Vue.js

Vue Fundamental

뷰 — 기초 학습

프로그래시브 JavaScript 프레임워크

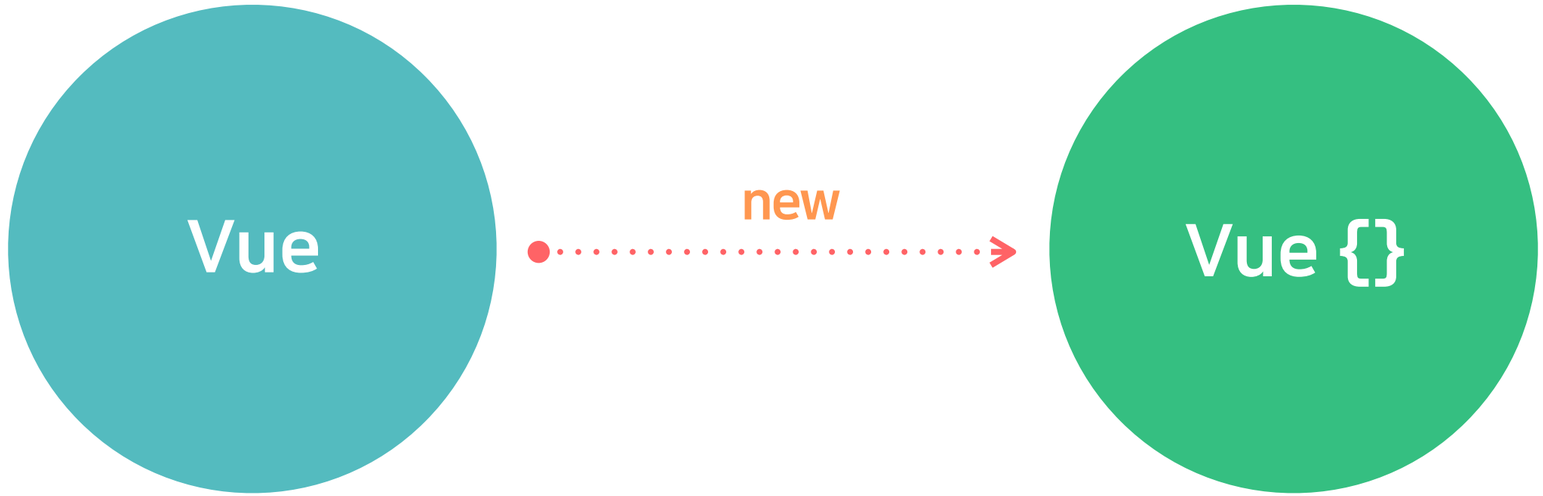
Vue JS Framework

Progressive

뷰 — 인스턴스

Vue Instance

Progressive



뷰 생성자 (Vue Class)

뷰 객체 (Vue Instance)

Progressive

생성자

부트스트랩 bootstrap
[전산] 컴퓨터를 사용하기 위해 시스템을 시동하는 작업

모든 Vue vm은 `Vue` 생성자 함수로 **root Vue 인스턴스**를 생성하여 부트스트래핑됩니다.

```
JS

var vm = new Vue({
  // 옵션
})
```

엄격히 MVVM 패턴과 관련이 없지만 Vue의 디자인은 부분적으로 그것에 영감을 받았습니다. 컨벤션으로, Vue 인스턴스를 참조하기 위해 종종 변수 `vm` (ViewModel의 약자)을 사용합니다.

Vue 인스턴스를 인스턴스화 할 때는 데이터, 템플릿, 마운트할 엘리먼트, 메소드, 라이프사이클 콜백 등의 옵션을 포함 할 수있는 **options 객체**를 전달 해야합니다. 전체 옵션 목록은 API reference에서 찾을 수 있습니다.

{ … }

https://kr.vuejs.org/v2/api/#인스턴스-속성

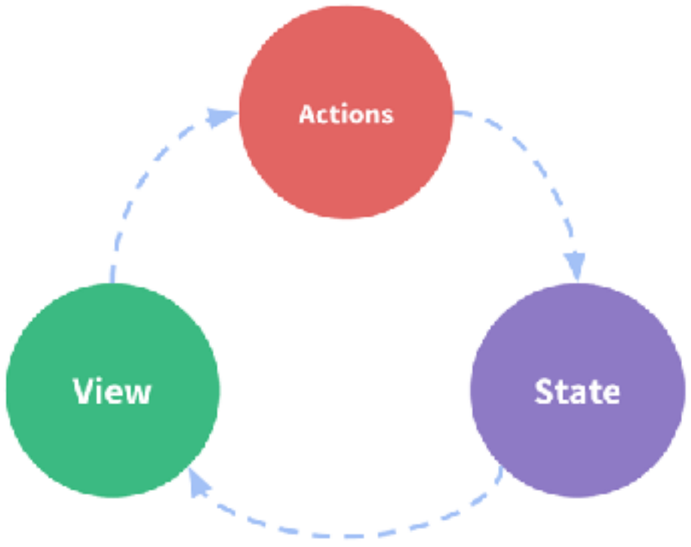
Vue {}

뷰 객체 (Vue Instance)

Progressive

뷰 인스턴스 (Vue Object)

Vue 생성자를 통해 생성된 객체(인스턴스)는 **상태(State)**를 통해 **뷰(View)**를 그리고, **액션(Actions)**을 처리하여 상태 변경을 수행, 뷰를 업데이트 한다.




```
new Vue({
  // 상태
  data () {
    return {
      count: 0
    }
  },
  // 뷰
  template: `
    <div>{{ count }}</div>
  `,
  // 액션
  methods: {
    increment () {
      this.count++
    }
  }
})
```

앱을 구동하는
데이터 (상태, State)

상태의 선언적 매핑
(템플릿, 디렉티브)

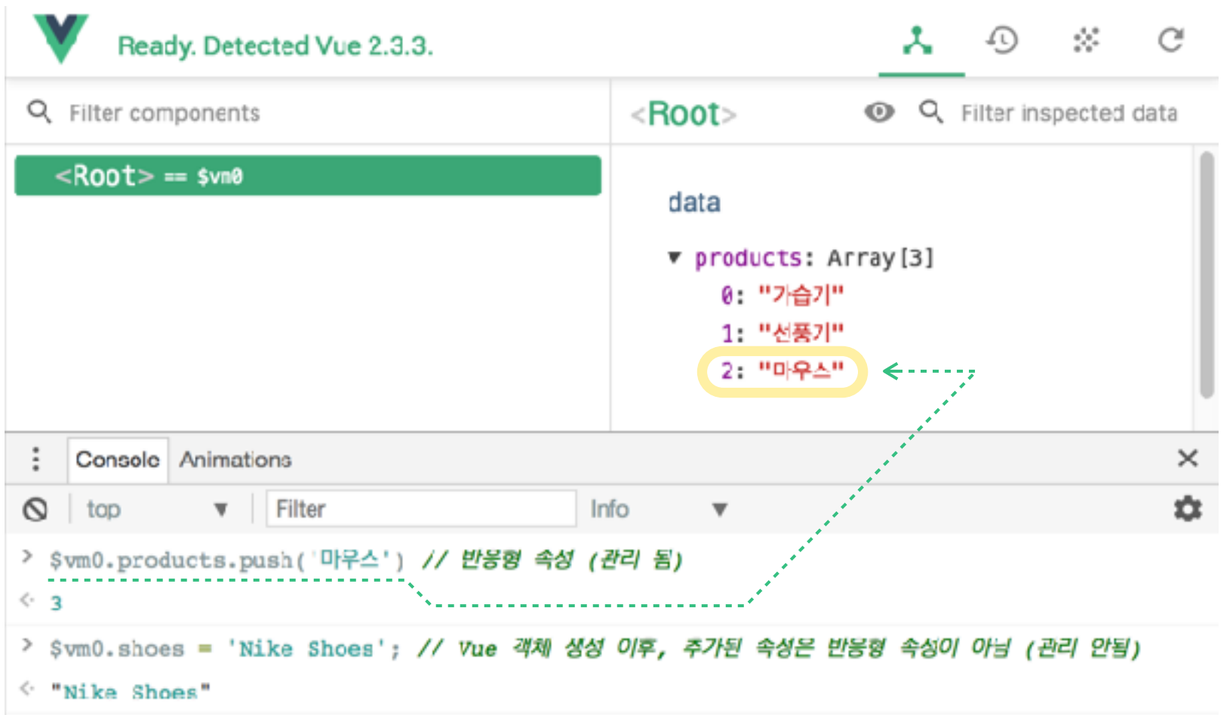
뷰 를 통해 사용자 입력에
반응하여 상태를 바꾸는 메서드

각 Vue 인스턴스는 `data` 객체에 있는 모든 속성을 **프록시** 처리 합니다.

```JS
var data = { a: 1 }
var vm = new Vue({
  data: data
})

vm.a === data.a // -> true

// 속성 설정은 원본 데이터에도 영향을 미칩니다.
vm.a = 2
data.a // -> 2

// ... 당연하게도 ← 값 참조 (Pass by Reference)
data.a = 3
vm.a // -> 3
```

이러한 프록시 속성은 **반응형** 입니다. 인스턴스를 작성한 후 인스턴스에 새 특성을 첨부하면 뷰 업데이트가 트리거되지 않습니다. 우리는 나중에 반응 시스템에 대해 자세히 논의 할 것입니다.

생성 과정에서 설정된 속성과 달리,
생성 이후 추가된 속성은 반응형이 아닙니다.



Progressive

Vue 인스턴스는 데이터 속성 외에도 유용한 인스턴스 속성 및 메소드를 제공합니다. 이 프로퍼티들과 메소드들은 `$` 접두사로 프록시 데이터 속성과 구별됩니다. 예:

```
> for (var prop in $vm0) {
  //    if ( $vm0.hasOwnProperty(prop) ) {
          if (typeof $vm0[prop] !== 'function' && prop.indexOf('_') !== 0)
              console.log('Vue 인스턴스 속성:', prop);
          }
  }
```

```
Vue 인스턴스 속성: $options
Vue 인스턴스 속성: $parent
Vue 인스턴스 속성: $root
Vue 인스턴스 속성: $children
Vue 인스턴스 속성: $refs
Vue 인스턴스 속성: $vnode
Vue 인스턴스 속성: $slots
Vue 인스턴스 속성: $scopedSlots
Vue 인스턴스 속성: products
Vue 인스턴스 속성: $el
Vue 인스턴스 속성: shoes
```

Vue 인스턴스는 데이터 속성 외에도 유용한 인스턴스 속성 및 메소드를 제공합니다. 이 프로퍼티들과 메소드들은 `$` 접두사로 프록시 데이터 속성과 구별됩니다. 예:

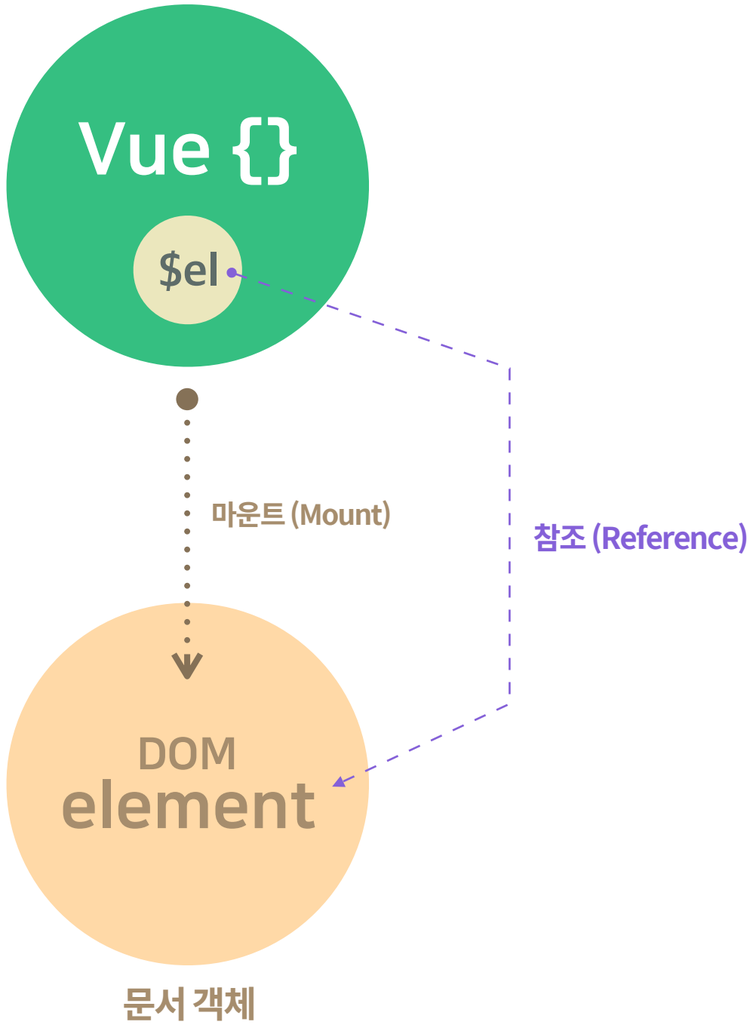
```
> for (var prop in $vm0) {
  //    if ( $vm0.hasOwnProperty(prop) ) {
          if (typeof $vm0[prop] === 'function' && prop.indexOf('_') !== 0)
              console.log('Vue 인스턴스 메서드:', prop);
          }
  //    }
  }
  Vue 인스턴스 메서드: $createElement
  Vue 인스턴스 메서드: $set
  Vue 인스턴스 메서드: $delete
  Vue 인스턴스 메서드: $watch
  Vue 인스턴스 메서드: $on
  Vue 인스턴스 메서드: $once
  Vue 인스턴스 메서드: $off
  Vue 인스턴스 메서드: $emit
  Vue 인스턴스 메서드: $forceUpdate
  Vue 인스턴스 메서드: $destroy
  Vue 인스턴스 메서드: $nextTick
  Vue 인스턴스 메서드: $mount
```

vm.$el

- 타입: `HTMLElement`
- 읽기 전용
- 상세:

  Vue 인스턴스가 관리하는 루트 DOM 엘리먼트 입니다.

- Source

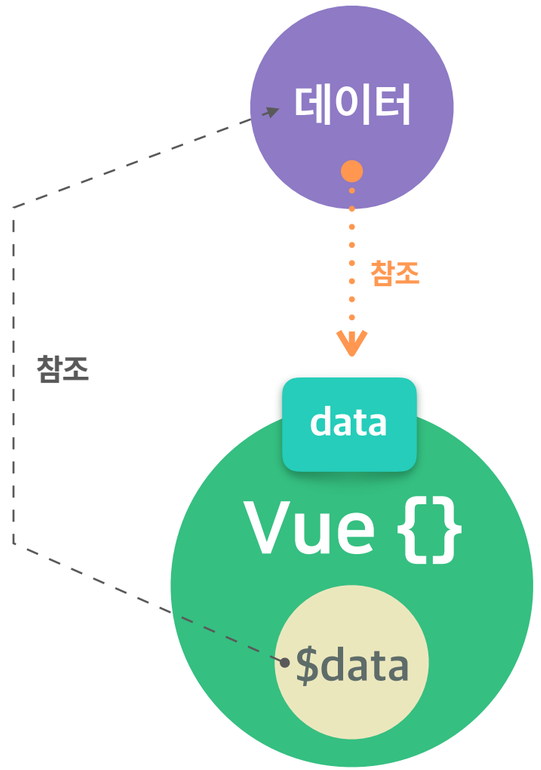


vm.$data

- 타입: `Object`
- 상세:

  Vue 인스턴스가 관찰하는 데이터 객체입니다. Vue 인스턴스는 데이터 객체의 속성에 대한 엑세스를 프록시 합니다.

- 참고: 옵션 - data
- Source



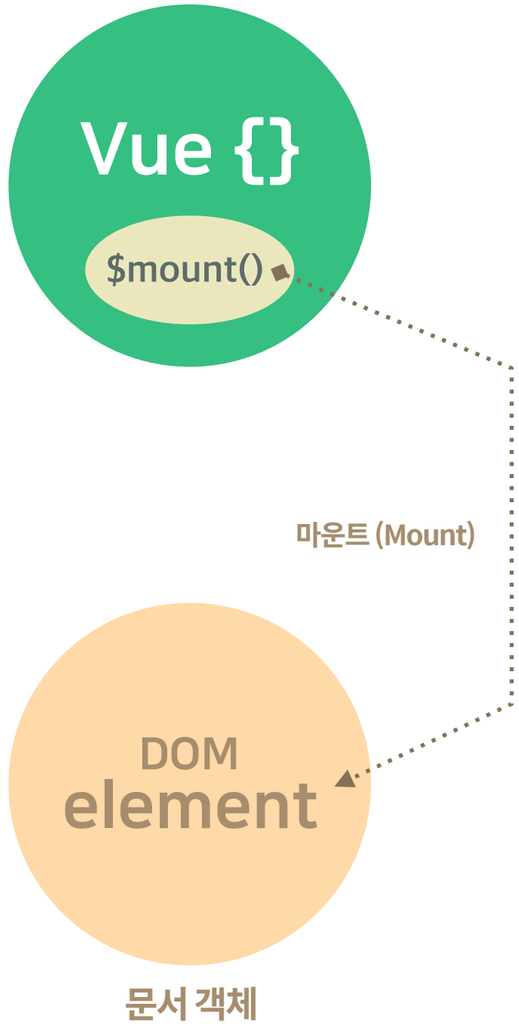
vm.$mount([elementOrSelector])

- 전달인자:
  - `{Element | string} [elementOrSelector]`
  - `{boolean} [hydrating]`
- 반환 값: `vm` - 인스턴스 그 자체
- 사용방법:

Vue 인스턴스가 인스턴스화 할 때 `el` 옵션이 없으면 연결된 DOM 엘리먼트 없이 “unmounted” 상태가 됩니다. `vm.$mount()` 는 unmounted 된 Vue인스턴스의 마운트를 수동으로 시작하는데 사용할 수 있습니다.

`elementOrSelector` 인자가 제공되지 않으면, 템플릿은 문서가 아닌 엘리먼트로 렌더링 될 것이므로 DOM API를 사용하여 문서에 직접 삽입해야 합니다.

이 메소드는 다른 인스턴스 메소드를 체이닝 할 수 있도록 인스턴스 그 자체를 반환 합니다.



템플릿 Template & 디렉티브(지시자) Directives

Progressive

Progressive

템플릿 문법

Vue.js는 렌더링 된 DOM을 기본 Vue 인스턴스의 데이터에 선언적으로 바인딩 할 수있는 HTML 기반 템플릿 구문을 사용합니다. 모든 Vue.js 템플릿은 스펙을 호환하는 브라우저 및 HTML 파서로 구문 분석 할 수있는 유효한 HTML입니다.

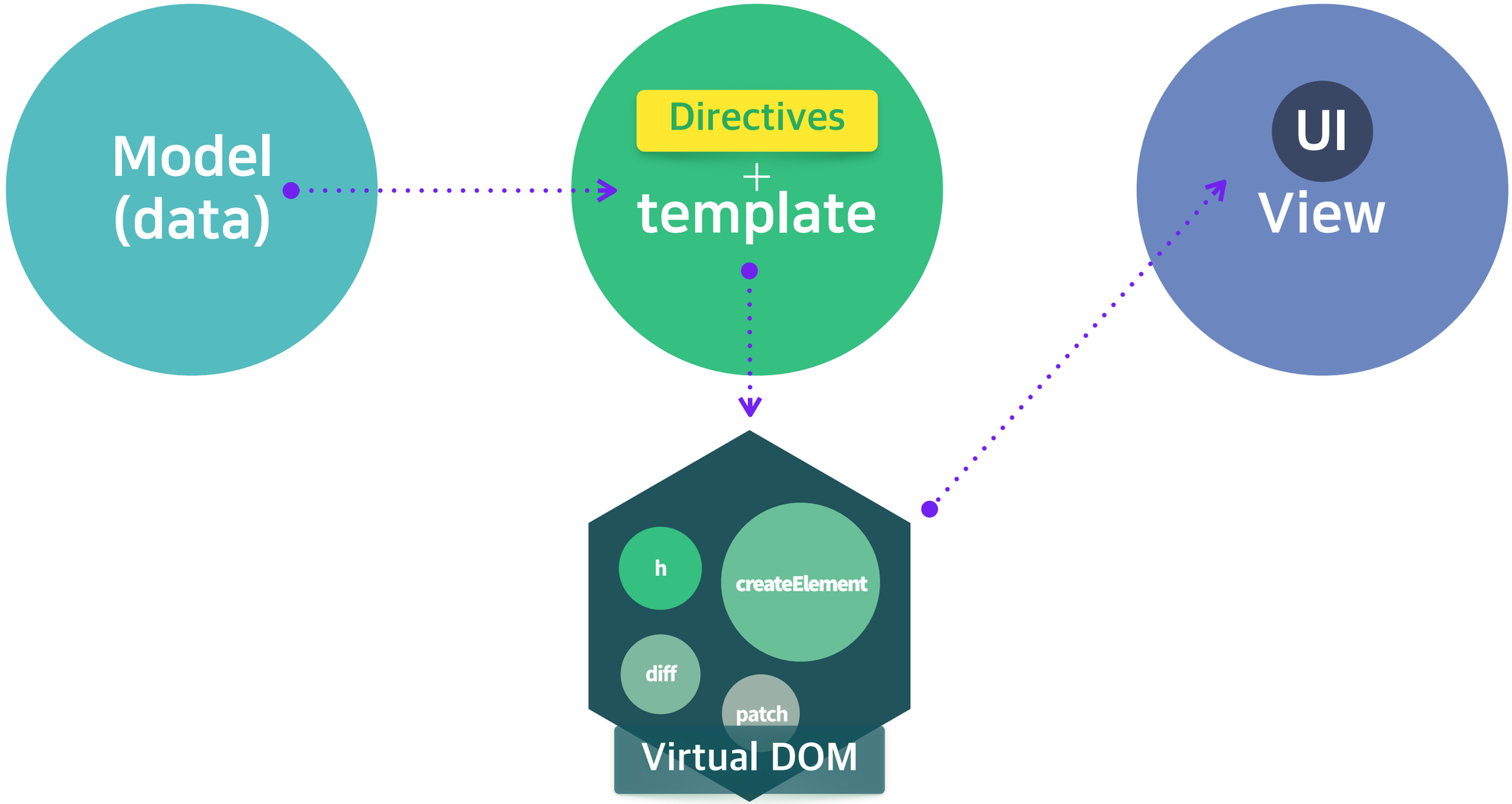
내부적으로 Vue는 템플릿을 가상 DOM 렌더링 함수로 컴파일 합니다. 반응형 시스템과 결합된 Vue는 앱 상태가 변경 될 때 최소한으로 DOM을 조작하고 다시 적용할 수 있는 최소한의 컴포넌트를 지능적으로 파악할 수 있습니다.

가상 DOM 개념에 익숙하고 JavaScript의 기본 기능을 선호하는 경우 템플릿 대신 렌더링 함수를 직접 작성할 수 있으며 선택사항으로 JSX를 지원합니다.

https://kr.vuejs.org/v2/guide/syntax.html

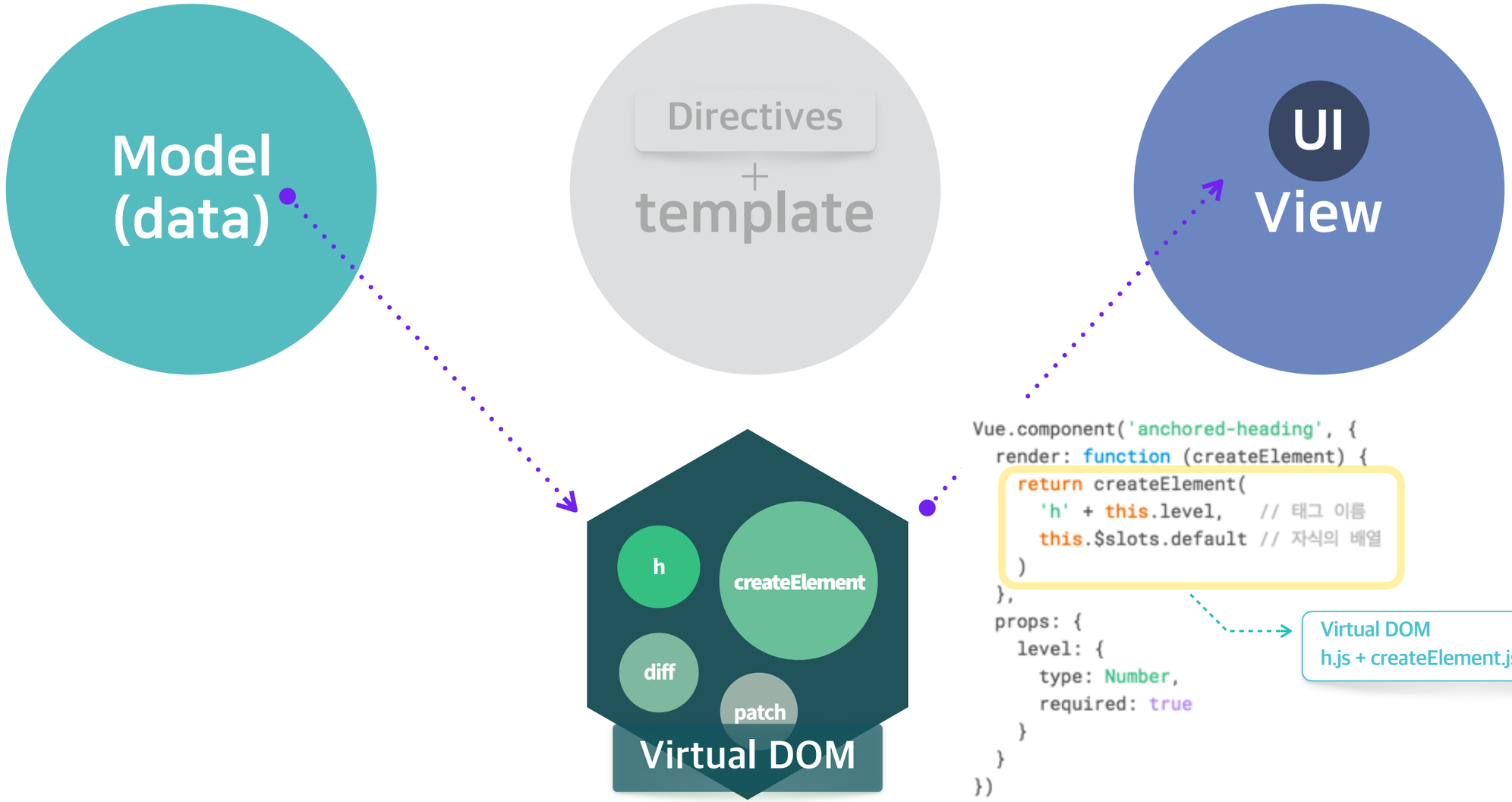
Vue 템플릿 컴파일 구조

Vue는 대다수의 경우, 템플릿을 사용하여
HTML을 작성할 것을 권장합니다.



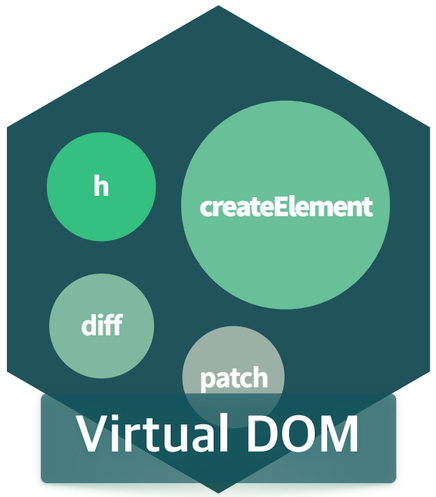
Progressive

Vue 가상 DOM 렌더링 함수



Progressive

Vue 가상 DOM 렌더링 함수



h

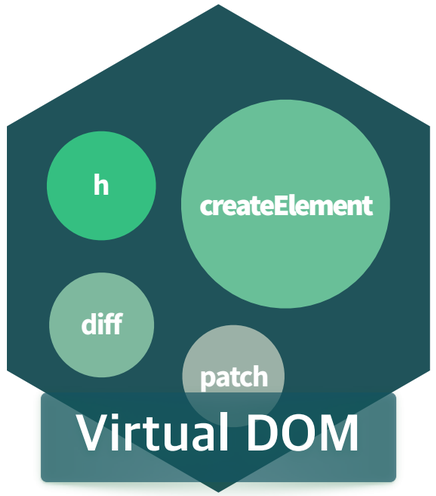
```
// @returns {VNode}
createElement(
  // {String | Object | Function}
  // HTML 태그 이름, 컴포넌트 옵션 또는 함수 중
  // 하나를 반환하는 함수입니다. 필수 사항.
  'div',

  // {Object}
  // 템플릿에서 사용할 속성에 해당하는 데이터 객체입니다
  // 데이터 객체입니다. 선택 사항.
  {
    // (아래 다음 섹션에 자세히 설명되어 있습니다.)
  },

  // {String | Array}
  // VNode 자식들. 선택 사항.
  [
    createElement('h1', 'hello world'),
    createElement(MyComponent, {
      props: {
        someProp: 'foo'
      }
    }),
    'bar'
  ]
)
```

HTML 요소 | 컴포넌트

속성(데이터) 객체

VNode | VText

Vue 가상 DOM 렌더링 함수




```
import AnchoredHeading from './AnchoredHeading.vue'

new Vue({
  el: '#demo',
  render (h) {
    return (
      <AnchoredHeading level={1}>
        <span>Hello</span> world!
      </AnchoredHeading>
    )
  }
})
```

h

JSX

! `createElement` 를 `h` 별칭을 이용하는 것은 Vue 생태계에서 볼 수 있는 공통된 관습이며 실제로 JSX에 필요합니다. 사용하는 범위에서 `h` 를 사용할 수 없다면, 앱은 오류를 발생시킵니다.

JavaScript 표현식 사용하기

지금까지 템플릿의 간단한 속성 키에만 바인딩했습니다. 그러나 실제로 Vue.js는 모든 데이터 바인딩 내에서 JavaScript 표현식의 모든 기능을 지원합니다.

HTML

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>
```

식(Expression) 만 가능

이 표현식은 Vue 인스턴스 데이터 범위 내에서 JavaScript로 계산됩니다. 한가지 제한사항은 각 바인딩에 **하나의 단일 표현식** 만 포함될 수 있으므로 아래처럼 작성하면 **안됩니다**

HTML

```
<!-- 아래는 구문입니다, 표현식이 아닙니다. -->
{{ var a = 1 }}

<!-- 조건문은 작동하지 않습니다. 삼항 연산자를 사용해야 합니다. -->
{{ if (ok) { return message } }}
```

문(Statement)은 불가능

템플릿 표현식은 샌드박스 처리되며 `Math` 와 `Date` 같은 전역으로 사용 가능한 것에만 접근할 수 있습니다. 템플릿 표현식에서 사용자 정의 전역에 액세스 하지 마십시오.

사용자 정의 변수, 함수는 안됨

https://kr.vuejs.org/v2/guide/syntax.html#JavaScript-표현식-사용하기

Vue, X 템플릿

```vue
new Vue({
  // 상태
  data () {
    return {
      count: 0
    }
  },
  // 뷰
  template: `
    <div>{{ count }}</div>
  `,
  // 액션
  methods: {
    increment () {
      this.count++
    }
  }
})
```


X-Templates

템플리트를 정의하는 또 다른 방법은 `text/x-template` 유형의 스크립트 엘리먼트 내부에 ID로 템플릿을 참조하는 것입니다. 예:

id 속성만 가능

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

<script type="text/x-template" id="hello-vue-template">
  <p>{{ greeting_message }}</p>
</script>
```

템플릿 내부는
단 하나의 root 요소만 인정합니다.

```vue
new Vue({
  el: '#app',
  data: {
    greeting_message: 'Hello Vue'
  },
  template: '#hello-vue-template'
});
```

이 기능은 큰 템플릿이나 매우 작은 응용 프로그램의 데모에는 유용 할 수 있지만 템플릿을 나머지 컴포넌트 정의와 분리하기 때문에 피해야합니다.

디렉티브

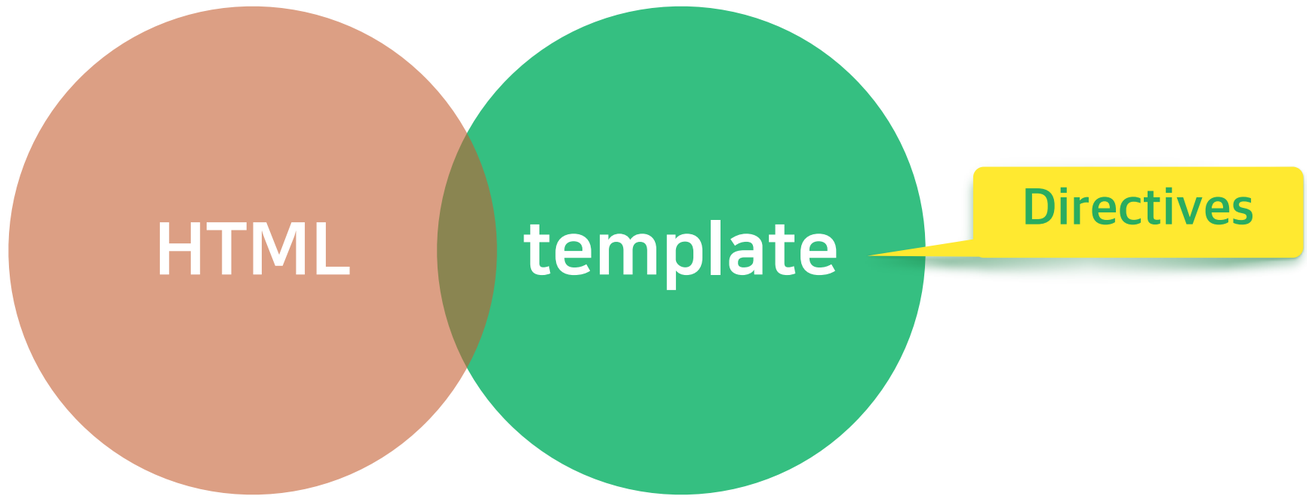
디렉티브는 `v-` 접두사가 있는 특수 속성입니다. 디렉티브 속성 값은 **단일 JavaScript 표현식** 이 됩니다. (나중에 설명할 `v-for` 는 예외입니다.) 디렉티브의 역할은 표현식의 값이 변경될 때 사이드이펙트를 반응적으로 DOM에 적용하는 것 입니다. 아래 예제에서 살펴보겠습니다.

문자가 아닌, JavaScript 식(Expression) 입니다.

```HTML
<p v-if="seen">이제 나를 볼 수 있어요</p>
```

여기서, `v-if` 디렉티브는 `seen` 표현의 진실성에 기반하여 `<p>` 엘리먼트를 제거 또는 삽입합니다.

https://kr.vuejs.org/v2/guide/syntax.html#디렉티브



v-text

v-html

v-pre

v-cloak

v-once

v-for

v-if

v-else

v-else-if

v-show

v-on

v-bind

v-model

Progressive

https://kr.vuejs.org/v2/guide/syntax.html#디렉티브

Vue Directives

뷰 — 디렉티브

Progressive

v-text

v-html

v-pre

v-cloak

v-once

v-for

v-if

v-else

v-else-if

v-show

v-on

v-bind

v-model

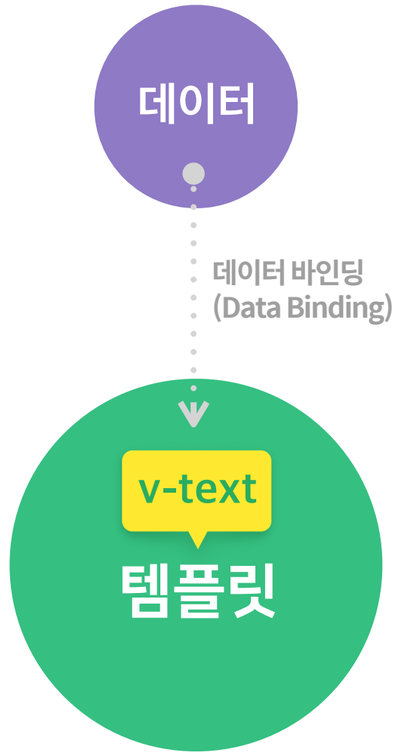
v-text

- 예상됨: `string`
- 상세:

  엘리먼트의 `textContent` 를 업데이트 합니다. `textContent` 의 일부를 갱신해야 하면 `{{ Mustache }}` 를 사용해야 합니다.

- 예제:

```HTML
<span v-text="msg"></span>
<!-- 같습니다 -->
<span>{{msg}}</span>
```



Progressive

https://kr.vuejs.org/v2/api/#v-text

보간법(Interpolation)

문자열

데이터 바인딩의 가장 기본 형태는 “Mustache” 구문(이중 중괄호)을 사용한 텍스트 보간입니다.

```HTML
<span>메시지: {{ msg }}</span>
```

Mustache 태그는 해당 데이터 객체의 `msg` 속성 값으로 대체됩니다. 또한 데이터 객체의 `msg` 속성이 변경될 때 마다 갱신됩니다.

https://kr.vuejs.org/v2/guide/syntax.html#보간법-Interpolation

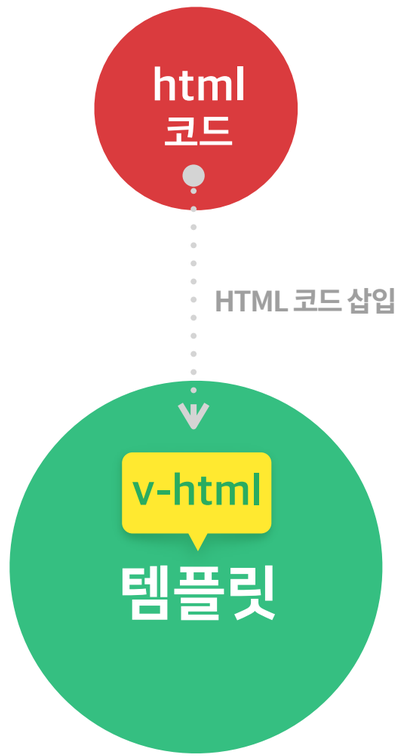
v-html

- **예상됨:** `string`
- **상세:**

  엘리먼트의 `innerHTML` 을 업데이트 합니다. **내용은 일반 HTML으로 삽입되므로 Vue 템플릿으로 컴파일 되지 않습니다.** `v-html` 을 사용하여 템플릿을 작성하려는 경우 컴포넌트를 사용하여 솔루션을 다시 생각해 보십시오.

- **예제:**

```HTML
<div v-html="html"></div>
```



원시 HTML

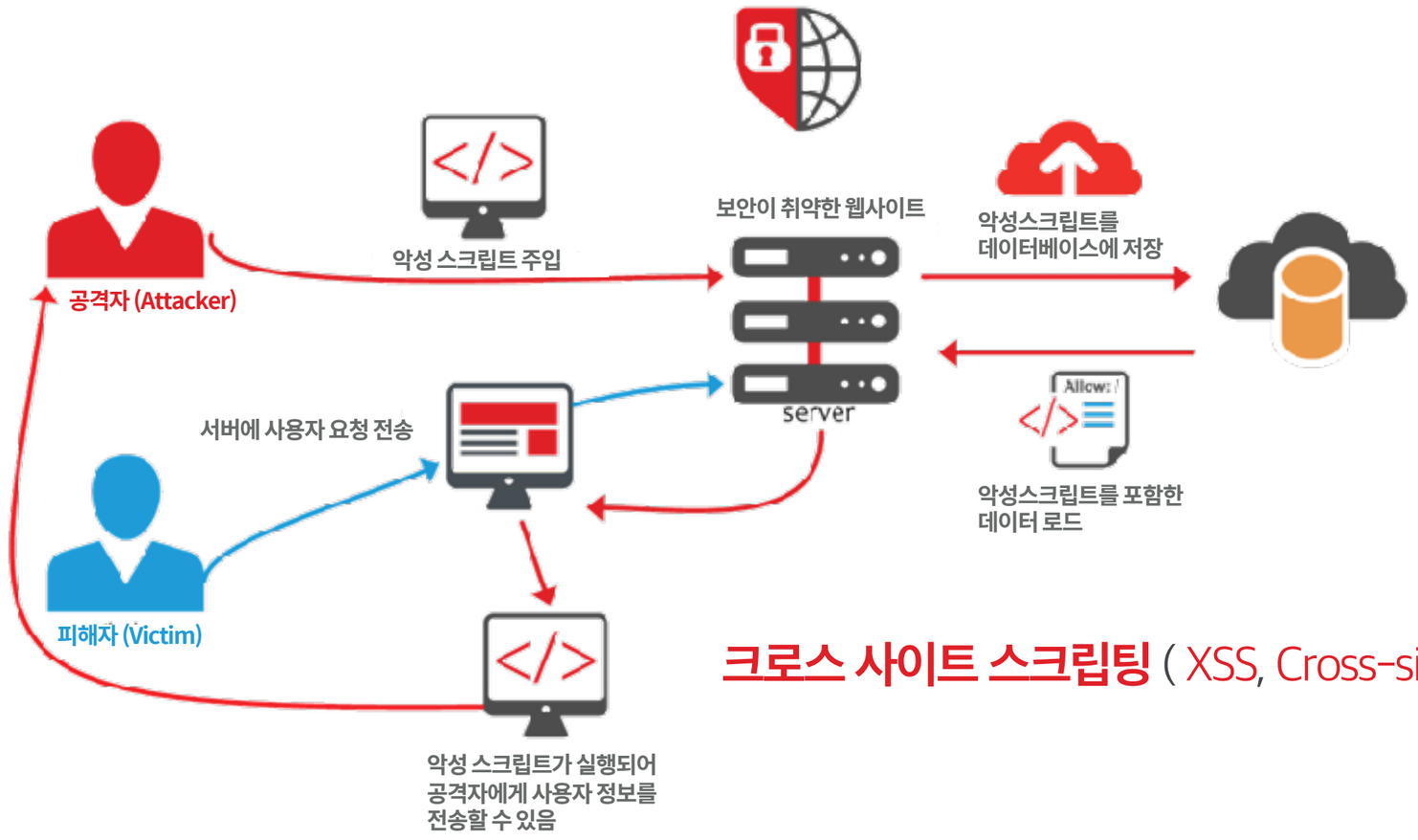
이중 중괄호(mustaches)는 HTML이 아닌 일반 텍스트로 데이터를 해석합니다. 실제 HTML을 출력하려면 `v-html` 디렉티브를 사용해야 합니다.

```HTML
<div v-html="rawHtml"></div>
```

컨텐츠는 일반 HTML 형식으로 삽입됩니다. 데이터 바인딩은 무시됩니다. Vue는 문자열 기반 템플릿 엔진이 아니기 때문에 `v-html` 을 사용하여 템플릿 조각을 작성할 수 없습니다. 대신 컴포넌트는 UI 재사용 및 구성을 위한 기본 단위로 사용해야 합니다.

> ! 웹사이트에서 임의의 HTML을 동적으로 렌더링하려면 XSS 취약점으로 쉽게 이어질 수 있으므로 매우 위험할 가능성이 있습니다. 신뢰할 수 있는 콘텐츠에서는 HTML 보간만 사용하고 사용자가 제공한 콘텐츠에서는 **절대** 사용하면 안됩니다.

https://kr.vuejs.org/v2/guide/syntax.html#보간법-Interpolation



크로스 사이트 스크립팅 (XSS, Cross-site scripting)

https://ko.wikipedia.org/wiki/사이트_간_스크립팅

v-pre

- **표현식이 필요하지 않습니다**
- **사용방법:**

  이 엘리먼트와 모든 자식 엘리먼트에 대한 컴파일을 건너 뜁니다. 원시 mustache 태그를 표시하는데 사용할 수 있습니다. 디렉티브가 없는 많은 수의 노드를 뛰어 넘으면 컴파일 속도가 빨라집니다.

- **예제:**

```HTML
<span v-pre>{{ 이 부분은 컴파일 되지 않습니다 }}</span>
```

https://kr.vuejs.org/v2/api/#v-pre

v-text
v-html
v-pre
v-cloak
v-once
v-for
v-if
v-else
v-else-if
v-show
v-on
v-bind
v-model

v-cloak

- 표현식이 필요하지 않습니다
- 사용방법:

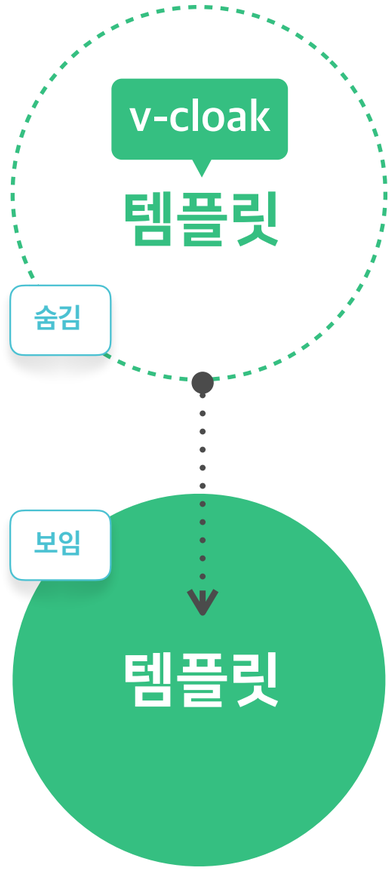
  이 디렉티브는 Vue 인스턴스가 컴파일을 완료할 때까지 엘리먼트에 남아있습니다.
  `[v-cloak] { display: none }` 와 같은 CSS규칙과 함께 이 디렉티브는 Vue인스턴스가 준비될 때까지 컴파일되지 않은 mustache 바인딩을 숨기는데 사용할 수 있습니다.

- 예제:

CSS
```
[v-cloak] {
  display: none;
}
```

HTML
```
<div v-cloak>
  {{ message }}
</div>
```

`<div>` 는 컴파일이 끝날 때까지 보이지 않습니다.



Progressive

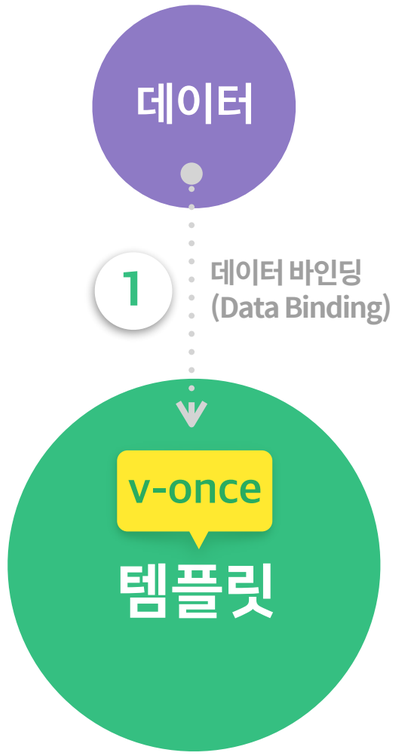
https://kr.vuejs.org/v2/api/#v-cloak

v-once

- 표현식이 필요하지 않습니다
- 상세:

  엘리먼트 및 컴포넌트를 **한번만** 렌더링 합니다. 후속 렌더링에서 엘리먼트 / 컴포넌트와 모든 하위 엘리먼트는 정적으로 처리되어 건너 뜁니다. 이는 업데이트 성능을 최적화하는데 사용합니다.

```HTML
<!-- 단일 엘리먼트 -->
<span v-once>This will never change: {{msg}}</span>
<!-- 자식 엘리먼트를 포함하는 엘리먼트 -->
<div v-once>
  <h1>comment</h1>
  <p>{{msg}}</p>
</div>
<!-- 컴포넌트 -->
<my-component v-once :comment="msg"></my-component>
<!-- v-for 디렉티브 -->
<ul>
  <li v-for="i in list" v-once>{{i}}</li>
</ul>
```



https://kr.vuejs.org/v2/api/#v-once

조건부 렌더링

Conditional Rendering

Progressive

v-text

v-html

v-pre

v-cloak

v-once

v-for

v-if

v-else

v-else-if

v-show

v-on

v-bind

v-model

v-if

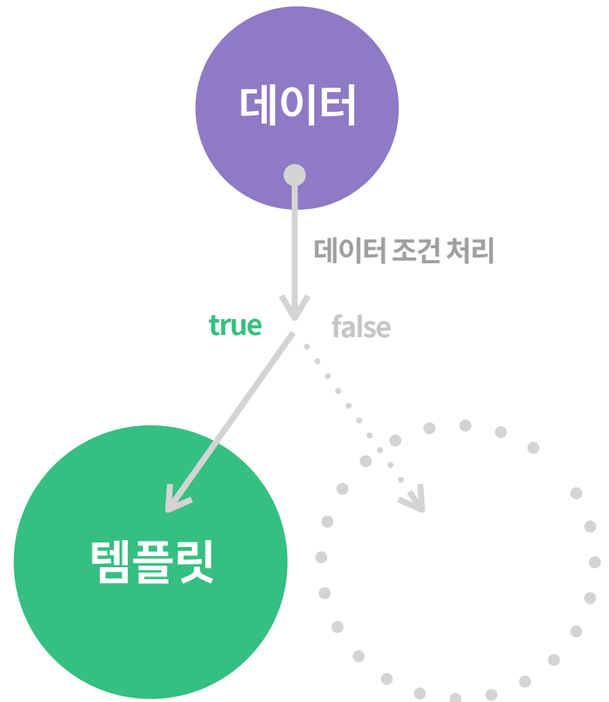
- **예상됨:** `any`
- **사용방법:**

  표현식 값의 참 거짓을 기반으로 엘리먼트를 조건부 렌더링 합니다. 엘리먼트 및 포함된 디렉티브 / 컴포넌트는 토글하는 동안 삭제되고 다시 작성됩니다. 엘리먼트가 `<template>` 엘리먼트인 경우 그 내용은 조건부 블록이 됩니다.

  조건이 변경될 때 전환이 호출 됩니다.

- Source

> ! v-if와 함께 사용하는 경우, v-for는 v-if보다 높은 우선순위를 갖습니다. 자세한 내용은 리스트 렌더링 가이드를 확인하십시오.



Progressive

https://kr.vuejs.org/v2/api/#v-if

v-text

v-html

v-pre

v-cloak

v-once

v-for

v-if

v-else

v-else-if

v-show

v-on

v-bind

v-model

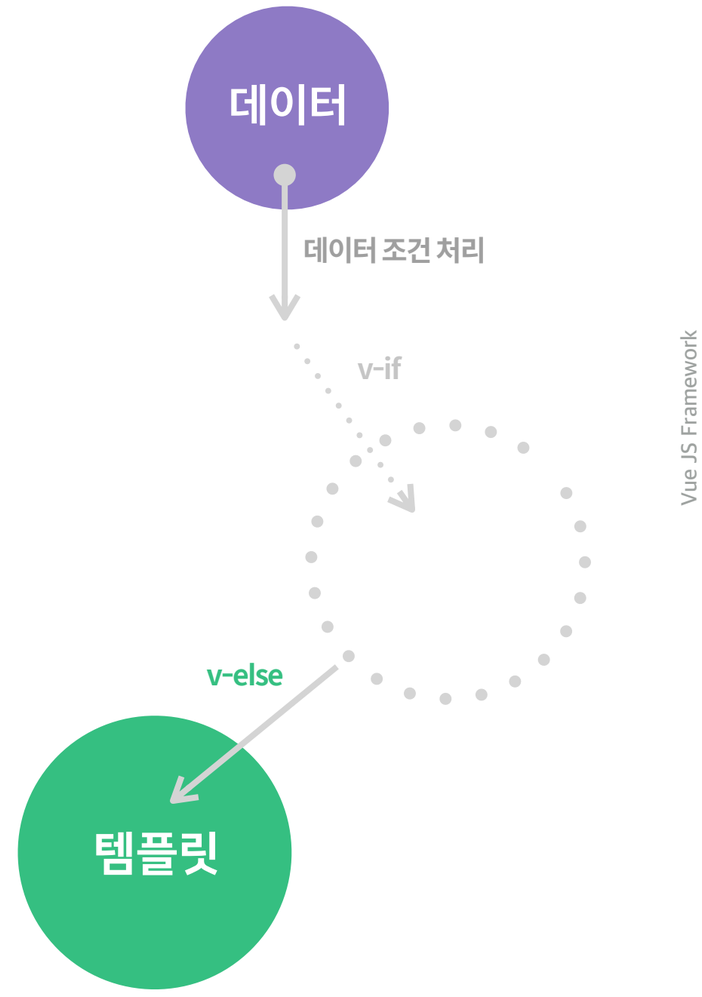
v-else

- **표현식이 필요 없습니다.**
- **제한:** 이전 형제 엘리먼트가 `v-if` 또는 `v-else-if` 이어야 합니다.
- **사용방법:**

`v-if` 또는 `v-if` / `v-else-if` 체인을위한 “else 블록”을 나타냅니다.

HTML

```
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
<div v-else>
  Now you don't
</div>
```



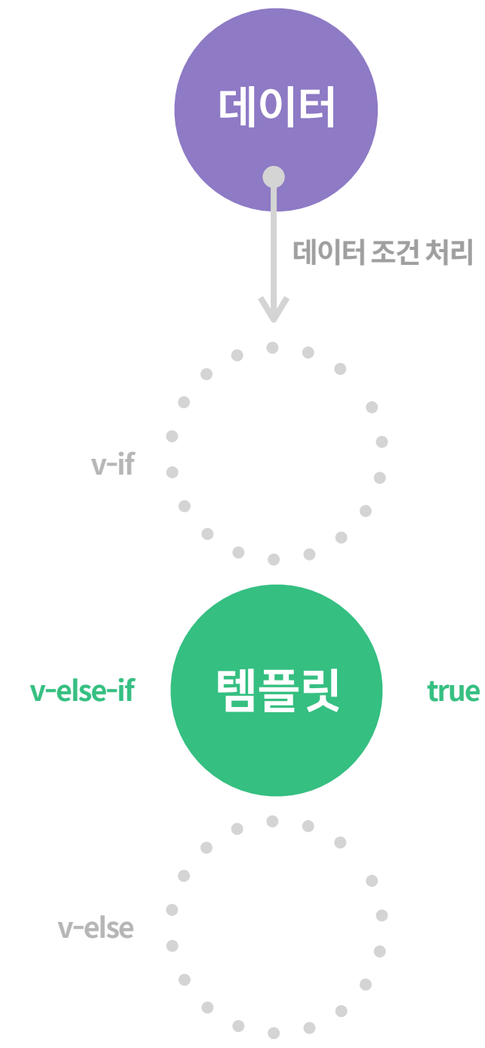
https://kr.vuejs.org/v2/api/#v-else

v-else-if

- **예상됨:** `any`
- **제한:** 이전 형제 엘리먼트가 `v-if` 또는 `v-else-if` 이어야 합니다.
- **사용방법:**

`v-if` 에 대한 “else if 블록”을 나타냅니다. 체이닝 가능합니다.

```HTML
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```



https://kr.vuejs.org/v2/api/#v-else-if

v-text

v-html

v-pre

v-cloak

v-once

v-for

v-if

v-else

v-else-if

v-show

v-on

v-bind

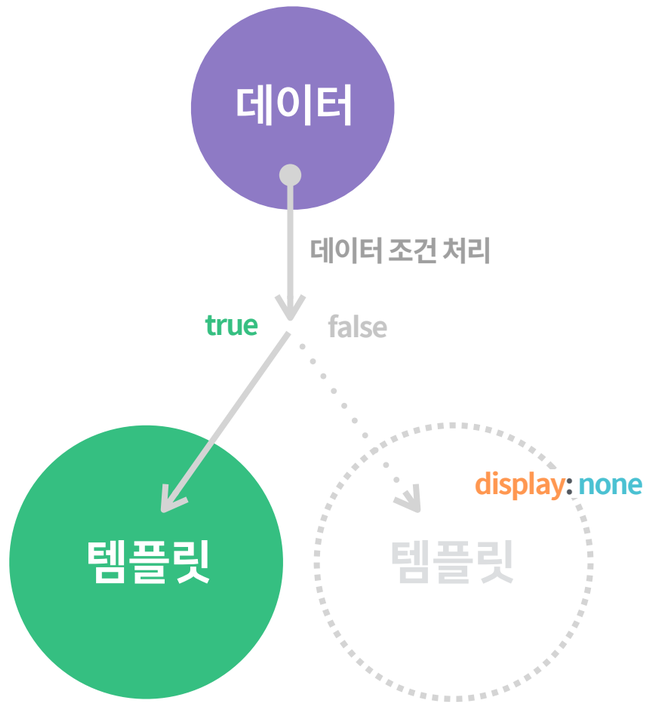
v-model

v-show

- 예상됨: `any`
- 사용방법:

토글은 표현식 값의 참에 기반을 둔 `display` CSS 속성입니다.

이 디렉티브는 조건이 바뀌면 전환이 호출 됩니다.



Progressive

https://kr.vuejs.org/v2/api/#v-show

`v-if` vs `v-show`

`v-if` 는 조건부 블럭 안의 이벤트 리스너와 자식 컴포넌트가 토글하는 동안 적절하게 제거되고 다시 만들어지기 때문에 "진짜" 조건부 렌더링 입니다

`v-if` 는 또한 **게으릅니다** 초기 렌더링에서 조건이 거짓인 경우 아무것도 하지 않습니다. 조건 블록이 처음으로 참이 될 때 까지 렌더링 되지 않습니다.

비교해보면, `v-show` 는 훨씬 단순합니다. CSS 기반 토글만으로 초기 조건에 관계 없이 엘리먼트가 항상 렌더링 됩니다.

일반적으로 `v-if` 는 토글 비용이 높고 `v-show` 는 초기 렌더링 비용이 더 높습니다. 매우 자주 바꾸기를 원한다면 `v-show` 를, 런타임 시 조건이 바뀌지 않으면 `v-if` 를 권장합니다.

> `v-show` 는 `<template>` 구문을 지원하지 않으며 `v-else` 와도 작동하지 않습니다.

https://kr.vuejs.org/v2/guide/conditional.html#v-if-vs-v-show

List Rendering

리스트 렌더링

Progressive

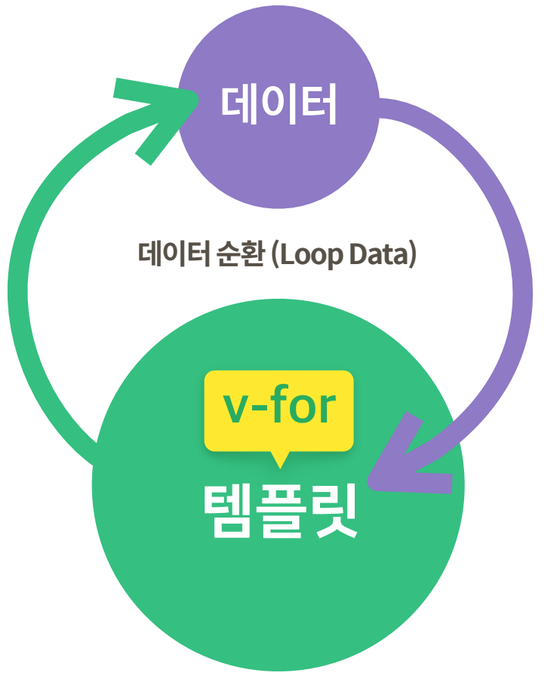
v-for

- 예상됨: `Array | Object | number | string`
- 사용방법:

원본 데이터를 기반으로 엘리먼트 또는 템플릿 블록을 여러번 렌더링합니다. 디렉티브의 값은 반복되는 현재 엘리먼트에 대한 별칭을 제공하기 위해 특수 구문인 `alias in expression` 을 사용해야 합니다.

속성 in 객체

```HTML
<div v-for="item in items">
  {{ item.text }}
</div>
```



https://kr.vuejs.org/v2/api/#v-for

```
<div class="demo">
  <h1>Vue JS 프레임워크 기능</h1>
  <ul class="vue-list">
    <li class=“vue-list__item”></li>
  </ul>
</div>
```

템플릿

Progressive

```
new Vue({
  'el': '.demo',
  'data': {
    'features': [
      '빠른 랜더링 속도',
      '초 경량 프레임워크',
      '컴포넌트 UI 지향',
      '데이터 바인딩',
      'v-* 디렉티브 지원',
      '리엑티브 시스템',
    ]
  }
});
```

Progressive

```
<div class="demo">
  <h1>Vue JS 프레임워크 기능</h1>
  <ul class="vue-list">
    <li
      class="vue-list__item"
      v-for="feature in features"
      v-text="feature"></li>
  </ul>
</div>
```

mustache 문법으로 대체 가능
{{ feature }}

```
▼<div class="demo">
  <h1>Vue JS 프레임워크 기능</h1>
 ▼<ul class="vue-list">
    <li class="vue-list__item">빠른 랜더링 속도</li>
    <li class="vue-list__item">초 경량 프레임워크</li>
    <li class="vue-list__item">컴포넌트 UI 지향</li>
    <li class="vue-list__item">데이터 바인딩</li>
    <li class="vue-list__item">v-* 디렉티브 지원</li>
    <li class="vue-list__item">리엑티브 시스템</li>
  </ul>
</div>
```

Vue JS 데이터 순환 처리 및 데이터/뷰 업데이트

vue__loop-list-add-items.html

Raw

```
<!DOCTYPE html>
<html lang="ko-KR">
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=Edge">
  <meta charset="UTF-8">
  <title>VueJS - Loop Lists &amp;amp; Add List Items</title>
  <script src="https://unpkg.com/vue@2.1.6/dist/vue.js"></script>
</head>
<body>

  <div class="demo">
    <h1>Vue JS 프레임워크 기능</h1>
    <ul class="vue-list">
      <li class="vue-list__item" v-for="feature in features">{{ feature }}</li>
    </ul>
  </div>

<script>
new Vue({
  'el': '.demo',
  'data': {
```

```
<div id="app" v-cloak>
  <h1>{{app_name}}</h1>
  <ul class="reset-list vue-features">
    <li v-for="(feature, index) of vue_features">
      <div v-for="(value, key, index) of feature">
        ({{ index }}) {{ key }}: {{ value }}
      </div>
      <hr v-if="index !== vue_features.length - 1">
    </li>
  </ul>
```

`in` 대신에 `of` 를 구분자로 사용할 수 있습니다.

배열(Array)

객체(Object)

표현식의 결과에 따라, 조건부 렌더링

디렉티브

템플릿

템플릿 요소

<template>

Progressive

HTML의 **<template>요소는** 페이지가로드 될 때 렌더링 할 것이 아니라, 이후에 자바 스크립트를 사용하여 런타임시 인스턴스화 할 수 클라이언트 측 함량을 유지하는 메커니즘입니다.

템플릿은 나중에 문서에서 사용하기 위해 저장되는 내용 조각으로 생각하십시오. 파서는 **<template>** 페이지를로드하는 동안 요소 의 내용을 처리하지만 해당 내용이 유효한지 만 확인합니다. 그러나 요소의 내용은 렌더링되지 않습니다.

콘텐츠 카테고리	메타 데이터 콘텐츠 , 흐름 콘텐츠 , 프레이 징 콘텐츠 , 스크립트 지원 요소
허용 된 콘텐츠	콘텐츠 메타 데이터 , 플로우 콘텐츠는 허용 유효한 HTML 콘텐츠 내에서 발생하는 , <dl>, <figure>, <ruby>, <object>, <video>, <audio>, <table>, <colgroup>, <thead>, <tbody>, <tfoot>, <tr>, <fieldset>, <select>, <details>요소 및 <menu>그 type속성 팝업 메뉴의 상태에있다.
태그 누락	없음, 시작 및 종료 태그는 필수 항목입니다.
허락 된 부모	<body>, <frameset>, <head>, <dl>및 <colgroup>없는 span속성
허용 된 ARIA 역할	없음
DOM 인터페이스	HTMLTemplateElement

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template

`<template>` 에 `v-if` 을 갖는 조건부 그룹 만들기

`v-if` 는 디렉티브기 때문에 하나의 엘리먼트에 추가해야합니다. 하지만 하나 이상의 엘리먼트를 전환하려면 어떻게 해야할까요? 이 경우 우리는 보이지 않는 래퍼 역할을 하는 `<template>` 엘리먼트에 `v-if` 를 사용할 수 있습니다. 최종 렌더링 결과에는 `<template>` 엘리먼트가 포함되지 않습니다.

```HTML
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

컴파일 과정에서 <template> 요소는 해석되어 처리되지 않습니다.

> ! `v-show` 는 `<template>` 구문을 지원하지 않으며 `v-else` 와도 작동하지 않습니다.

`v-for` 템플릿

템플릿 `v-if` 와 마찬가지로, `v-for` 와 함께 `<template>` 태그를 사용하여 여러 엘리먼트의 블럭을 렌더링 할 수 있습니다. 예를 들어,

HTML

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider"></li>
  </template>
</ul>
```

엘리먼트 그룹 역할 수행
그룹 단위 렌더링이 필요한 경우 사용

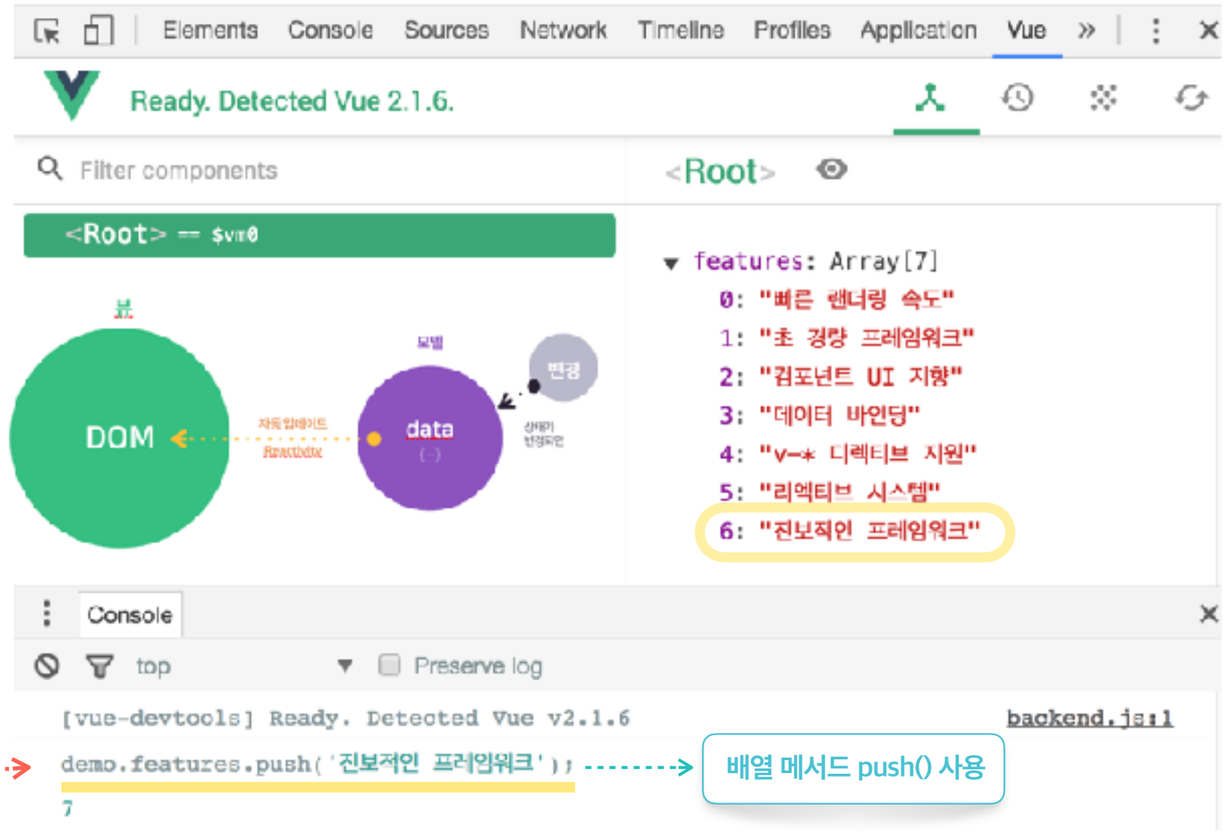
뷰 — 업데이트

View Update

Progressive

Vue JS 프레임워크 기능

- 빠른 랜더링 속도
- 초 경량 프레임워크
- 컴포넌트 UI 지향
- 데이터 바인딩
- v-* 디렉티브 지원
- 리엑티브 시스템
- 진보적인 프레임워크



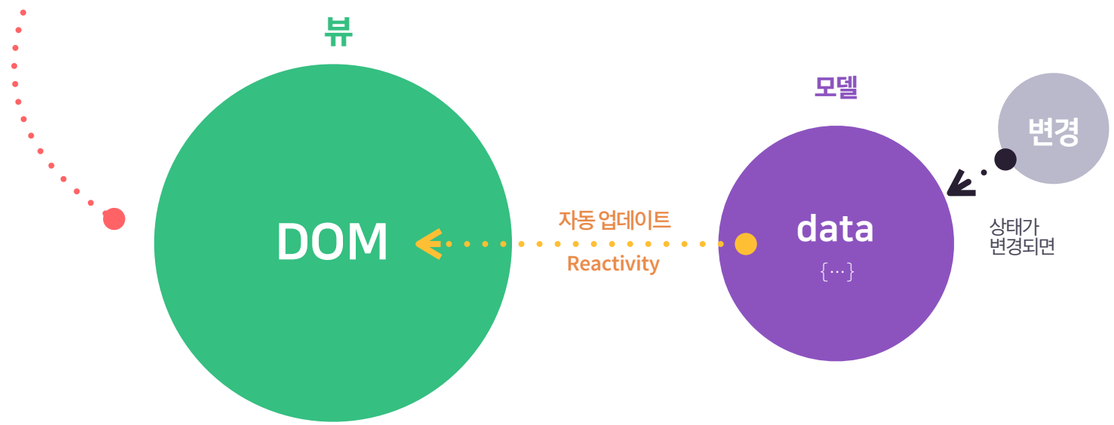

```
let demo = new Vue({
});
```

Progressive

```
<div class="demo">
  <h1>Vue JS 프레임워크 기능</h1>
  <p class="form-control">
    <label for="vue-update">아이템 추가</label>
    <input type="text" id="vue-update">
    <button type="button" class="vue-update__button">추가</button>
  </p>
  <ul class="vue-list">
    <li
      class="vue-list__item"
      v-for="feature in features"
      v-text="feature"></li>
  </ul>
</div>
```


Progressive

```
// 이벤트 바인딩
var vue_update_input  = document.querySelector('#vue-update');
var vue_update_button = document.querySelector('.vue-update__button');

vue_update_button.addEventListener('click', function() {
  var new_item = vue_update_input.value;
  demo.features.push(new_item);
});
```



Progressive

Vue JS 데이터 순환 처리 및 데이터/뷰 업데이트

vue__loop-list-add-items.html — Raw

```
<!DOCTYPE html>
<html lang="ko-KR">
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=Edge">
  <meta charset="UTF-8">
  <title>VueJS - Loop Lists &amp;amp; Add List Items</title>
  <script src="https://unpkg.com/vue@2.1.6/dist/vue.js"></script>
</head>
<body>

  <div class="demo">
    <h1>Vue JS 프레임워크 기능</h1>
    <ul class="vue-list">
      <li class="vue-list__item" v-for="feature in features">{{ feature }}</li>
    </ul>
  </div>

<script>
new Vue({
  'el': '.demo',
  'data': {
```

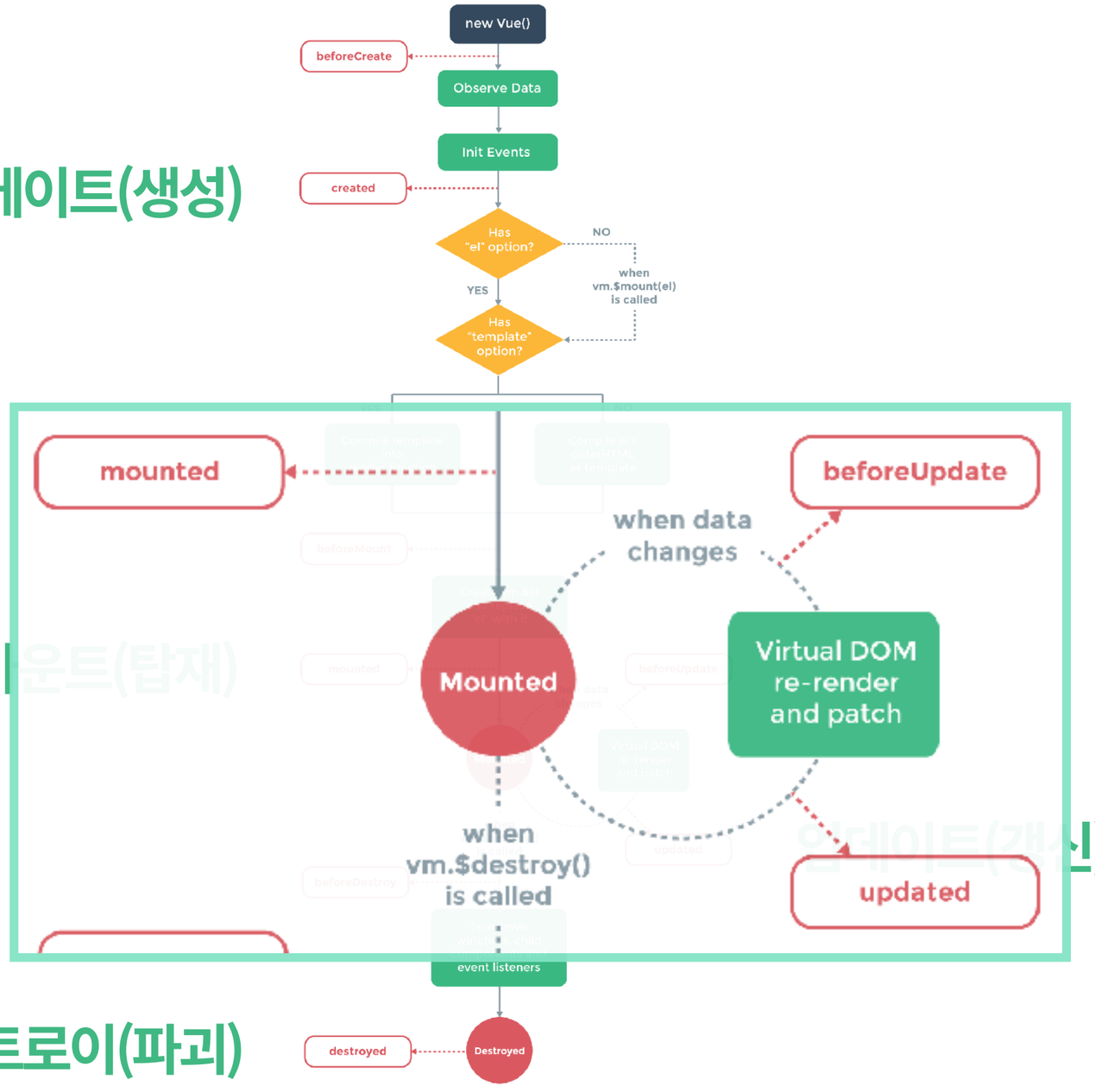

https://goo.gl/zgTfbq

Life Cycle Hook

라이프 사이클 훅

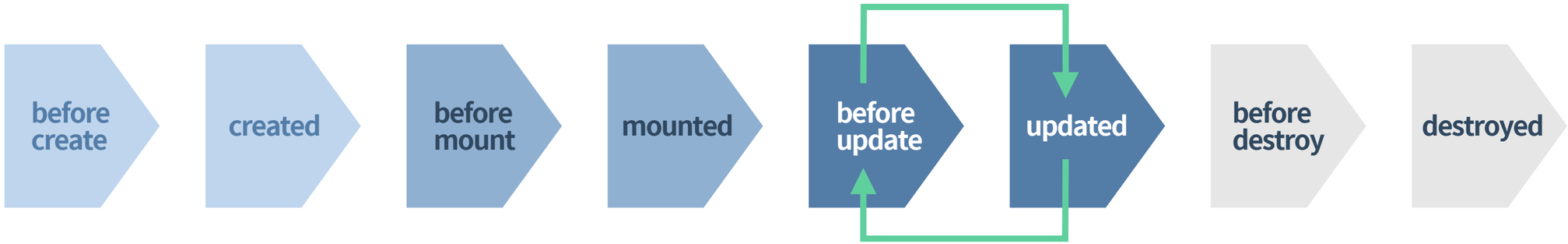
Progressive

크리에이트(생성)



디스트로이(파괴)

https://kr.vuejs.org/v2/guide/instance.html#라이프사이클-다이어그램



Progressive

```
let demo = new Vue({
  'el': '.demo',
  'data': {...},
  // 라이프사이클 훅 (콜백 함수)
  mounted: function() {
    // 이벤트 바인딩
    var vue_update_input  = document.querySelector('#vue-update');
    var vue_update_button = document.querySelector('.vue-update__button');

    vue_update_button.addEventListener('click', function() {
      var new_item = vue_update_input.value;
      demo.features.push(new_item);
    });
  }
});
```

Vue JS 데이터 순환 처리 및 데이터/뷰 업데이트

vue__loop-list-add-items.html

Raw

```
<!DOCTYPE html>
<html lang="ko-KR">
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=Edge">
  <meta charset="UTF-8">
  <title>VueJS - Loop Lists &amp;amp; Add List Items</title>
  <script src="https://unpkg.com/vue@2.1.6/dist/vue.js"></script>
</head>
<body>

  <div class="demo">
    <h1>Vue JS 프레임워크 기능</h1>
    <ul class="vue-list">
      <li class="vue-list__item" v-for="feature in features">{{ feature }}</li>
    </ul>
  </div>

<script>
new Vue({
  'el': '.demo',
  'data': {
```