

Vue.js를 이용한 프론트 엔드 개발

Vue.js 소개

- ▶ Vue.js : 웹 페이지 화면을 개발하기 위한 라이브러리와 프레임워크
 - ▶ 라이브러리 : 소프트웨어를 개발할 때 사용하는 비휘발성 자원의 모임.
주로 반복 사용될 기능들을 재활용할 수 있는 형태로 모아둔 것
 - ▶ 프레임워크 : 개발 생산성 향상을 위해 일정한 규칙과 관례에 따라 개발할 수 있도록 미리 구조를 정의해 놓은 뼈대.
 - ▶ 라이브러리로서의 Vue.js : 화면단 데이터 표현에 관한 기능들 중점 지원
 - ▶ 프레임워크로서의 Vue.js : 라우터, 상태 관리, 테스트 등을 쉽게 결합할 수 있는 프레임워크의 형태로도 제공
-
- ▶ Rerefence
 - ▶ https://docs.google.com/presentation/d/1zQ3Frm3DxSw_qY-KEuykkIURE0-ueFb0yMd1Kd8nqKE/edit#slide=id.g15ca90c7ab_0_0
 - ▶ <https://kr.vuejs.org/v2/guide/>



Evan You

Vue.js 소개

pro·gres·sive

/prə'gresiv/

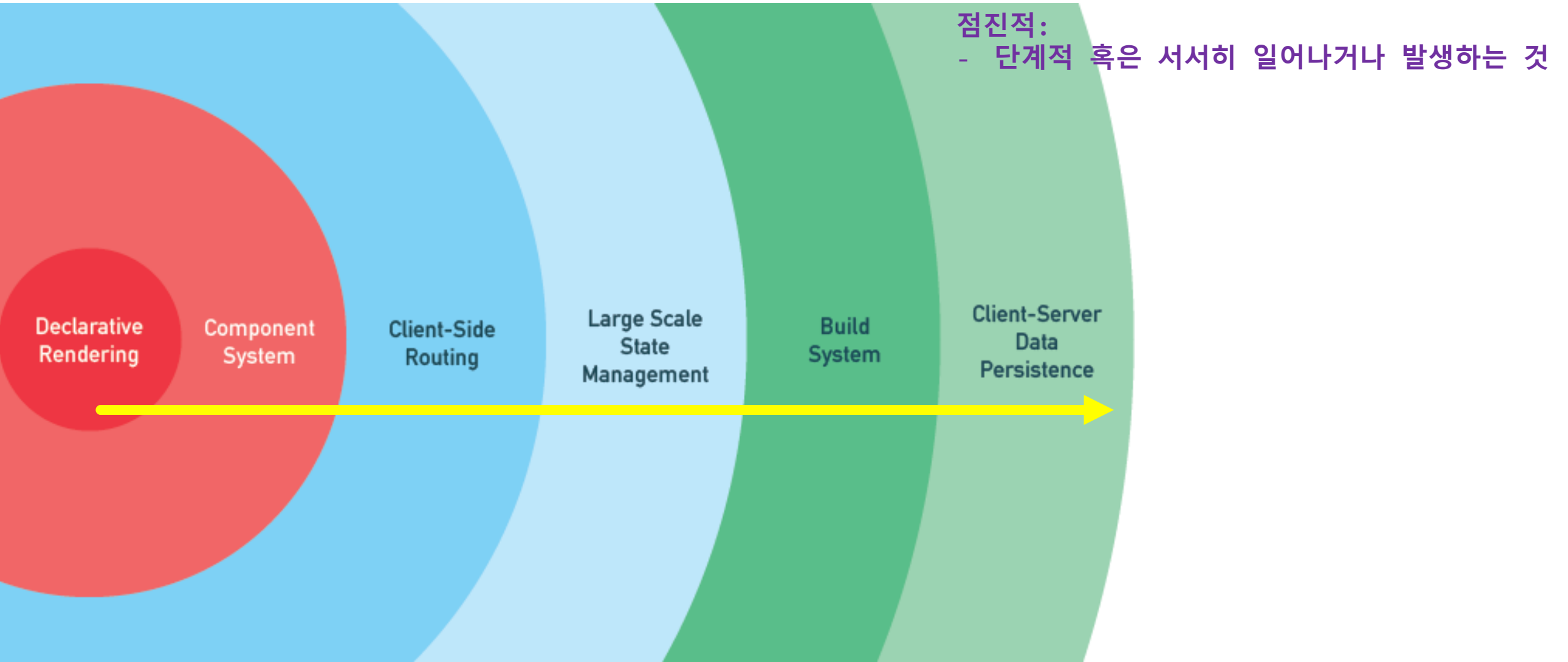
adjective

1. happening or developing gradually or in stages; proceeding step by step.

"a progressive decline in popularity"

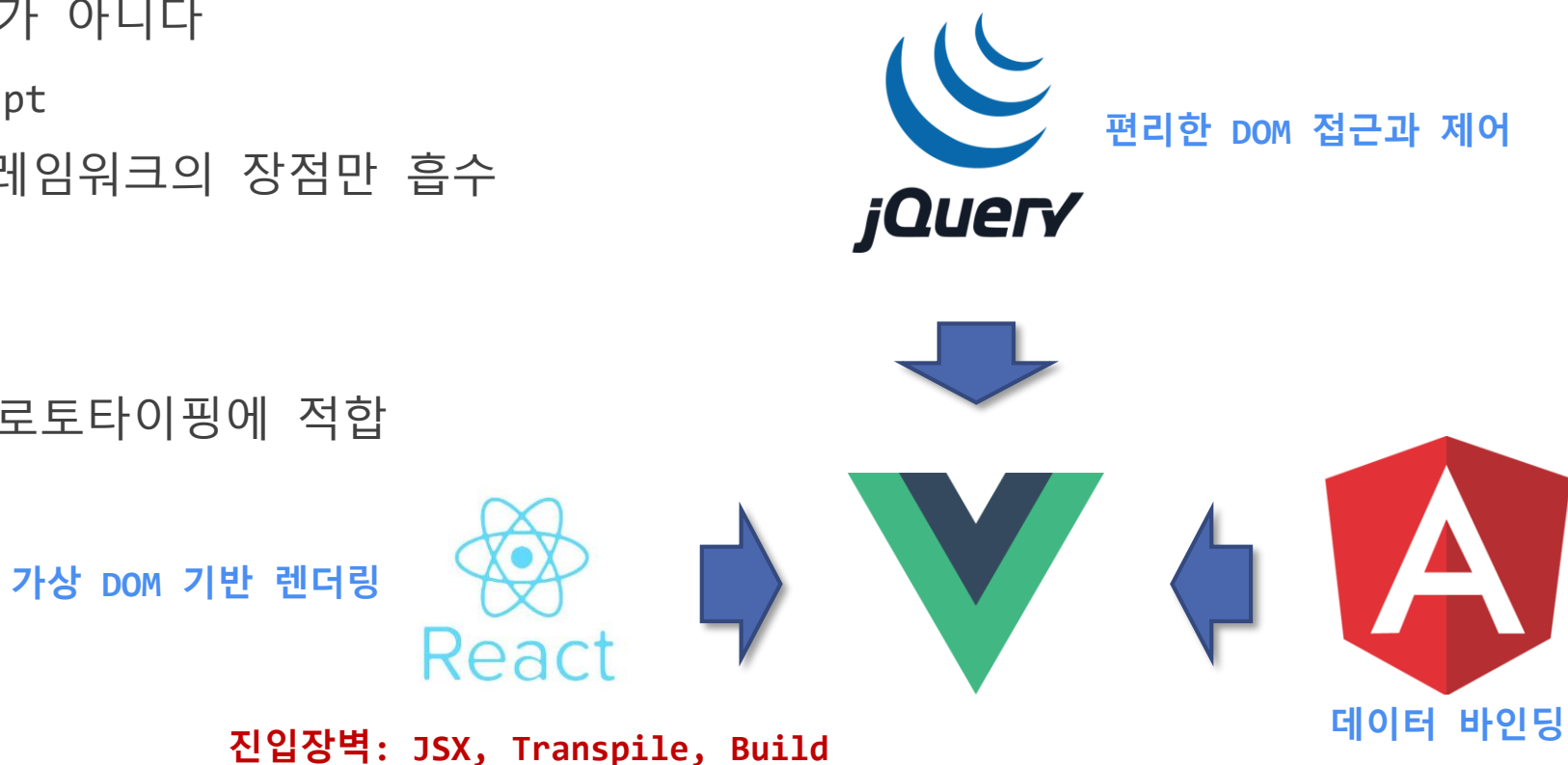
synonyms: continuing, **continuous**, increasing, growing, developing, **ongoing**, accelerating, escalating; **More**

▶ 점진적인 프레임워크(Progressive Framework)



Vue.js의 장점

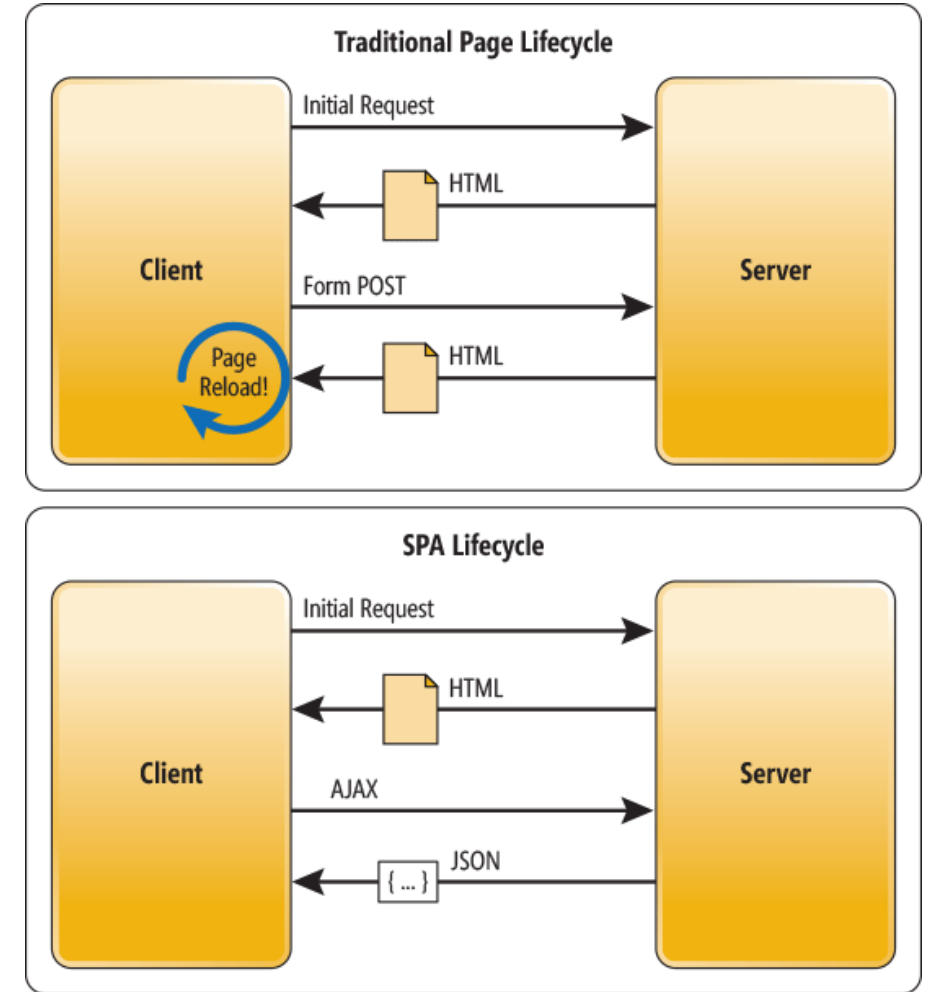
- ▶ 초기 진입 장벽이 상대적으로 낮다
 - ▶ HTML/CSS/JavaScript에 대한 이해만 있으면 손쉽게 도입, 적용이 가능
 - ▶ Build 작업이 필수가 아니다
 - ▶ Babel, TypeScript
 - ▶ 타 라이브러리, 프레임워크의 장점만 흡수
 - ▶ 가볍고 빠르다
-
- ▶ SPA 개발 입문, 프로토타이핑에 적합



<https://kr.vuejs.org/v2/guide/comparison.html>

Single Page Application

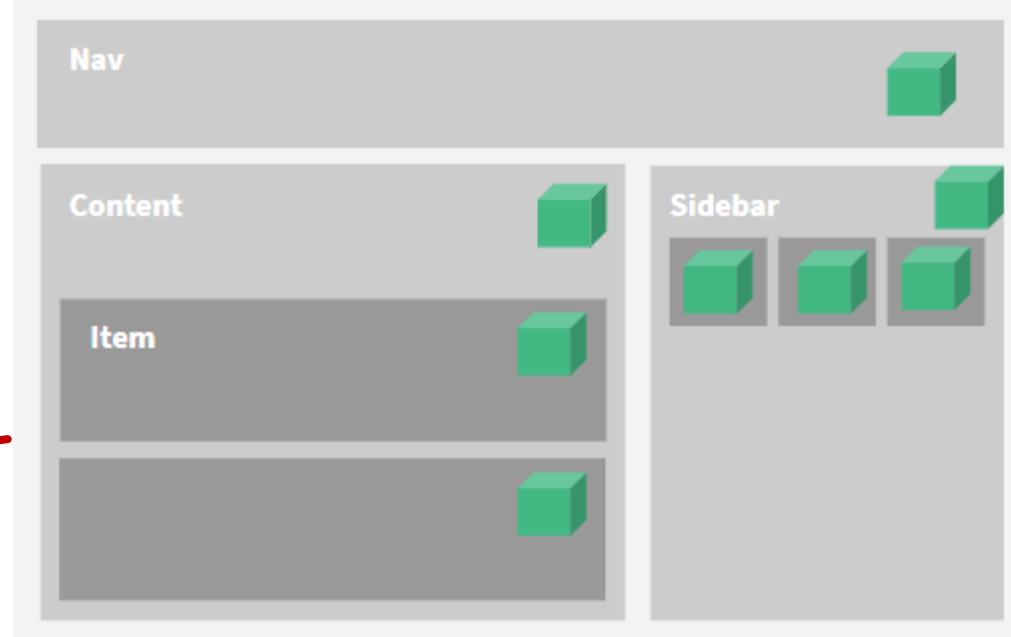
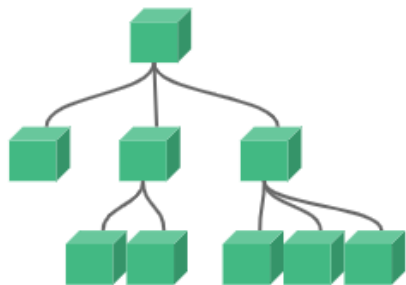
- ▶ SPA : 기존 웹 개발 방식과는 다른 방식
 - ▶ Traditional Way : 웹 서버가 전체 페이지를 전송하고 브라우저는 단순 렌더링만 하던 방식
 - ▶ SPA Way : 브라우저가 전체 페이지를 매번 렌더링하지 않고 서버는 필요한 정보만 브라우저에 전송, 브라우저는 해당 부분만 다시 렌더링하는 방식
- ▶ SPA의 장점
 - ▶ App과 비슷한 사용자 경험을 제공할 수 있음
 - ▶ Flickering 없음
 - ▶ 유지보수의 편의성
 - ▶ 본격적인 Front-End와 Back-End의 역할 분담과 협업
 - ▶ 점진적 개발이 가능
 - ▶ 컴포넌트 단위의 개발 및 조합



Traditional Page vs SPA

Vue.js의 특징

- ▶ UI 화면단 라이브러리
 - ▶ MVVM 패턴의 뷰 모델(ViewModel)에 해당하는 화면단 라이브러리
- ▶ 컴포넌트 기반 프레임워크
- ▶ 컴포넌트의 이해
 - ▶ 컴포넌트 : 독립적인 기능을 수행하는 소프트웨어 모듈
 - ▶ 대부분의 앱 UI는 안에 내포된 컴포넌트로 쪼갤 수 있다
 - ▶ 각각의 컴포넌트는 DOM의 조각을 관리할 책임이 있다
 - ▶ 전체 UI는 컴포넌트들의 트리로 추상화될 수 있다



개발 환경 설정

- ▶ Visual Studio Code

- ▶ 추천 확장(Extension) : Vetur, Live Server

- ▶ Vue.js DevTools

- ▶ 구글에서 vue.js devtools로 검색
 - ▶ Chrome, Firefox, Safari 등 브라우저별 확장 혹은 플러그인 제공

- ▶ Node.js (Optional)

- ▶ Vue CLI(Command Line Interface) 등을 설치, 활용할 수 있다

Vue Instance

: 첫 번째 Vue Application



```
...  
<!-- Vue.js Script 불러오기 -->  
<script src="https://unpkg.com/vue@2.6.10/dist/vue.js"></script>  
...  
<!-- Vue 렌더링 영역 -->  
<div id="app">  
  {{ message }}  
</div>  
...  
<!-- Vue 인스턴스 만들기 -->  
<script>  
var app = new Vue({  
  el: "#app", // Selector - id로 선택하자!  
  data: {  
    message: "Hello"  
  }  
});  
</script>
```

View

Model

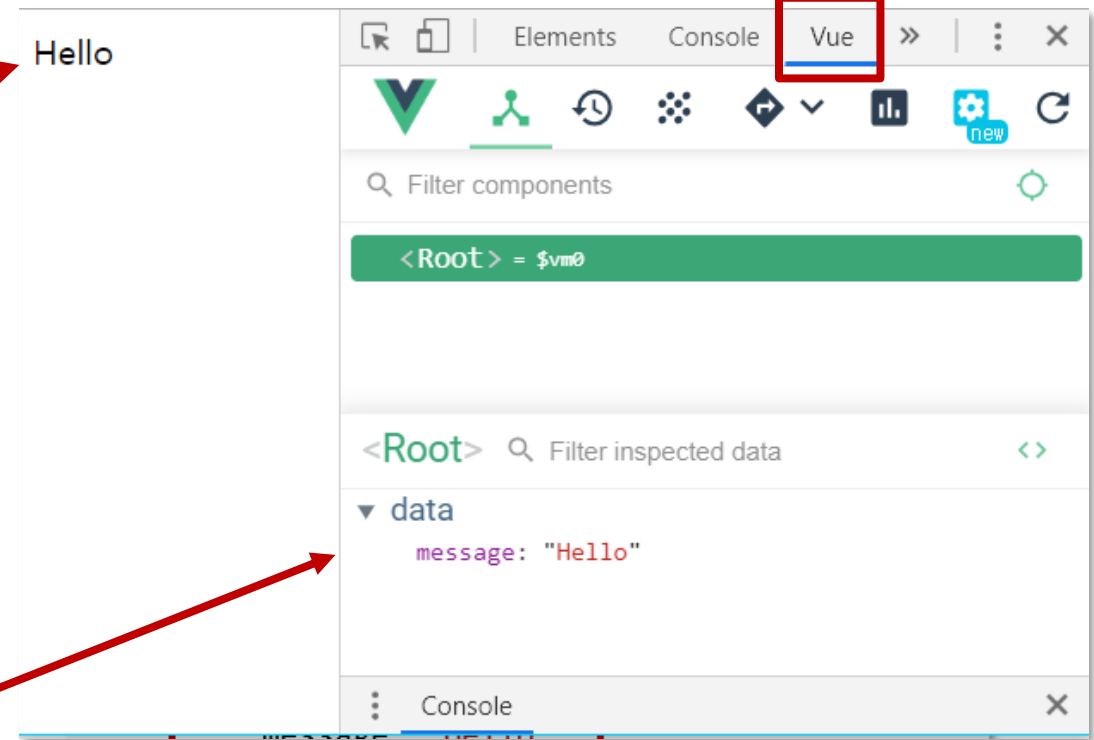
Vue Instalce

: 첫 번째 Vue Application

▶ Vue DevTools에서 확인

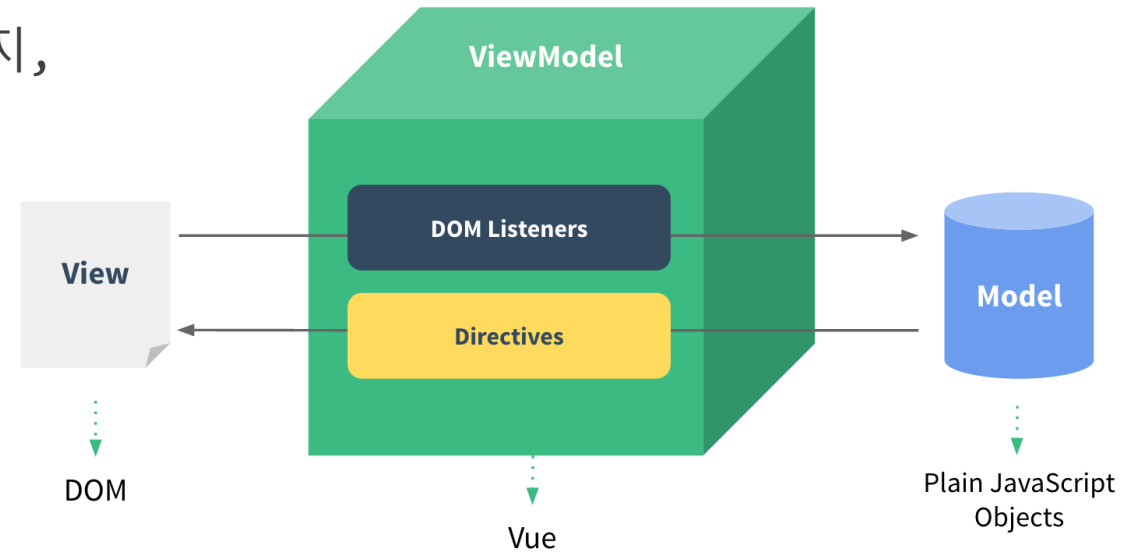
```
...  
<!-- Vue 렌더링 영역 -->  
<div id="app">  
  {{ message }}  
</div>  
...  
<!-- Vue 인스턴스 만들기 -->  
<script>  
var app = new Vue({  
  el: "#app", // Selector - id로 선택 하자!  
  data: {  
    message: "Hello"  
  }  
});  
</script>
```

Rendering



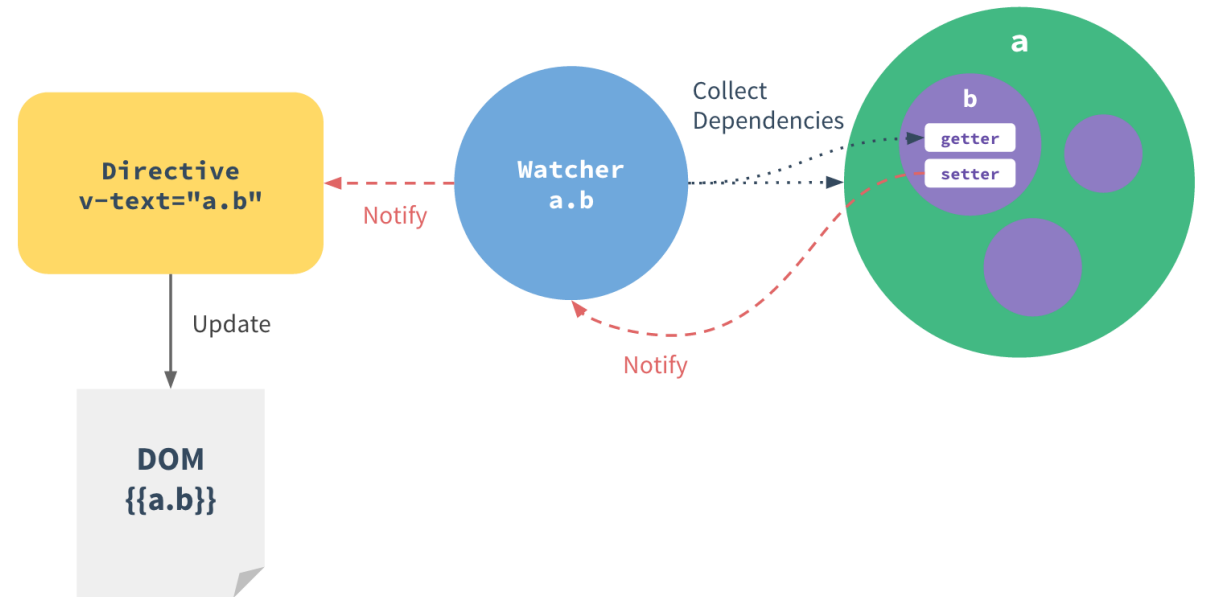
Vue Instance

- ▶ Vue Instance는 MVVM 패턴의 ViewModel에 해당
 - ▶ View(DOM)과 Model(Plain JavaScript Object) 사이의 통신을 해 주는 역할
- ▶ Vue Instance는 Binder를 가지고 있어, View와 Model을 서로 맞춰줌
 - ▶ Data Binding이 자동으로 가능한 이유
- ▶ DOM Listener : DOM의 여러 이벤트들을 감지, 변경된 내용을 Model에 반영
- ▶ Directives : JavaScript Object에 있는 내용을 View에 드러낼 수 있게 해주는 각종 '지시사항'들(선언적)



Vue Instance

- ▶ Vue의 구성 요소들은 Model의 변화에 반응적(Reactively)으로 대응
- ▶ Model의 빈번한 변화는 DOM의 잦은 갱신과 재 렌더링 작업을 수행하게 되고 웹 페이지 성능상에 영향을 미치게 됨
- ▶ Vue는 React가 도입했던 Virtual DOM 개념을 차용, DOM의 일부를 복사본으로 만들어 두고, Virtual DOM에서 대부분의 Binding과 Rendering 작업을 수행한 후 최종적으로 실제 DOM에 반영 (실제 DOM의 변경 최소화, 성능 개선)



Vue Instance

: Instance Lifecycle

created

- data 속성과 method 속성의 정의
- 데이터 요청 로직 수행에 적합한 단계



mounted

- 화면 요소에 인스턴스 부착
- 화면 요소 제어 로직 수행에 적합 단계



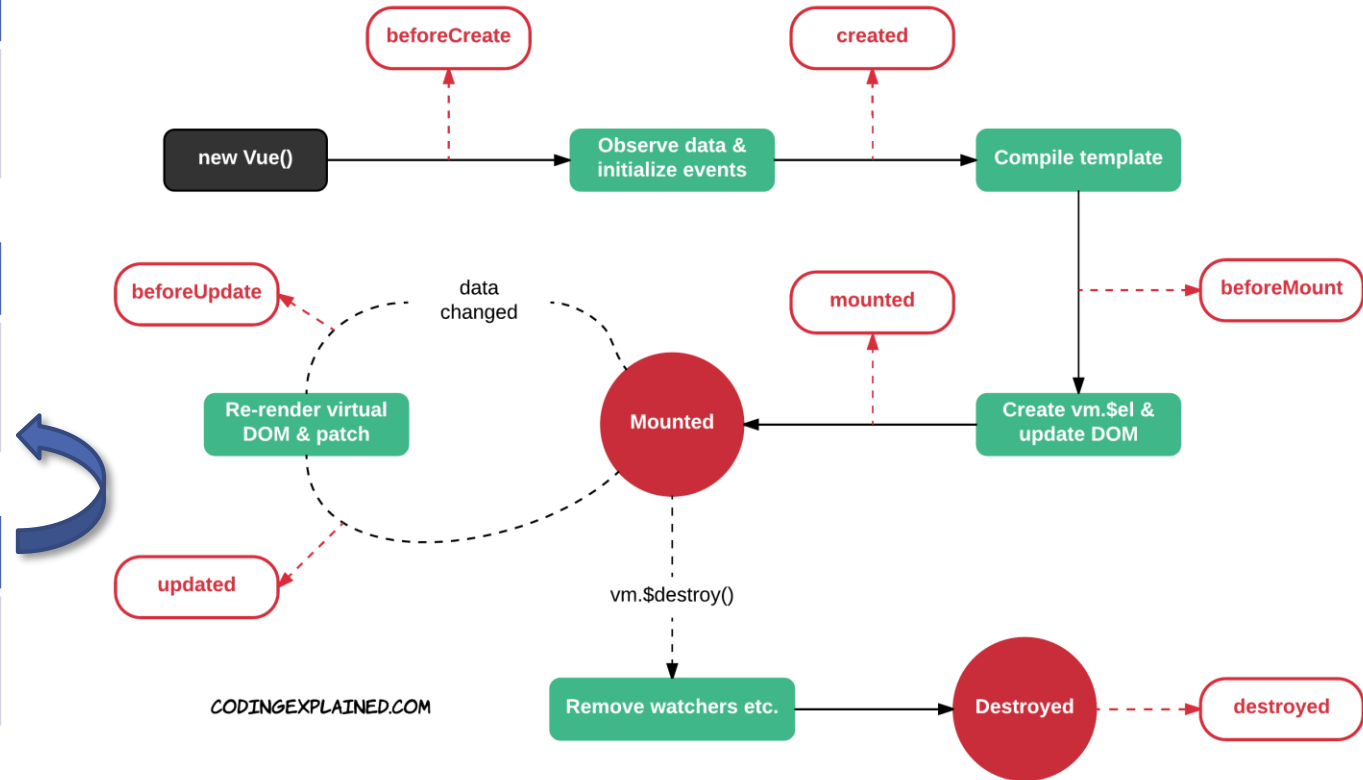
updated

- 데이터 변경 후 화면 요소 변경 완료
- 데이터 변경 후 화면 요소 제어에 적합



destroyed

- 뷰 인스턴스 파괴 후 호출
- 뷰 인스턴스와 하위 인스턴스들이 모두 제거



Vue Instance

: Instance Properties

- ▶ Vue Instance가 만들어질 때, Option Object를 Properties로 지정할 수 있음
 - ▶ <https://kr.vuejs.org/v2/guide/instance.html#%EC%86%8D%EC%84%B1%EA%B3%BC-%EB%A9%94%EC%86%8C%EB%93%9C>
- ▶ Data 프로퍼티
 - ▶ 컴포넌트 내부에 사용할 변수를 생성하고 지정할 수 있음
 - ▶ Data Binding을 가능하게 해주는 속성
 - ▶ 단, 한 컴포넌트 내에서 정의된 Data는 해당 컴포넌트 내에서만 접근이 가능
 - ▶ 컴포넌트가 살아있는 한, 내부 Data 변수들은 변경사항을 모두 반영하고 접근 가능함
 - ▶ 일반적인 JS 문법으로도 접근 가능
 - ▶ 코드상에서는 보통 this 키워드를 이용하여 접근
 - ▶ 만약(인스턴스가) 삭제됐다가 다시 만들어졌다면, 초기 상태로 돌아 감

Vue Instance

: Instance Properties

- ▶ Methods 프로퍼티
 - ▶ Vue Instance를 사용하면서 활용 가능한 함수들
 - ▶ Data, Event, Lifecycle을 활용하는 함수들로 구성
- ▶ Lifecycle Hooks
 - ▶ Vue Instance의 각 생애 주기별로 호출되는 콜백 함수들
 - ▶ 특정 Object의 초기화나 삭제시에 유용

Vue Component

- ▶ 컴포넌트 : 독립적인 기능을 수행하는 소프트웨어 모듈
 - ▶ Vue의 컴포넌트는 화면을 구성할 수 있는 블록(화면의 영역)을 의미
 - ▶ 화면을 빠르게 구조화하여 일괄적인 패턴으로 개발할 수 있음
 - ▶ 코드의 재사용에 유리
- ▶ 전역 컴포넌트의 선언
 - ▶ 전역 컴포넌트는 여러 인스턴스에서 공통으로 사용할 수 있음

```
<script>
  // 전역 컴포넌트 선언
  Vue.component("global-header", {
    template: "<div>Header</div>";
  });
</script>
```

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

<script>
  ...
  new Vue({
    el: "#app"
  });
</script>
```

Vue Component

▶ 지역 컴포넌트의 선언과 사용

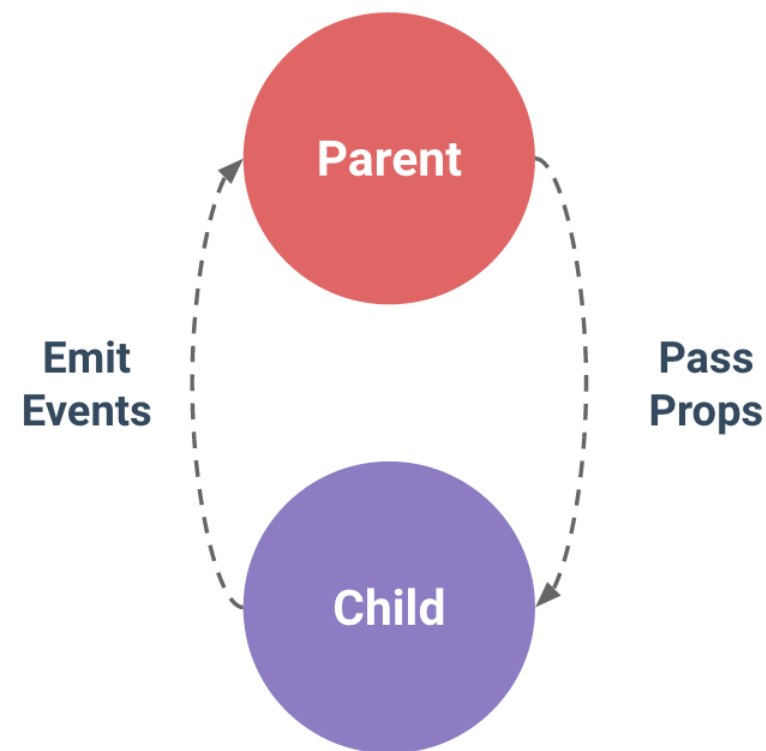
- ▶ 지역 컴포넌트는 Vue 인스턴스 내에 components 속성을 추가하여 등록할 컴포넌트의 이름과 내용을 정의
- ▶ 지역 컴포넌트는 새 Vue 인스턴스가 생성될 때마다 등록해 줘야 한다

```
<script>  
// 지역 컴포넌트 선언  
var cmp1 = {  
  template: '<div>Local Component</div>'  
}  
</script>
```

```
<div id="app">  
  <local-component></local-component>  
</div>  
  
<script>  
  ...  
  new Vue({  
    el: "#app",  
    components: {  
      'local-component': cmp1  
    }  
  });  
</script>
```


Vue Component 통신

- ▶ Vue는 컴포넌트 단위로 화면을 구성
 - ▶ 같은 웹 페이지 내에 있더라도 데이터를 공유할 수 없음
 - ▶ 컴포넌트마다 자체적으로 고유한 유효 범위(Scope)를 갖기 때문
- ▶ 기본적인 컴포넌트간 데이터 전달 방식
 - ▶ 부모 -> 자식 : props 전달
 - ▶ 자식 -> 부모 : 이벤트 발생



Vue Component 통신

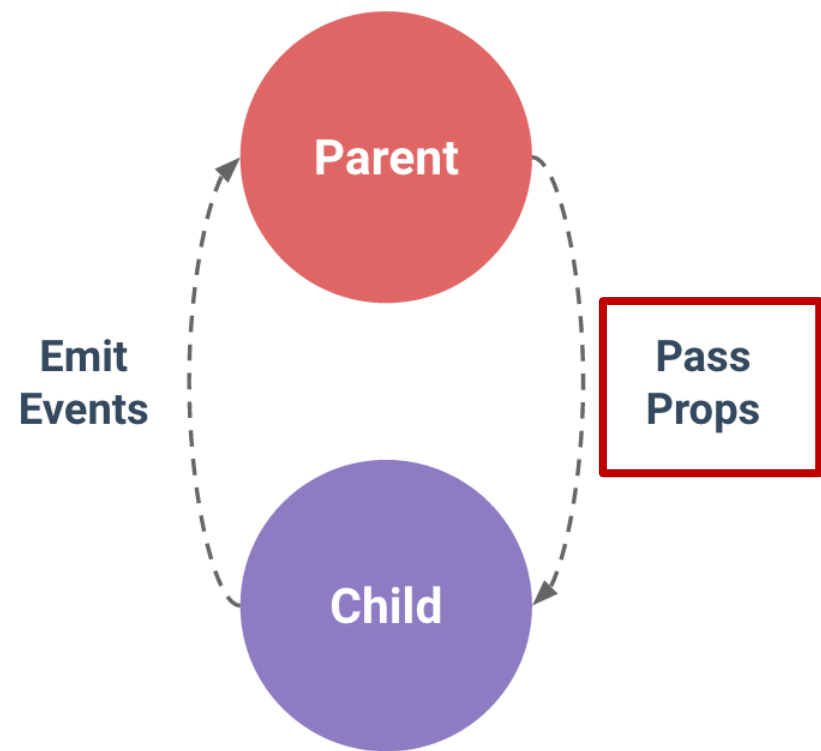
: props

- ▶ props 속성 : 상위 컴포넌트에서 하위 컴포넌트로 데이터를 전달할 때 사용
 - ▶ 하위 컴포넌트에 전달 받을 props 속성명을 정의

```
<script>  
// 지역 컴포넌트 선언  
var cmp1 = {  
  props: ['propsdata'],  
  template: "<p>{{propsdata}}</p>"  
}  
</script>
```

- ▶ HTML에 v-bind 속성을 설정

```
<local-component v-bind:propsdata="message">  
</local-component>
```



상위 컴포넌트

```
new Vue({  
  ...  
  data: {  
    message: "Message From Parent"  
  },  
  ...  
});
```

Vue Component 통신

: 이벤트 발생과 수신

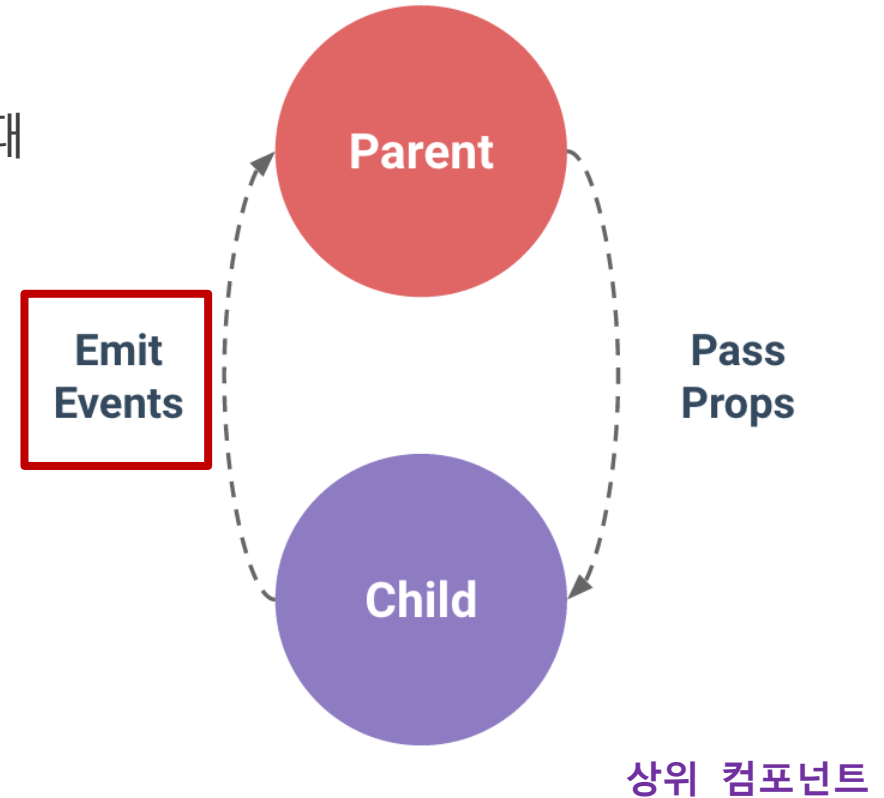
- ▶ 하위 컴포넌트에서 상위 컴포넌트로 데이터를 전달하고자 할 때
 - ▶ 이벤트를 발생(Event Emit), 상위 컴포넌트에 신호를 보냄

```
<script>
// 지역 컴포넌트 선언
var cmp1 = {
  template: "<button v-on:click='sendMsg'>Send</button>",
  methods: {
    sendMsg: function() {
      this.$emit("send-msg", "Message From Child")
    }
  }
}
</script>
```

- ▶ HTML에 v-on 속성을 설정

```
<local-component v-on:send-msg="rcvMsg">
</local-component>
```

```
new Vue({
  ...
  methods: {
    rcvMsg: function(message) {
      alert("Signal From Child");
    }
  }
  ...
});
```



Vue Component 통신

: 비 부모 자식간 통신

- ▶ 기본적으로 Vue는 상위-하위 컴포넌트간 데이터 통신을 지원
 - ▶ 경우에 따라서는 상위-하위 컴포넌트 구조를 벗어난 컴포넌트간 통신이 필요
 - ▶ 이벤트를 중계할 수 있는 별도의 Vue 인스턴스를 하나 생성, 상위-하위 관계를 벗어난 컴포넌트간에도 데이터를 주고받을 수 있다

```
// EventBus  
var eventBus = new Vue();
```

```
var sender = {  
  ...  
  methods: {  
    sendMsg: function() {  
      eventBus.$emit("triggerEventBus",  
        "Message From Other");  
    }  
  }  
  ...  
}
```

```
var receiver = new Vue({  
  ...  
  created: function() {  
    eventBus.$on("triggerEventBus",  
      function(message) {  
        ...  
      });  
  }  
});
```

Vue Template

- ▶ Vue의 템플릿 : 뷰 인스턴스에서 정의한 데이터 및 로직들을 HTML, CSS 등의 마크업 속성과 연결, 브라우저에서 볼 수 있는 형태의 HTML로 변환해주는 속성
- ▶ Vue 템플릿에서 사용하는 뷰의 속성과 문법들
 - ▶ 데이터 바인딩
 - ▶ 자바스크립트 표현식
 - ▶ 디렉티브
 - ▶ 이벤트 처리
 - ▶ 고급 템플릿 기법 등
- ▶ template 속성에서 정의한 마크업 + 뷰 데이터를 가상 돔 기반의 render() 함수로 변환
 - ▶ 변환된 render() 함수는 최종적으로 사용자가 볼 수 있게 화면을 그리는 역할
 - ▶ 뷰의 반응성(Reactivity)이 화면에 추가됨

Vue Template

: 데이터 바인딩

- ▶ 콧수염(Mustache) 괄호 - {{ }}
- ▶ 가장 기본적인 텍스트 삽입 방식

```
...  
<!-- Vue 렌더링 영역 -->  
<div id="app">  
  {{ message }}  
</div>  
...  
<!-- Vue 인스턴스 만들기 -->  
<script>  
var app = new Vue({  
  el: "#app",  
  data: {  
    message: "Hello"  
  }  
});  
</script>
```



모델이 변경되어도 반응하지 않음: 1회성 바인딩

```
...  
<div id="app" v-once>  
  {{ message }}  
</div>  
...
```

머스태시 문법이 타 환경과 충돌시 v-text 이용

```
...  
<div id="app" v-text="message">  
</div>  
...
```

Vue Template

: 데이터 바인딩

- ▶ v-bind : HTML 속성에 뷰 데이터 값을 연결할 때 사용

```
...  
<!-- Vue 렌더링 영역 -->  
<div id="app">  
    
</div>  
...  
<!-- Vue 인스턴스 만들기 -->  
<script>  
var app = new Vue({  
  el: "#app",  
  data: {  
    mainImageSrc: "Toystory.png"  
  }  
});  
</script>
```



Vue Template

: Directive (지시자)

- ▶ 표시 제어용 지시자 v-if vs v-show

```
...  
<div id="app">  
  <div v-if="showAll">Content</div>  
</div>  
...  
<!-- Vue 인스턴스 만들기 -->  
<script>  
var app = new Vue({  
  el: "#app",  
  data: {  
    showAll: true  
  }  
});  
</script>
```

v-if에 전달된 논리값이 true일 때만 화면에 렌더링

```
...  
<div id="app">  
  <div v-show="showAll">Content</div>  
</div>  
...  
<!-- Vue 인스턴스 만들기 -->  
<script>  
var app = new Vue({  
  el: "#app",  
  data: {  
    showAll: true  
  }  
});  
</script>
```

v-show에 전달된 논리값이 false이면 스타일 display 속성을 none으로 변경

v-show의 논리값 여부에 상관없이
DOM 상에는 노드로 존재

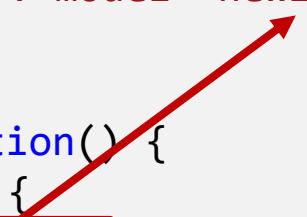


Vue Template

: Directive (지시자)

- ▶ v-model : Form 요소에 데이터 바인딩
 - ▶ 폼에 입력한 값을 뷰 인스턴스의 데이터와 즉시 동기화
 - ▶ 모델의 변경으로 폼에 입력값을 변경시킬 수도 있음

```
...  
var cartInput = {  
  template: `  
    <div>  
      <input v-model="newItem">  
    </div>  
  `,  
  data: function() {  
    return {  
      newItem: ""  
    }  
  },  
},  
...  
}
```



Vue Template

: Directive (지시자)

- ▶ v-on : 이벤트가 발생했을 때 지정한 이벤트 함수를 호출
 - ▶ 컴포넌트 내부의 함수를 호출하는 경우
 - ▶ 부모와의 통신을 위해 부모의 함수를 호출

```
...  
var cartInput = {  
  template: `  
    <div>  
      <button v-on:click="addItem">추가</button>  
    </div>  
  `,  
  methods: {  
    addItem: function() {  
      // ...  
    }  
  }  
}  
...
```

이벤트 처리 함수는 Vue Instance
혹은 컴포넌트 객체 내부의
methods 속성 내부에 정의

Vue Template

: Directive (지시자)

- ▶ v-for : Collection 객체를 루프를 돌면서 HTML 태그를 반복 출력
 - ▶ v-for로 루프를 돌릴 때, 키 값으로 사용될 내용을 v-key로 설정해 줄 것을 권장

```
...  
<ul>  
  <li v-for="item in items">{{ item }} </li>  
</ul>  
  
var app = new Vue({  
  el: "#app",  
  data: {  
    items: ["무", "배추", "쪽파", "고춧가루"]  
  }  
})  
...
```

Vue Template

: Directive (지시자)

▶ v-for + v-if (or v-show)

▶ 데이터(모델)의 상태에 따라 출력을 제어해야 할 경우는 v-for를 v-if(혹은 v-show) 디렉티브와 결합

```
...
<ul>
  <li v-for="item in items" v-if="item.buy">{{ item.name }}
</li>
</ul>

var app = new Vue({
  el: "#app",
  data: {
    items: [
      {name: "무", buy: false},
      {name: "배추", buy: false},
      {name: "쪽파", buy: true},
      {name: "고춧가루", buy: false}
    ]
  }
})
...
```

→ 출력되지 않음

실습 예제

: Vue-Cart

```
<cart-header></cart-header>
```

```
<shop-list></shop-list>
```

```
<cart-input></cart-input>
```

```
<bought-list></bought-list>
```

```
<cart-footer></cart-footer>
```

Vue Cart

살 물건들

- 무
- 배추
- 고춧가루

구매

구매

구매

추가

☒ 산 물건 보기

산 물건들

- 쪽파

취소

삭제

© Bit Academy

Vue HTTP 통신

: Using Axios

- ▶ Axios : Vue 커뮤니티에서 가장 널리 활용되는 HTTP 통신 라이브러리
 - ▶ Promise 기반의 API 형식이 다양하고 편리하게 제공
 - ▶ <https://github.com/axios/axios>

- ▶ Axios 라이브러리 로드 : HEAD 영역에 다음 스크립트를 포함

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

- ▶ Axios를 이용한 async 호출

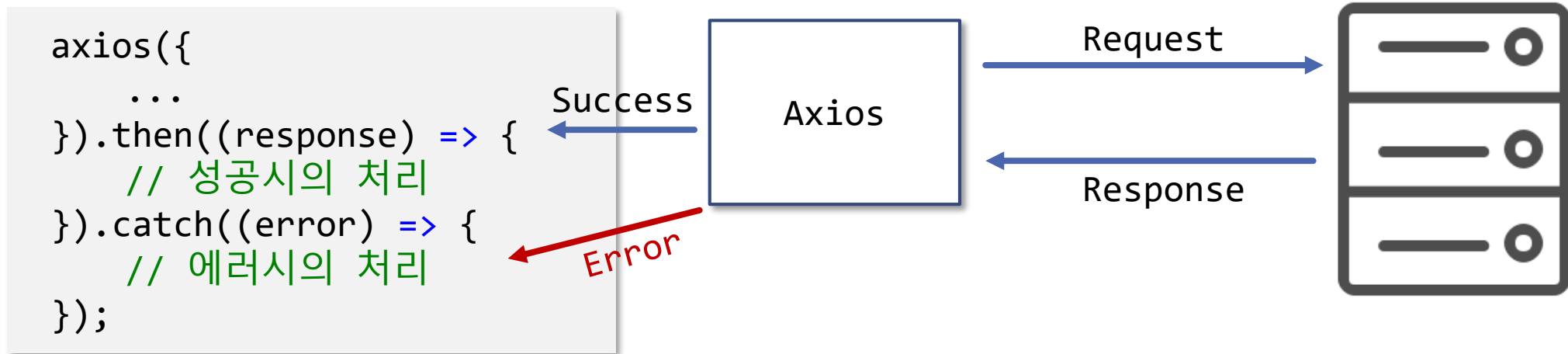
```
axios({
  method: 'get',
  url: 'URL 주소',
  data: {
    paramName1: 'Value1',
    paramName2: 'Value2'
  }
});
```

Vue HTTP 통신

: Using Axios

▶ Response의 처리

- ▶ Axios는 Promise 기반의 API로 직관적인 데이터 처리 흐름을 구성할 수 있음



- ▶ get 호출, post 호출을 위한 별도의 메서드가 있음

```
axios.get("URL주소");
```

Parameter가 없을 때

Parameter가 있을 때

```
axios.post('/user', {
  paramName1: 'Value1',
  paramName2: 'Value2'
});
```

Vue HTTP 통신

: Using Axios

- ▶ API 서버에서 데이터를 받아오는 가장 좋은 시점은 created

```
var app = new Vue({
  el: "#app",
  data: {
    movies: []
  },
  created: function () {
    console.log("Load Movies from API Server");
    var instance = this;
    axios.get("http://localhost:3000/movies")
      .then((response) => { // 요청이 성공했을 때
        console.log(response.data.movies);
        this.movies = response.data.movies;
      })
      .catch((error) => { // 요청 실패
        console.error(error);
      });
  }, ...
});
```

실습 예제

Pixar Movies



토이 스토리

감독: 존 라세터

개봉년도: 1995



벅스 라이프

감독: 존 라세터

개봉년도: 1998



토이 스토리 2

감독: 존 라세터

개봉년도: 1999



몬스터 주식회사

감독: 피트 닥터

개봉년도: 2001

Vue HTTP 통신

: Using Axios

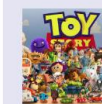
- ▶ Vue 렌더링 영역과 Model을 바인딩 해 봅니다

```
<div id="app">
  <div class="movie" v-for="movie in movies">
    
    <h2>{{ movie.title }}</h2>
    <p>감독: {{ movie.director }}</p>
    <p>개봉년도: {{ movie.year }}</p>
  </div>
</div>
```

```
{ "id":1,
  "title":"토이 스토리",
  "director":"존 라세터",
  "year":1995,
  "image":"toystory.jpeg"
}
```

```
{"movies": [
  { "id":1, "title":"토이 스토리",
    "director":"존 라세터",
    "year":1995, "image":"toystory.jpeg" },
  ...
]}
```

Pixar Movies



토이 스토리

감독: 존 라세터

개봉년도: 1995



벅스 라이프

감독: 존 라세터

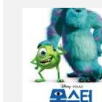
개봉년도: 1998



토이 스토리 2

감독: 존 라세터

개봉년도: 1999



몬스터 주식회사

감독: 피트 닥터

개봉년도: 2001

Vue Template

: computed 속성

▶ computed 속성

- ▶ 템플릿 내에 표현식을 넣으면 편리하기는 하지만 너무 많은 연산을 템플릿 안에서 하면 코드가 비대해지고 유지보수가 어려워짐
- ▶ computed 속성은
 - ▶ data의 변화에 따라 자동으로 다시 연산
 - ▶ 화면의 여러 곳에 값을 표시해야 한다면 computed 속성으로 지정하여 결과값을 미리 저장(캐싱)해 두고 필요할 때 불러 쓸 수 있다

```
Vue.component("cart-header", {  
  template: `

# {{ title.toUpperCase() }} </h1>` , data: function () { return { title: "Vue Cart" }; } });


```



```
Vue.component("cart-header", {  
  template: `

# {{ upperTitle }}</h1>`, data: function () { return { title: "Vue Cart" }; }, computed: { upperTitle: function() { return this.title.toUpperCase(); } } });


```

Vue Template

: watch 속성

▶ watch

- ▶ 데이터를 변경을 관찰하고 이에 반응하여 자동으로 특정 로직을 수행
- ▶ Vue.js는 선언적 프로그래밍 형식이지만, watch 속성은 감시할 데이터를 지정하고 데이터가 바뀌면 특정 함수를 실행
 - ▶ 명령형 프로그래밍 방식으로 작동
 - ▶ 특별한 상황이 아니라면 watch 보다 computed 속성을 사용할 것을 권장

```
var cartInput = {  
  template: `  
    <div>  
      <p>{{ message }}</p>  
      <input v-model="newItem">  
      <button v-on:click="addItem">추가</button>  
    </div>  
  `,  
  data: function() {  
    return {  
      newItem: "",  
      message: ""  
    }  
  },  
  watch: {  
    newItem: function(item) {  
      console.log(item);  
      if (item.trim().length > 0) {  
        this.message = "상품 등록 가능";  
      } else {  
        this.message = "상품을 입력하세요";  
      }  
    }  
  }  
}
```

Vue Project

: How-To

- ▶ 기존 방식으로 Vue 코드 작성시 마주하게 되는 한계점
 - ▶ template 속성에서 작성된 HTML 코드를 바로 분석해서 미리 예상하기 어려움
 - ▶ 개별 인스턴스 안쪽에 템플릿과 이벤트 함수들이 포함되어 어느 정도 구조화하는 할 수 있으나, CSS 등을 한 컴포넌트 내부에서 관리하기 쉽지 않다
- ▶ Single File Component 체계
 - ▶ 하나의 단일 파일 내에 다음의 내용을 하나로 묶어 관리
 - ▶ 템플릿
 - ▶ 스크립트
 - ▶ 객체 속성
 - ▶ 스타일
 - ▶ 개별 컴포넌트는 .vue 확장자로 작성

```
<template>
<!-- HTML Tag -->
</template>

<script>
export default {
  // JavaScript
}
</script>

<style>
/* StyleSheet */
</style>
```

Single File Component 체계

Vue Project

: How-To

▶ Vue CLI

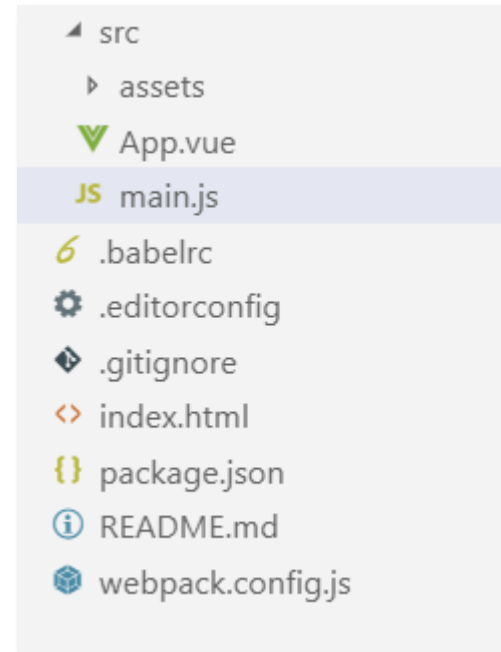
- ▶ 싱글 파일 컴포넌트 체계를 이용하기 위해 .vue 파일을 웹브라우저가 인식할 수 있는 형태의 파일로 변환이 필요 -> Webpack, Browserify 등
- ▶ Vue CLI(Command Line Interface)는 모듈 번들링 작업 등 편하게 프로젝트를 구성할 수 있는 명령행에서 실행되는 도구(공식)
- ▶ <https://cli.vuejs.org/guide/>

▶ Vue CLI의 설치 (Node.js 필요)

```
npm install vue-cli --global
```

▶ Webpack을 이용한 기본 Vue 템플릿으로 프로젝트 생성

```
vue init webpack-simple
```



Vue Project

: How-To

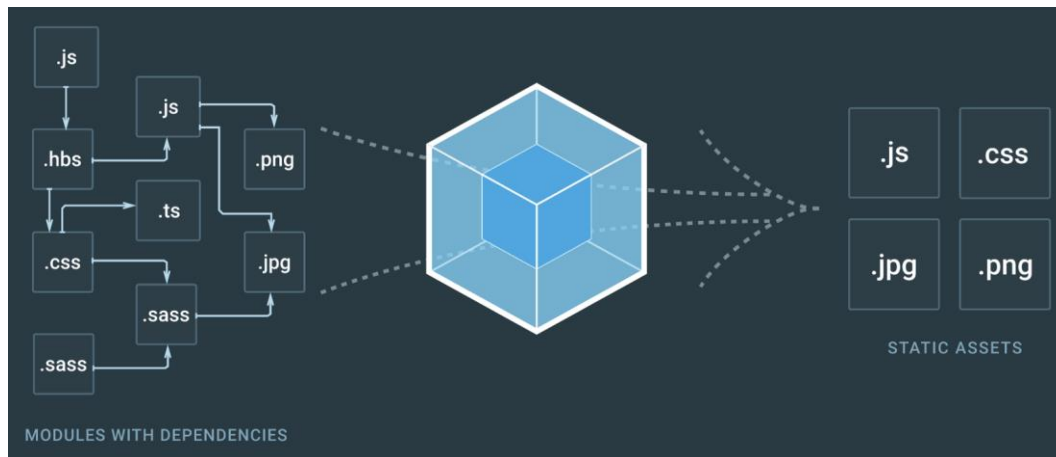
- ▶ 프로젝트에 등록되어 있는 의존성 라이브러리를 설치

```
npm install
```

- ▶ package.json 내의 dependencies, devDependencies 라이브러리를 찾아 일괄 설치

- ▶ 프로젝트 실행

```
npm run dev
```



Welcome to Your Vue.js App

Essential Links

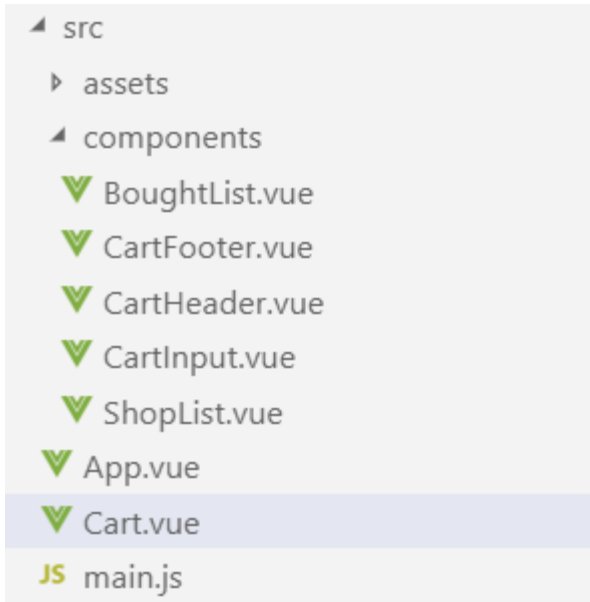
[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#)

Ecosystem

[vue-router](#) [vuex](#) [vue-loader](#) [awesome-vue](#)

Vue Project

: 실습 예제



CartHeader.vue

ShopList.vue

CartInput.vue

BoughtList.vue

CartFooter.vue

Vue Cart

전체 목록

무
배추
쪽파
고춧가루

살 물건들

무	구매
배추	구매
고춧가루	구매

☒ 산 물건 보기

산 물건들

쪽파	<input type="button" value="취소"/> <input type="button" value="삭제"/>
----	---

© Bit Academy

Vue Router

▶ Routing

- ▶ 웹 페이지 간의 전환 방식을 의미
- ▶ 전통적인 방식과는 달리, SPA(Single Page Application) 방식 응용프로그램은 웹 페이지 전환을 브라우저 내에서 직접 실행
- ▶ Vue.js는 Angular, React와 달리, 라우터가 기본적으로 로드되지 않으나 별도 스크립트 파일로 vue-router(공식)를 로드하여 뷰 라우팅 기능을 구현할 수 있다
 - ▶ 공식 가이드 : <https://router.vuejs.org/kr/>
 - ▶ 공식 저장소 : <https://github.com/vuejs/vue-router>

▶ 뷰 라우터의 로드

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


Vue Router

- ▶ `<router-view />`
 - ▶ 변경되는 URL에 따라 적합한 컴포넌트를 표시해 주는 영역
- ▶ `<router-link />`
 - ▶ 페이지 이동 태그. 기본적으로 a 태그로 바인딩되며 to 속성에 라우팅 url을 정의
- ▶ VueRouter 객체
 - ▶ 라우팅 정보를 담고 있는 객체

```
<!-- Vue Rendering 영역 -->
<div id="app">
  <h1>Routing</h1>
  <router-view></router-view>
  <ul>
    <li>
      <router-link to="/">Main</router-link>
    </li>
    <li>
      <router-link to="/Sub">Sub</router-link>
    </li>
  </ul>
</div>
```

```
var main = { template: "<div>Main</div>" };
var sub = { template: "<div>Sub</div>" };
var router = new VueRouter({
  routes: [
    { path: '/', component: main },
    { path: '/sub', component: sub }
  ]
});
```

Router 등록

```
new Vue({
  router: router
}).$mount("#app");
```

/sub URL이 호출되면 sub 컴포넌트를 표시

```
{ path: '/sub', component: sub }
```

Vue Router

: 동적 라우팅

- ▶ router-link의 to 속성은 정적 페이지 전송에는 적합하나 파라미터를 전송하기에는 적합하지 않음
 - ▶ v-bind 지시자를 사용하면 라우팅 대상 페이지에 전송할 파라미터를 손쉽게 작성할 수 있음
 - ▶ 전송된 파라미터는 컴포넌트 내부에서 `$route.params` 컬렉션 내부에서 확인 가능

```
var passParam = {  
  template: "<div><h2>PassParam: {{ param }}</h2></div>",  
  computed: {  
    param: function() {  
      return this.$route.params.message;  
    }  
  }  
}
```

```
<router-link  
  v-bind:to="'/passparam/' + param">  
  파라미터 전송  
</router-link>
```

```
var router = new VueRouter({  
  routes: [  
    { path: '/passparam/:message',  
      component: passParam }  
  ]  
});
```

```
new Vue({  
  router: router,  
  data: {  
    param: "파라미터 전송"  
  }  
}).$mount("#app");
```

Vue Router

: 네스티드 라우터

- ▶ 실제 개발 상황에서는 컴포넌트 내부에 서브 컴포넌트들이 포함되는 경우가 많음
 - ▶ 라우터에 라우팅 정보를 포함할 때, 내부에 children 속성으로 중첩된 라우터를 포함할 수 있음

```
var router = new VueRouter({
  routes: [
    ...
    { path: '/user', component: user,
      children: [{
        path: 'profile',
        component: {
          template: "<div><h3>User Profile</h3></div>"
        }
      ]
    }
  ]
});
```

/user/profile URL은 이쪽으로 매칭

```
var user = {
  template: `<div>
    <h2>User</h2>
    <router-view></router-view>
  </div>`
}
```

```
new Vue({
  router: router,
}).$mount("#app");
```

Vue Router

: 네임드 뷰 - 이름 기반 라우팅

▶ 네임드 뷰

- ▶ 특정 페이지로 이동했을 때, 여러 개의 컴포넌트를 각기 다른 위치에 동시에 표시하기 위한 라우팅
- ▶ 네스티드 라우팅 방식이 부모-자식 컴포넌트를 동시에 변경하는 것이라면
이름 기반의 라우팅은 상위-하위 관계에 있지 않은 여러 컴포넌트도 동시에 변경 가능함
- ▶ <router-view>에 특별히 이름을 지정하지 않으면 기본 이름은 default

default

```
<router-view name="header"></router-view>  
<router-view></router-view>  
<router-view name="footer"></router-view>
```

```
new Vue({  
  router: router,  
}).$mount("#app");
```

```
var router = new VueRouter({  
  routes: [  
    ...  
    // 이름 기반 라우팅  
    { path: '/named', components: {  
      header: headerComp,  
      default: contentComp,  
      footer: footerComp  
    } }  
  ]  
});
```