방향 데이터 바인딩이란 화면에 표시되는 값과 프레임워크의 모델 데이터 값이 동기화되어 한쪽이 변경되면 다른 한쪽도 자동으로 값이 변경되는 것을 의미한다. 단방향 데이터 흐름은 컴포넌트의 단방향 통신을 의미한다. 즉, 컴포넌트 간에 데이터를 전달할 때 항상 상위 컴포넌트에서 하위 컴포넌트 한 방향으로만 전달하게끔 프레임워크가 구조화되어 있는 것을 말한다.

이 외에도 뷰는 빠른 화면 렌더링을 위해 리액트의 가상 돔 렌더링 방식을 적용하여 사용자 인터랙션이 많은 요즘의 웹 화면에 적합한 동작 구조를 갖추고 있다. 가상 돔을 활용하면 특정 돔 요소를 추가하거나 삭제하는 변경이 일어날 때 화면 전체를 다시 그리지 않고 프레임워크에서 정의한 방식에 따라 화면을 갱신한다. 따라서 브라우저 입장에서는 성능 부하가 줄어들어 일반 렌더링 방식보다 더 빠르게 화면을 그릴 수 있다.

다음 포스팅부터는 실제 뷰를 다루어 볼 것이다.

Front-End - Vue.js 인스턴스(Vue 인스턴스)

뷰 인스턴스의 정의와 속성

뷰 인스턴스는 뷰로 화면을 개발하기 위해 필수적으로 생성해야 하는 기본 단위이다. 즉 뷰로 화면을 개발하기 위해 빠트릴 수 없는 필수 조건이다.

뷰 인스턴스 생성

뷰 인스턴스를 사용하기 위해 아래와 같은 형식으로 뷰 인스턴스를 생성한다.

```
1 <html>
2   <head>
3     <title>Vue Sample</title>
4   </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>
```

Colored by Color Scripter

위의 코드를 간단히 설명하자면 HTML 화면에 'Hello Vue.js !' 라는 텍스트를 출력하기 위한 코드이다. 일단 new Vue()로 뷰 인스턴스를 생성하였다. 물론 이전

에 Vue 인스턴스를 사용할 수 있게 vue.js 라이브러리를 내려받고 있는 코드가 존재한다. 그리고 인스턴스 안에 el 속성으로 해당 뷰 인스턴스의 범위를 설정해 주었고, data 속성에 message 값을 정의하여 화면의 {{message}}에 'Hello Vue.js !'가 출력되도록 하였다. 잘 보면 el 의 범위가 특정 div 의 아이디 값과 매칭되는 것이 보인다.(css 선택자) 즉, 해당 뷰 인스턴스는 app 이라는 아이디를 가진 특정 HTML 태그라는 범위를 갖고 사용되는 인스턴스인 것이다.

뷰 인스턴스 옵션 속성

뷰 인스턴스 옵션 속성은 인스턴스를 생성할 때 재정의할 data, el, template 등의 속성을 의미한다. 이 말은 무엇이냐, 이미 Vue 인스턴스에는 el,data,template 등의 속성이 선언되어 있지만 우리가 뷰 인스턴스를 생성할 때 해당 옵션들을 재정의 할 수 있다는 것이다. 위 코드에서는 data 라는 미리 선언된 옵션에 message 라는 data(key/value)를 재정의하였고 해당 값을 HTML 태그 내에서 사용하는 것을 볼 수 있다. 또한 해당 뷰 인스턴스의 범위(el)옵션을 '#app'으로 재정의하고 있다. 여기서 '#app'은 id 가 app 인 특정 돔 요소를 뜻한다.

속성	설명
template	화면에 표시할 HTML, CSS 등의 마크업 요소를 정의하는 속성.

	뷰의 데이터 및 기타 속성들도 함께 화면에 그릴 수 있다.
methods	화면 로직 제어와 관계된 메소드를 정의하는 속성. 마우스 클릭 이벤트 처리와 같이 화면의 전반적인 이벤트와 화면 동작과 관련된 로직을 추가 할 수 있다.
created	뷰 인스턴스가 생성되자마자 실행할 로직을 정의할 수 있는 속성. 뷰 인스턴스의 라이프 사이클과 관련된 속성이다. created 이외에 많은 라이프 사이클 훅을 정의할 수 있는 옵션이 존재한다.

뷰 인스턴스의 유효 범위

위에서 뷰 인스턴스의 유효범위라는 단어가 포함된 설명이 존재하였다. 그렇다면 자세히 뷰 인스턴스의 유효범위에 대하여 다루어보자. 뷰 인스턴스를 생성하면

HTML의 특정 범위 안에서만 옵션 속성들이 적용되어 나타난다. 이를 인스턴스의 유효 범위라고 한다. 다음 포스팅에서 다루어볼 "지역,전역 컴포넌트 이해"에도 중요한 내용이니 꼭 기억해야한다. 위에서 말했듯이 뷰 인스턴스의 유효범위는 el 속성과 밀접한 관계가 있다.

뷰 인스턴스의 유효 범위를 이해하려면 실제로 인스턴스의 라이플 사이클을 이해할 필요가 있다.


beforeCreate
인스턴스
생성
new Vue()
이벤트 및
라이프 사이클 초기

우리가 위에서 작성한 코드는 위의 라이프 사이클 흐름중 인스턴스를 화면에 부착하는 것까지의 행위를 정의한다. 먼저 자바스크립트 코드 상에서 인스턴스 옵션 속성 el 과 data 를 인스턴스에 정의하고 인스턴스를 생성하였다. 그리고 브라우저에서 코드를 실행시키면 el 속성에 지정된 HTML 의 특정 돔요소에 인스턴스가 부착된다.(부착된다의 의미를 그냥 적용된다라고 이해해도 좋다.) el 속성에 해당하는 특정 돔 요소에 뷰 인스턴스가 부착되고 나면 인스턴스에 정의한 data 속성이 el 속성에 지정한 특정 돔 요소에 값이 적용되고 화면의 {{message}}가 'Hello Vue.js !'라는 텍스트로 치환된다.

이해하기 힘들면 '아 그렇구나 !'하고 넘어가도 좋다. 나중 포스팅에서 다시 다루어보면서 해당 의미를 다시 한번 정확히 파악해 볼것이고, 이번 포스팅에서는 뷰가 무엇이고 어떻게 간단히 사용되는지 설명하는 포스팅이다.

그렇다면 현재 안에 있는 {{message}}를 해당 태그 밖으로 빼보자. 어떻게 되는가? {{message}}라는 문자열이 그대로 화면에 출력되는 것이 보일 것이다. 이것이 바로 뷰 인스턴스의 유효범위이다. app 이라는 아이디를 가진 특정 돔 요소에서 벗어나 버리면 뷰 인스턴스에 정의한 data 를 사용할 수가 없는 것이다. 꼭 기억하자.

뷰 인스턴스 라이프 사이클

앞에서 살펴본 뷰 인스턴스 옵션 중 created 라는 속성을 기억할 것이다. 해당 옵션은 인스턴스가 생성되었을 때 호출할 동작을 정의하는 속성이라고 설명하였다. 이처럼 인스턴스의 상태에 따라 호출할 수 있는 속성들을 라이프 사이클 속성이라고 한다. 그리고 각 라이프 사이클 속성에서 실행되는 커스텀 로직을 라이프 사이클 훅이라고 한다. 위의 그림에서 beforeUpdate, updated 사이클은 데이터의 변경이 없다면 지나지 않는 라이프 사이클임을 주의하자.

라이프 사이클 속성에는 총 8 개가 존재한다. 위의 그림을 같이 참조하여 살펴보자.

속성	설명
beforeCreate	인스턴스가 생성되고 나서 가장 처음으로 실행되는 라이프 사이클 단계이다. 이 단계에서는 data 속성과 methods 속성이 아직 인스턴스에 정의되어 있지 않고, 돔과 같은 화면 요소에도 접근할 수 없다.
created	beforeCreate 라이프

	사이클 단계 다음에 실행되는 단계이다. data 속성과 methods 속성이 정의되었기 때문에 this.data 또는 this.fetchData()와 같은 로직들을 이용하여 data 속성과 methods 속성에 정의된 값에 접근하여 로직을 실행할 수 있다. 다만, 아직 인스턴스가 화면 요소에 부착되기 이전이기 때문에 template 속성에 정의된 돔 요소로 접근할 수 없다. data 속성과 methods 속성에 접근할 수 있는 가장 첫 라이프 사이클 단계이자 컴포넌트가 생성되고 나서 실행되는 단계이기 때문에 서버에 데이터를 요청하여 받아오는 로직을 수행하기 좋다.
beforeMount	created 단계 이후

	template 속성에 지정한 마크업 속성을 render() 함수로 변환한 후 el 속성에 지정한 특정 화면 요소에 인스턴스를 부착하기 전에 호출되는 단계이다. render()가 호출되기 이전에 로직을 추가하기 좋다. render() - 자바스크립트로 화면의 돔을 그리는 함수
mounted	el 속성에서 지정한 화면 요소에 인스턴스가 부착되고 나면 호출되는 단계로, template 속성에 정의한 화면 요소에 접근할 수 있어 화면 요소를 제어하는 로직을 수행하기 좋은 단계이다. 다만, 돔에 인스턴스가 부착되자마자 바로 호출되기 때문에 하위

	컴포넌트나 외부 라이브러리에 의해 추가된 화면 요소들이 최종 HTML 코드로 변환되는 시점과 다를 수 있다. 변환되는 시점이 다를 경우 $nextTick() API를 이용하여 HTML 코드로 최종 파싱될 때까지 기다린 후 돔 제어 로직을 추가한다.
beforeUpdate	el 속성에서 지정한 화면 요소에 인스턴스가 부착되고 나면 인스턴스에 정의한 속성들이 화면에 치환된다. 치환된 값은 뷰의 반응성을 제공하기 위해 $watch 속성으로 감시한다. 또한 beforeUpdate는 관찰하고 있는 데이터가 변경되면 가상 돔으로 화면을 다시 그리기 전에 호출되는 단계이며,

	변경 예정인 새 데이터에 접근할 수 있어 변경 예정 데이터의 값과 관련된 로직을 미리 넣을 수 있다. 뷰의 반응성 : 뷰의 특징 중 하나로, 코드의 변화에 따라 화면이 반사적으로 반응하여 빠르게 화면을 갱신하는 것을 의미.
updated	데이터가 변경되고 나서 가상 돔으로 다시 화면을 그리고나면 실행되는 단계이다. 데이터 변경으로 인한 화면 요소 변경까지 완료된 시점이므로, 데이터 변경 후 화면 요소 제어와 관련된 로직을 추가하기 좋은 단계이다. 이 단계에서 데이터 값을 변경하면 무한 루프에 빠질 수 있기 때문에 값을

	변경하려면 computed, watch 와 같은 속성을 사용해야 한다. 따라서 데이터 값을 갱신하는 로직은 가급적이면 beforeUpdate 에 추가하고, updated 에서는 변경 데이터의 화면 요소와 관련된 로직을 추가하는 것이 좋다.
beforeDestroy	뷰 인스턴스가 파괴되기 직전에 호출되는 단계이다. 이 단계에서는 아직 인스턴스에 접근할 수 있다. 따라서 뷰 인스턴스의 데이터를 삭제하기 좋은 단계이다.
destroyed	뷰 인스턴스가 파괴되고 나서 호출되는 단계이다. 뷰 인스턴스에 정의한 모든 속성이 제거되고 하위에 선언한 인스턴스들 또한 모두 제거된다.

예제 코드를 살펴보자.

```
<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 !'
          },beforeCreate: function(){
            console.log("beforeCreate");
          },
          created: function(){
            console.log("created");
          },
          mounted: function(){
            console.log("mounted");
          },
          updated: function(){
            console.log("updated");
          }
        });
```

```
        </script>
    </body>
</html>
```

Colored by Color Scripter

위코드를 수행해보자. 어떠한 로그값이 출력될까? 한번 생각해보고 결과를 예측해보자.

결과는 beforeCreate-created-mounted 만 출력될 것이고, updated 는 출력되지 않을 것이다. 라이프 사이

클 시작점에서 필자가 이야기 했던 부분이 있다. data가 변경되지 않으면 지나지 않는 라이프 사이클이 존재한다고 했는 데, 그 부분이 beforeUpdate, updated라는 라이프 사이클이다.

그렇다면 데이터를 변경하는 로직을 삽입해보자. 과연 updated 라는 로그가 출력될까?

```
<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 !'
          },beforeCreate: function(){

            console.log("beforeCreate");
          },
          created: function(){
            console.log("created");
          },
```

```
            mounted: function(){
              console.log("mounted");
              this.message="Hello Vue !?";
            },
            updated: function(){
              console.log("updated");
            }
          });
        </script>
  </body>
</html>
```

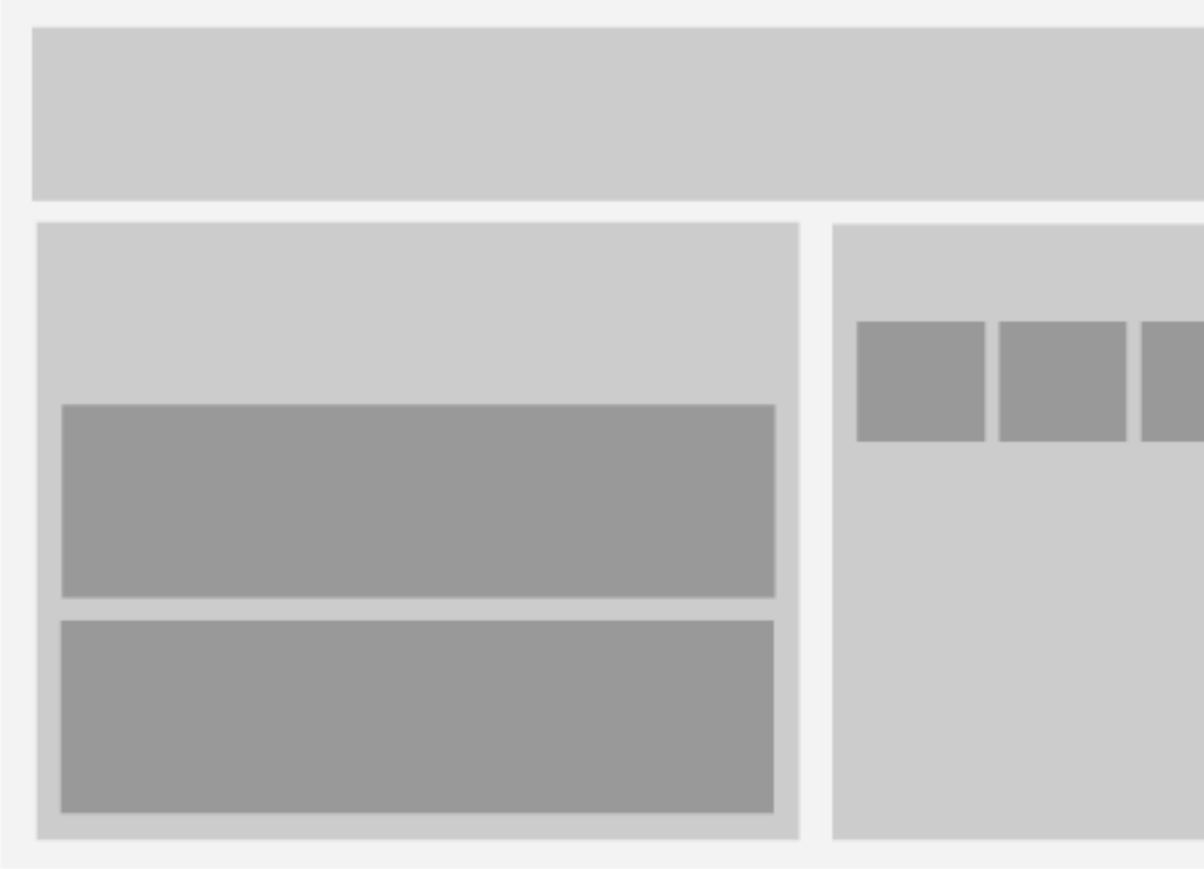
3
3

mounted 라는 라이프 사이클에서 message 데이터 값을 변경하였기 때문에 updated 라는 라이프 사이클을 지나게 되므로 updated 라는 로그가 출력되는 것을 볼 수 있다.

여기까지 뷰 인스턴스에 대해 간략히 다루어보았다.
다음 포스팅에서는 뷰 컴포넌트에 대한 포스팅을 진행할 것이다.

Front-End - Vue.js 컴포넌트란?(뷰 컴포넌트)

오늘 포스팅할 내용은 Vue 컴포넌트이다. 여기서 말하는 컴포넌트란 무엇일까? 컴포넌트는 하나의 블록을 의미한다. 레고처럼 여러 블럭을 쌓아서 하나의 집을 모양을 만들 듯이, 컴포넌트를 활용하여 화면을 만들면 보다 빠르게 구조화하여 일괄적인 패턴으로 개발할 수 있다. 이렇게 화면의 영역을 컴포넌트로 쪼개서 재활용할 수 있는 형태로 관리하면 나중에 코드를 다시 사용하기가 훨씬 편리하다. 또한 모든 사람들이 레고의 사용설명서처럼 정해진 방식대로 컴포넌트를 등록하거나 사용하게 되므로 남이 작성한 코드를 직관적으로 이해할 수 있다.



위 그림에서 보듯이 하나의 페이지를 여러 블록으로 나누었고, 여러 블록으로 조합하여 하나의 화면으로 만들었다. 그리고 하나의 특징이라면 하나의 블록 안에 또 다른 블록들이 들어가 있는 상-하위 구조를 가진다. 이러한 컴포넌트들은 커다란 최상위 컴포넌트

위에 작성되어 있기 때문에 오른쪽 그림과 같이 Root 컴포넌트를 시작으로 트리 구조를 형성한다.

컴포넌트 등록하기

컴포넌트를 등록하는 방법은 전역과 지역 두 가지가 있다. 지역 컴포넌트는 특정 인스턴스내에서만 유효한 범위를 갖고, 전역 컴포넌트는 여러 인스턴스에서 공통으로 사용할 수 있다. 즉, 지역은 특정 범위 내에서만 사용할 수 있고, 전역은 뷰로 접근 가능한 모든 범위에서 사용할 수 있다(new Vue 로 만든 각각의 인스턴스를 의미).

전역 컴포넌트

전역 컴포넌트는 뷰 라이브러리를 로딩하고 나면 접근 가능한 Vue 변수를 이용하여 등록한다. 전역 컴포넌트를 모든 인스턴스에 등록하려면 Vue 생성자에서 .component()를 호출하여 수행한다.

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

CS

전역 컴포넌트 등록 형식에는 컴포넌트 이름과 컴포넌트 내용이 있다. 컴포넌트 이름은 template 속성에서 사용할 HTML 사용자 정의 태그 이름을 의미한다. 그리고 컴포넌트 태그가 실제 화면의 HTML 요소로 변환될 때 표시될 속성들을 컴포넌트 내용에 작성한다.

컴포넌트 내용에는 template, data, methods 등 인스턴스 옵션 속성을 정의할 수 있다.

```
1
2
3  <html>
4    <head>
5      <title>Vue Sample</title>
6    </head>
7    <body>
8        <div id="app">
9          <button>컴포넌트 등록</button>
10         <my-component></my-component>
11       </div>
12       <script src="https://cdn.jsdelivr.net
11 /npm/vue@2.5.2/dist/vue.js"></script>
11       <script>
12         Vue.component('my-component',{
13           template: '<div>전역 컴포넌트가 등
11 록되었습니다 !</div>'
14         });
11         new Vue({
15           el:'#app'
11         });
16       </script>
11   </body>
17 </html>
11
18
11
19
```

```
2
0
2
1
```

HTML 태그로 전역 컴포넌트의 이름을 사용하고, 해당 태그의 내용은 전역 컴포넌트의 내용으로 표시된다. 결과는 아래와 같다.

전역 컴포넌트가 등록되었습니다 !

지역 컴포넌트 등록

지역 컴포넌트 등록은 전역 컴포넌트 등록과는 다르게 인스턴스에 components 속성을 추가하고 등록할 컴포넌트 이름과 내용을 정의한다.

```
new Vue({
    ...
    components:{
```

```
        'componentName':componentContent
    }
})
```

Colored by Color Scripter

바로 예제를 살펴보자.

```
<html>
  <head>
    <title>Vue Sample</title>
  </head>
  <body>
      <div id="app">
        <button>컴포넌트 등록</button>
        <my-component></my-component>
        <my-local-component></my-local-component>
      </div>
      <hr>
      <div id="app2">
        <button>컴포넌트 2 등록</button>
        <my-component></my-component>
        <my-local-component></my-local-component>
      </div>
      <script src="https://cdn.jsdelivr.net/npm/vue@2.5.2/dist/vue.js"></script>
      <script>
        Vue.component('my-component',{
          template: '<div>전역 컴포넌트가 등
```

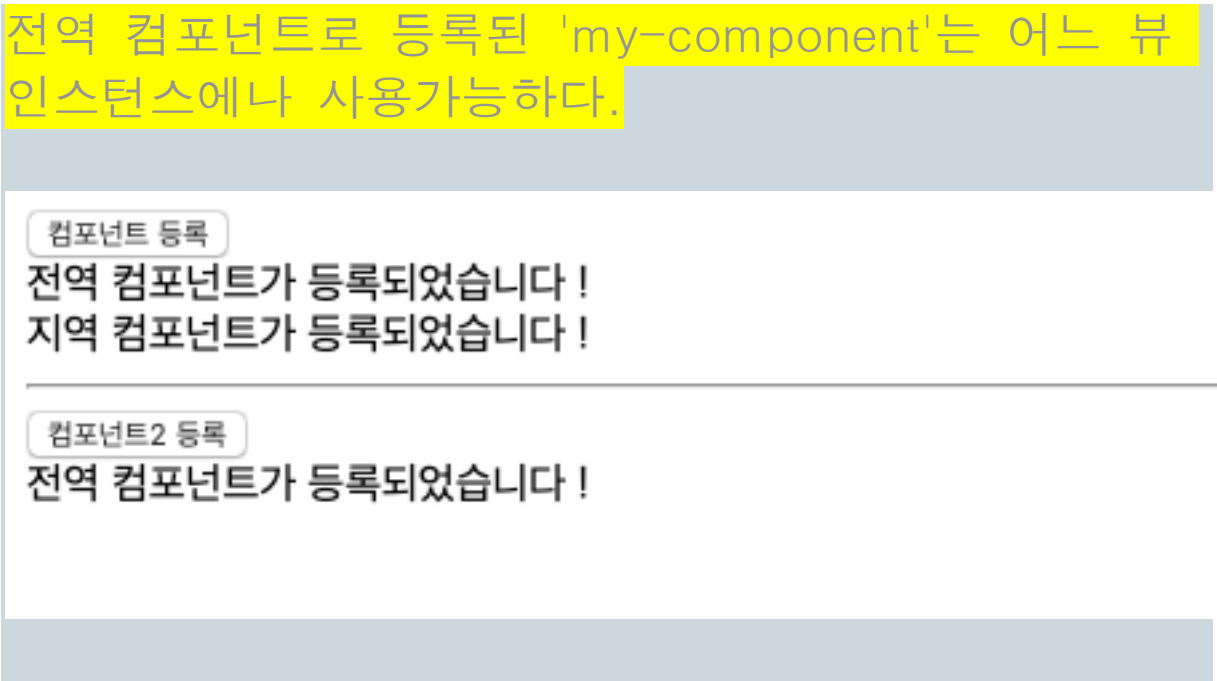
```
록되었습니다 !</div>'
        });

        var cmp = {
          template: '<div>지역 컴포넌트가 등
록되었습니다 !</div>'
        }

        new Vue({
          el:'#app',
          components:{
            'my-local-component': cmp
          }
        });

        new Vue({
          el: '#app2'
        })
      </script>
  </body>
</html>
```

```
3
3
4
3
5
3
6
3
7
3
8
3
9
4
0
```

여기서 전역과 지역의 유효 범위까지 살펴볼 것이다. 예제코드에서는 전역 컴포넌트와 지역 컴포넌트를 모두 작성하였다. 그리고 특정 태그 내에 컴포넌트들을 등록하였는데, 결과는 두 영역이 다르게 나타난다. app 이라는 id 를 가진 돔 요소는 전역,지역 컴포넌트를 모두 가지지만 app2 라는 id 를 가진 돔 요소는 전역 컴포넌트만 가지게 된다. 그 이유는 처음에서 설명한 유효성 범위와 관련있다.

전역 컴포넌트는 어느 인스턴스에나 사용가능한 컴포넌트이지만 지역 컴포넌트는 특정 인스턴스 내에서만 사용가능한 컴포넌트이기 때문에 app 이라는 유효범위 내에서 설정된 'my-local-component' 같은 경우는 app 이라는 id 를 가진 돔 요소에서만 사용가능하고,

전역 컴포넌트로 등록된 'my-component'는 어느 뷰 인스턴스에나 사용가능하다.



시작하기

Vue.js 가 무엇인가요?

Vue(/vjuː/ 로 발음, **view** 와 발음이 같습니다.)는 사용자 인터페이스를 만들기 위한 **프로그레시브 프레임워크** 입니다. 다른 단일형 프레임워크와 달리 Vue 는 점진적으로 채택할 수 있도록 설계하였습니다. 핵심 라이브러리는 뷰 레이어만 초점을 맞추어 다른 라이브러리나 기존 프로젝트와의 통합이 매우 쉽습니다. 그리고 Vue 는 **현대적 도구** 및 **지원하는 라이브러리**와 함께 사용한다면 정교한 단일 페이지 응용프로그램을 완벽하게 지원할 수 있습니다.

Vue 를 알아보기 전에 Vue 에 대해 자세히 알고 싶다면 핵심 원칙과 샘플 프로젝트를 바탕으로 설명하는 **비디오**를 만들었으니, 참고하시면 좋을 것입니다.

이미 숙련된 프론트엔드 개발자이고 Vue 를 다른 라이브러리/프레임워크와 비교하고 싶다면 **다른 프레임워크와의 비교**를 확인하십시오.

Vue Mastery의 무료 강의 보기

시작하기

설치

> 공식 가이드는 HTML, CSS 및 JavaScript 에 대한 중간 수준의 지식을 전제로 하고 있습니다. 이제 막 프론트 엔드 개발에 대해 배우기 시작했다면 첫 번째 단계로 프레임워크를 시작하는 것은 좋은 생각이 아닙니다. 기본을 파악한 다음 다시 해보세요! 다른 프레임워크에 대한 사전 경험이 도움될 수 있지만 반드시 필요한것은 아닙니다.

Vue.js 를 시험해 볼 수 있는 가장 쉬운 방법은 JSFiddle Hello World 예제를 사용하는 것입니다. 브라우저의 새 탭에서 열어 본 후 몇 가지 기본 예제를 따라가십시오. 또는 단순히 `index.html` 파일을 만들고 Vue 를 다음과 같이 포함할 수 있습니다.

```
<!-- 개발버전, 도움되는 콘솔 경고를 포함. -->
<script
src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

또는:

```
<!-- 상용버전, 속도와 용량이 최적화됨. -->
<script
src="https://cdn.jsdelivr.net/npm/vue"></script>
```

설치 페이지에는 Vue 를 설치하기 위한 옵션이 추가로 제공됩니다. 특히 Node.js 기반 빌드 도구에 아직 익숙하지 않으면 `vue-cli` 로 시작하는 것을 권장하지 **않습니다.**

보다 인터랙티브한 학습을 원한다면, 언제든지 중지/재생 할 수 있는 스크린캐스트와 코드를 테스트 할 수 있는 환경를 제공해주는 Scrimba 의 튜토리얼 시리즈를 확인해보세요.

선언적 렌더링

Scrimba에서 레슨 받기

Vue.js 의 핵심에는 간단한 템플릿 구문을 사용하여 DOM 에서 데이터를 선언적으로 렌더링 할 수있는 시스템이 있습니다.

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

```
var app = new Vue({
  el: '#app',
  data: {
    message: '안녕하세요 Vue!'
  }
})
```

안녕하세요 Vue!

첫 Vue 앱을 성공적으로 만들었습니다! 이것은 문자열 템플릿을 렌더링하는 것과 매우 유사하지만, Vue.JS 내부에서는 더 많은 작업을 하고 있습니다. 이제 데이터와 DOM 이 연결되었으며 모든 것이 **반응형**이 되었습니다. 우리는 그것을 어떻게 확인할 수 있을까요? 브라우저의 JavaScript 콘솔을 열고 `app.message` 를 다른 값으로 설정해 보십시오. 위 예제가 업데이트 변경된 값에 따라 업데이트되는 것을 볼 수 있습니다.

더 이상 HTML 과 직접 상호작용할 필요가 없습니다. Vue 앱은 단일 DOM 요소(우리의 경우 `#app`)에 연결되어 DOM 요소를 완전히 제어합니다. HTML 는 엔트리 포인트일뿐 다른 모든 것은 새롭게 생성된 Vue 인스턴스 내에서 발생합니다.

텍스트 보간 이외에도 다음과 같은 엘리먼트 속성을 바인딩할 수 있습니다.

```
<div id="app-2">
  <span v-bind:title="message">
    내 위에 잠시 마우스를 올리면 동적으로 바인딩 된 title 을 볼 수 있습니다!
  </span>
</div>
```

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: '이 페이지는 ' + new Date() + ' 에 로드 되었습니다'
  }
})
```

내 위에 잠시 마우스를 올리면 동적으로 바인딩 된 title을 볼 수 있습니다!(마우스를 두고 약간 대기하면 툴팁 툴력)

여기서 우리는 새로운 것을 만났습니다. `v-bind` 속성은 **디렉티브** 이라고 합니다. 디렉티브는

Vue 에서 제공하는 특수 속성임을 나타내는 `v-` 접두어가 붙어있으며 사용자가 짐작할 수 있듯 렌더링 된 DOM 에 특수한 반응형 동작을 합니다. 기본적으로 "이 요소의 `title` 속성을 Vue 인스턴스의 `message` 속성으로 최신 상태를 유지합니다."
다시 JavaScript 콘솔을 열고 `app2.message = '새로운 메시지'`라고 입력하면 HTML(이 경우에 `title` 속성)이 업데이트되었음을 다시 한번 확인할 수 있습니다.

조건문과 반복문

Scrimba에서 레슨 받기

엘리먼트가 표시되는지에 대한 여부를 제어하는 것은 아주 간단합니다.

```
<div id="app-3">
  <p v-if="seen">이제 나를 볼 수 있어요</p>
</div>
```

```
var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
```

```
})
```

이제 나를 볼 수 있어요

계속해서, Javascript 콘솔에 `app3.seen = false` 를 입력하면, 메시지가 사라지는 것을 볼 수 있습니다.

이 예제는 텍스트와 속성뿐 아니라 DOM 의 **구조**에도 데이터를 바인딩 할 수 있음을 보여줍니다. 또한 Vue 엘리먼트가 Vue 에 삽입/업데이트/제거될 때 자동으로 **트랜지션 효과**를 적용할 수 있는 강력한 전환 효과 시스템을 제공합니다.

몇가지 디렉티브가 있습니다. 각 디렉티브마다 고유한 기능이 있습니다. 예를 들어 `v-for` 디렉티브는 배열의 데이터를 바인딩하여 Todo 목록을 표시하는데 사용할 수 있습니다.

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

```
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: 'JavaScript 배우기' },
      { text: 'Vue 배우기' },
```

```
      { text: '무언가 멋진 것을 만들기' }
    ]
  }
})
```

1. JavaScript 배우기
2. Vue 배우기
3. 무언가 멋진 것을 만들기

콘솔에서, `app4.todos.push({ text: 'New item' })`을 입력하십시오. Todo 목록에 새 항목이 동적으로 추가 된것을 볼 수 있습니다.

사용자 입력 핸들링

Scrimba에서 레슨 받기

사용자가 앱과 상호 작용할 수 있게 하기 위해 우리는 `v-on` 디렉티브를 사용하여 Vue 인스턴스에서 메소드를 호출하는 이벤트 리스너를 추가 할 수 있습니다 :

```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">메시지 뒤집기</button>
</div>
```

```
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: '안녕하세요! Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message =
this.message.split('').reverse().join('')
    }
  }
})
```

안녕하세요! Vue.js!

메시지 뒤집기

이 방법은 직접적으로 DOM 을 건드리지 않고 앱의 상태만을 업데이트합니다. 모든 DOM 조작은 Vue 에 의해 처리되며 작성한 코드는 기본 로직에만 초점을 맞춥니다.

Vue 는 또한 양식에 대한 입력과 앱 상태를 양방향으로 바인딩하는 `v-model` 디렉티브를 제공합니다.

(input 에 데이터를 입력하면 메세지가 같이 변경되서 연동이 된다. 양방향 데모)

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
```

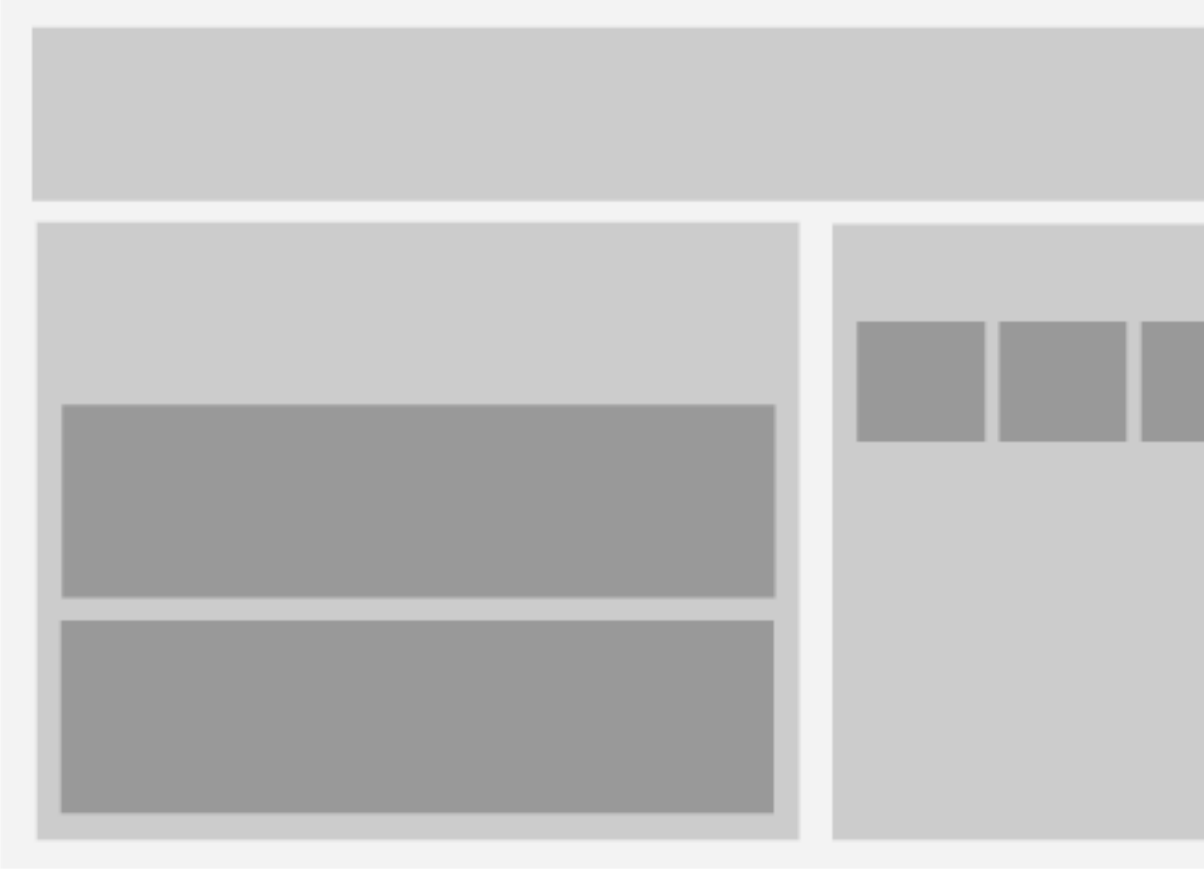
```
</div>
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: '안녕하세요 Vue!'
  }
})
```

안녕하세요 Vue!

컴포넌트를 사용한 작성방법

Scrimba에서 레슨 받기

컴포넌트 시스템은 Vue 의 또 다른 중요한 개념입니다. 이는 작고 독립적이며 재사용할 수 있는 컴포넌트로 구성된 대규모 애플리케이션을 구축할 수 있게 해주는 추상적 개념입니다. 생각해보면 거의 모든 유형의 애플리케이션 인터페이스를 컴포넌트 트리로 추상화할 수 있습니다.



Vue 에서 컴포넌트는 미리 정의된 옵션을 가진 Vue 인스턴스 입니다. Vue 에서 컴포넌트를 등록하는 방법은 간단합니다.

```
// todo-item 이름을 가진 컴포넌트를 정의합니다
```

```
Vue.component('todo-item', {
  template: '<li>할일 항목 하나입니다.</li>'
})

var app = new Vue(...)
```

이제 다른 컴포넌트의 템플릿에서 이 컴포넌트를 추가할 수 있습니다.

```
<ol>
  <!-- todo-item 컴포넌트의 인스턴스 만들기 -->
  <todo-item></todo-item>
</ol>
```

그러나 이것은 `todo-item` 컴포넌트를 사용할 때마다 똑같은 텍스트를 렌더링할뿐 무언가가 부족합니다. 부모 영역의 데이터를 자식 컴포넌트에 전달할 수 있어야 합니다. **prop** 을 전달받을 수 있도록 `todo-item` 컴포넌트의 정의를 수정해봅시다.

```
Vue.component('todo-item', {
  // 이제 todo-item 컴포넌트는 "prop" 이라고 하는
  // 사용자 정의 속성 같은 것을 입력받을 수 있습니다.
  // 이 prop 은 todo 라는 이름으로 정의했습니다.
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

이제 `v-bind`를 사용하여 각각의 반복되는 `todo-item` 컴포넌트에 전달할 수 있습니다.

```
<div id="app-7">
  <ol>
    <!--
      이제 각 todo-item 에 todo 객체를 제공합니다.
      화면에 나오므로, 각 항목의 컨텐츠는 동적으로 바뀔 수 있습니다.
      또한 각 구성 요소에 "키"를 제공해야합니다 (나중에 설명 됨).
    -->
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id"
    ></todo-item>
  </ol>
</div>
```

```
Vue.component('todo-item', {
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})

var app7 = new Vue({
  el: '#app-7',
  data: {
    groceryList: [
      { id: 0, text: 'Vegetables' },
      { id: 1, text: 'Cheese' },
      { id: 2, text: 'Whatever else
```

```
humans are supposed to eat' }
      ]
    }
})
```

1. Vegetables
2. Cheese
3. Whatever else humans are supposed to eat

이것은 인위적으로 만든 예시이지만, 우리는 앱을 두 개의 더 작은 단위로 나눌 수 있었고, 자식을 props 인터페이스를 통하여 부모로부터 합리적인 수준으로 분리할 수 있었습니다. 이제 앞으로는 부모 앱에 영향을 주지 않으면서 `<todo-item>` 컴포넌트를 더 복잡한 템플릿과 로직으로 더욱 향상시킬 수 있을 것입니다.

대규모 애플리케이션에서는 개발을 보다 쉽게 관리할 수 있도록 전체 앱을 컴포넌트로 나누는 것이 필수적입니다. **가이드의 뒷부분**에서 컴포넌트에 대해 자세히 설명하겠지만 여기서는 컴포넌트를 사용한 앱의 모습이 어떻게 구성될지에 대한 (가상의) 예를 작성하겠습니다.

```
<div id="app">
  <app-nav></app-nav>
  <app-view>
    <app-sidebar></app-sidebar>
```

```
    <app-content></app-content>
  </app-view>
</div>
```

사용자 정의 엘리먼트와의 관계

Vue 컴포넌트는 Web Components Spec 의 일부인 **사용자 지정 엘리먼트** 와 매우 유사하다는 것을 눈치 챘을 수 있습니다. Vue 의 컴포넌트 구문은 스펙 이후 느슨하게 모델링 되었기 때문입니다. 예를 들어 Vue 컴포넌트는 Slot API 와 `is` 특수 속성을 구현합니다. 그러나 몇가지 중요한 차이가 있습니다.

1. Web Components Spec 은 최종안이 정해졌지만 모든 브라우저들이 기본적으로 구현되는 것은 아닙니다. 현재 사파리 10.1+, 크롬 54+ 그리고 파이어폭스 63+ 기본적으로 Web Components 를 지원합니다. 이에 비해 Vue 컴포넌트는 지원되는 모든 브라우저 (IE 9 이상)에서 폴리필을 필요로 하지 않으며 일관된 방식으로 작동합니다. 필요한 경우 Vue 컴포넌트는 기본 사용자 정의 엘리먼트 내에 래핑할 수 있습니다.

2. Vue 컴포넌트는 컴포넌트간 데이터의 흐름을 비롯해, 사용자 정의 이벤트와 통신, 빌드 도구와의 통합 등 기본 사용자 지정 엘리먼트에서 사용할 수 없었던 중요한 기능을 제공합니다.

Vue 는 내부적으로 사용자 정의 엘리먼트를 사용하지 않지만, 사용자 정의 엘리먼트로 사용 또는 배포하는 경우에는 **뛰어난 상호운용성**을 가집니다. Vue CLI 는 자기자신을 네이티브 커스텀 엘리먼트로서 등록하는 Vue 컴포넌트의 빌드도 지원하고 있습니다.