Clock 컴포넌트

**학습목표**

moment.js 라이브러리 사용

Virtual DOM 개념

life cycle hook 메소드

메모리 누출 해결

**목차**

[1. clock2 프로젝트 2](#_Toc73637896)

[1) 개요 2](#_Toc73637897)

[2) clock2 프로젝트 생성 2](#_Toc73637898)

[3) 포트 번호 변경 2](#_Toc73637899)

[4) moment.js 3](#_Toc73637900)

[5) Clock.vue 생성 4](#_Toc73637901)

[6) App.vue 수정 6](#_Toc73637902)

[7) 실행 7](#_Toc73637903)

[2. 메모리 누출 8](#_Toc73637904)

[1) Virtual DOM 과 실제 DOM 8](#_Toc73637905)

[2) life cycle hook 메소드 10](#_Toc73637906)

[3) 메모리 누출 확인 12](#_Toc73637907)

[4) 메모리 누출 해결 15](#_Toc73637908)

[5) 메모리 누출 요약 16](#_Toc73637909)

[3. 재사용성 향상 17](#_Toc73637910)

[1) 개요 17](#_Toc73637911)

[2) Clock.vue 수정 18](#_Toc73637912)

[3) App.vue 수정 19](#_Toc73637913)

[4) 실행 20](#_Toc73637914)

# clock2 프로젝트

## 개요

Vue 플젝의 장점 중의 하나는 컴포넌트를 구현하기 편하다는 것이다.

프로젝트 개발을 하다 보면, 이 프로젝트 저 프로젝트에 두루 사용되는 기능을 구현하게 된다.

그런 기능을 컴포넌트 형태로 구현하면, 다음 프로젝트에 또 그 기능이 필요할 때,

컴포넌트 파일만 복사하여 쉽게 재사용할 수 있다.

동료가 만든 컴포넌트 파일을 가져와서 재사용하기도 쉽고,

오픈소스 컴포넌트를 다운로드 받아서 재사용하기도 쉽다.

현재 시각을 표시하는 예제를 컴포넌트 형태로 구현해 보자.

재사용하기 좋은 형태로 구현하자.

## clock2 프로젝트 생성

프로젝트를 생성할 적당한 폴더로 이동하자.

|  |
| --- |
| cd /c/PJ/frontend |

아래 명령을 실행해서 프로젝트를 생성하자.

|  |
| --- |
| vue create clock2 |

|  |
| --- |
|  |

default 가 선택된 상태에서 엔터 키를 누르자.

## 포트 번호 변경

컴퓨터에 오라클 DBMS나 톰캣이 설치되어 있다면, 8080 포트가 이미 사용중일 수 있다.

vue 프로젝트가 사용할 포트 번호를 8082로 변경하는 방법은 다음과 같다.

### package.json

|  |
| --- |
| ... 생략 ...    "scripts": {      "serve": "vue-cli-service serve --port 8082",      "build": "vue-cli-service build",      "lint": "vue-cli-service lint"    },  ... 생략 ... |

package.json 파일을 열고 위의 내용을 찾아서,

노란색으로 칠한 부분을 추가하자.

## moment.js

javascript 언어에서 날짜를 다룰 때, 편리한 오픈소스 라이브러리가 moment.js 이다.

사실상 javascript 표준 날짜 라이브러리이다.

<https://momentjs.com/>

CDN 서비스를 활용하여 moment 라이브러리를 사용하려면 아래 태그가 필요하다.

|  |
| --- |
| <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.1/moment.min.js"></script> |

웹페이지에 이 sript 태그가 포함되어 있으면, 웹브라우저가 moment.min.js 파일을 다운로드하여 실행한다.

Vue 프로젝트에서는 CDN 서비스를 시용하지 않고,

라이브러리 파일을 미리 프로젝트 폴더 아래 node\_modules 폴더에 다운로드하고,

트랜스파일하고 빌드하여 배포 파일을 생성한다.

### 설치

|  |
| --- |
| npm install moment |

프로젝트 폴더에서 위 명령을 실행하자.

위 명령에 의해서 moment 라이브러리 파일이 node\_modules 폴더에 다운로드된다.

## MyClock.vue 생성

src/MyClock.vue

|  |
| --- |
| <template>    <div>{{ time }}</div>  </template>  <script>  import moment from "moment";  export default {    name: "MyClock",    data() {      return { time: null };    },    mounted() {      let callback = () => this.time = moment().format("HH:mm:ss");      callback();      setInterval(callback, 1000);    }  }  </script>  <style scoped>  div { display: inline-block; font-size: 15pt; padding:3px 12px; border: 1px solid gray; }  </style> |

import moment from "moment";

moment 모듈(라이브러리)를 사용하기 위해 필요한 import 선언이다.

name: "MyClock",

이 컴포넌트의 이름은 "MyClock" 이다.

data() {

  return { time: null };

},

이 컴포넌트의 속성을 정의한다. 컴포넌트 속성 정의는 함수 형태로 구현해야 한다.

time 속성의 초기 값은 null 이다.

mounted() { ... }

이 컴포넌트가 생성되고, 컴포넌트의 속성들도 생성되고, 아직 화면은 그려지기 전에,

이 메소드가 한 번 자동으로 호출된다.

컴포넌트의 초기화를 구현하기에 적당한 곳이다.

이렇게 컴포넌트의 특정 시점에 자동으로 호출되는 메소드를 life cycle hook 메소드라고 한다.

let callback = () => this.time = moment().format("HH:mm:ss")

함수를 생성하여, callback 지역 변수에 대입한다.

화살표 함수 문법으로 구현된 함수이다.

이 함수가 호출되면, 현재 시각으로부터 "13:25:34" 형태의 문자열을 생성하여,

컴포넌트의 time 속성에 대입한다.

연월일 포멧은 "yyyy-MM-dd"

callback();

callback 지역변수의 값은 함수이다. 이 함수를 호출한다.

현재 시각 문자열이, 컴포넌트의 time 속성에 대입된다.

setInterval(callback, 1000);

지금부터 1000 밀리초 간격으로 callback 함수를 반복 호출한다.

## App.vue 수정

src/App.vue

|  |
| --- |
| <template>    <div id="app">      <h1>시계</h1>      <MyClock></MyClock>    </div>  </template>  <script>  import MyClock from './MyClock.vue'  export default {    name: "App",    components: { MyClock }  }  </script>  <style scoped>  div#app { padding: 0 30px 30px 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }  </style> |

import MyClock from './MyClock.vue'

Clock 컴포넌트를 사용하기 위해 import 한다.

name: "App",

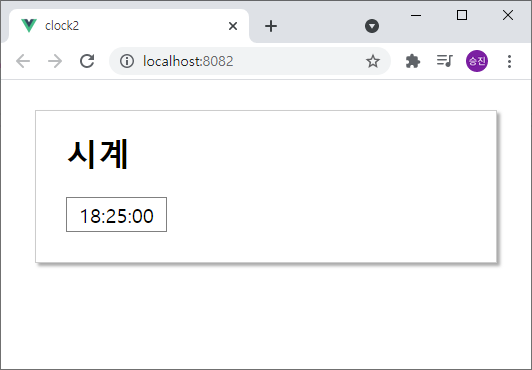
이 컴포넌트의 이름은 "App" 이다.

components: { MyClock }

사용할 컴포넌트를 여기서 미리 선언해야 한다.

## 실행

|  |
| --- |
| npm run serve |



# 메모리 누출

## Virtual DOM 과 실제 DOM

### DOM(Document Object Model)

웹브라우저가 HTML 텍스트 문서를 읽고 분석하여 (parsing 하여), HTML 문서 객체를 생성한다.

이 HTML 문서 객체가 웹브라우저 창에 그려진다.

HTML 문서 객체는, HTML 태그 객체들을 포함하고 있다.

이 객체들이 웹브라우저 내부 메모리(RAM)에 들어있다.

이 객체들을 DOM 객체라고 부른다.

### Virtual DOM

Vue 컴포넌트의 <template> 태그 부분에 정의된 HTML 태그들을 읽고 분석하여 (parsing 하여),

HTML 문서 객체와 태그 객체를 생성하는 작업은, Vue 엔진이 수행한다.

웹브라우저 내부의 DOM 객체와 구별하기 위해서,

Vue 엔진이 생성한 태그 객체를 Virtual DOM 객체라고 부른다.

Virtual DOM 객체는 Vue 엔진 내부의 객체이기 때문에 웹브라우저 창에 그려지지 않는다.

웹브라우저 창에 그려지는 것은 DOM 객체이다.

Virtual DOM 객체들이 웹브라우저 창에 그려지려면,

Virutal DOM 객체들과 똑같은 구조로, 웹브라우저 내부에 DOM 객체를 생성해야 한다.

그리고 Virual DOM 객체의 내용이 변경되면, 그 변경된 객체에 해당하는 DOM 객체의 내용도 변경되어야 한다.

### 렌더링 (rendering)

컴포넌트의 속성 값이 변경되면, 그 값이 태그 부분에 즉시 출력되어야 한다.

즉 컴포넌트의 속성 값이 변경되면, 그 속성과 연결된 태그 객체의 내용도 즉시 변경되어야 한다.

만약 속성 값이 목록이라면, 그 목록의 크기에 따라서 태그 객체가 새로 생성되거나 제거 되어야 할 수도 있다.

이 작업을 렌더링 작업이라고 부른다.

컴포넌트 속성값이 변경되면,

Vue 엔진은 Virtual DOM 객체들만 대상으로 변경 작업을 수행한다.

아직 DOM 객체는 변경하지 않기 때문에, 웹브라우저 창의 내용도 변화하지 않는다.

### 업데이트(update)

변경된 Virtual DOM 객체들과 동일한 구조로, 웹브라우저 내부의 DOM 객체를 생성하고 수정하는 작업을

업데이트 작업이라고 부른다.

업데이트 작업까지 마쳐야 웹브라우저 창의 내용도 변화하게 된다.

### 렌더링 작업과 업데이트 작업을 분리한 이유

컴포넌트의 속성 값이 변화하면, 바로 DOM 객체를 수정하면 화면이 즉시 변화할 텐데,

Virual DOM 객체가 왜 필요한가?

이런 의문을 가지는 것이 정상이다.

프론트엔드 프레임웍의 초기 버전들은 Virtual DOM 없이 바로 DOM을 수정하는 방식으로 구현되었다.

즉 DOM 객체를 대상으로 렌더링 작업을 수행하면, 그 내용이 즉시 웹브라우저 창에 그려진다.

따라서 업데이트 작업이 필요없다.

그런데, 위에서 설명한 렌더링 작업 절차가 생각보다 복잡하다.

컴포넌트 속성 값이 변경된 경우에, 그 속성에 연결된 DOM 객체를 HTML 문서 내부에서 찾고, 구조를 변경하고 내용을 수정하는 작업이 꽤 복잡해서 시간이 많이 걸린다.

렌더링 작업에 비해 업데이트 작업은 상대적으로 간단하다.

그런데 javascript 코드에서 웹브라우저 내부의 DOM 객체를 대상으로 렌더링 작업을 하는 것 보다,

Virtual DOM 객체를 대상으로 렌더링 작업을 하는 것이 훨씬 빠르다.

물론 Virtual DOM 객체를 대상으로 렌더링 작업을 했다면, 그 결과 구조에 동일하게 DOM 객체들도 수정하는 업데이트 작업을 추가로 또 해야 하지만, 업데이트 작업은 상대적으로 간단해서 금방 끝난다.

DOM 객체만 대상으로 작업하는 경우

(a) DOM 객체를 대상으로 렌더링

Virtual DOM + DOM 객체의 조합으로 작업하는 경우

(b) Virtual DOM 객체를 대상으로 렌더링

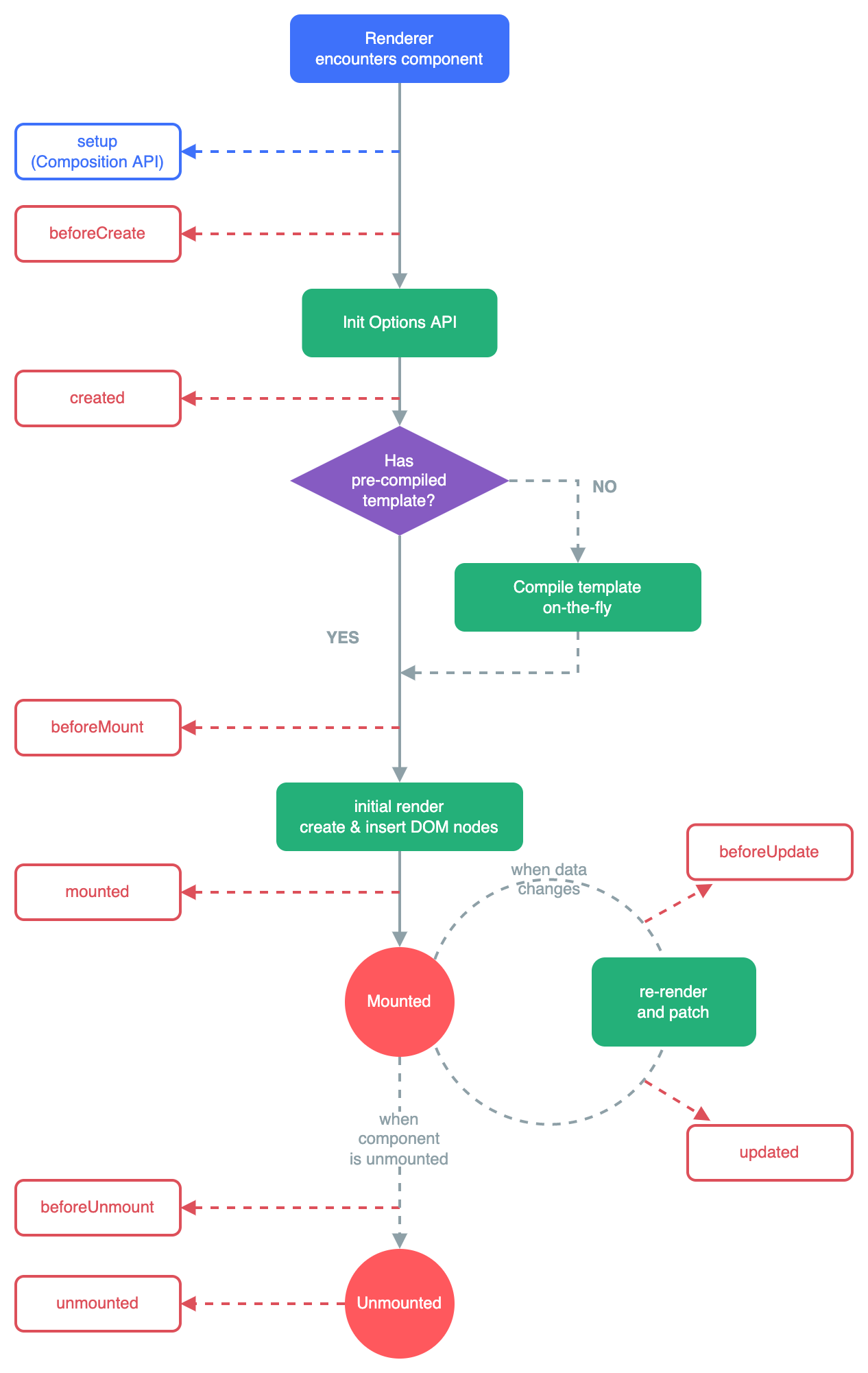
(c) Virtual DOM 객체 렌더링 결과 대로 DOM 객체 업데이트

(a) 작업시간이 (b) + (c) 작업시간 보다 훨씬 길다.

## life cycle hook 메소드

컴포넌트의 생명주기(life cycle)의 특정 시점에 자동으로 호출되는 메소드들을

life cycle hoot 메소드라고 한다.



### beforeCreate 메소드

컴포넌트 객체가 생성된 직후에 자동으로 호출된다.

아직 컴포넌트의 속성들도 생성되기 전이다.

이 메소드를 호출하여 어떤 기능을 구현하는 것은 매우 고급 작업들 뿐이다. 즉 아직 이 메소드는 몰라도 된다.

### created 메소드

컴포넌트 객체가 생성되고, 컴포넌트의 속성들도 만들어진 후에 자동으로 호출된다.

아직 Virtual DOM 객체가 생성되기 전이다.

컴포넌트의 속성들과 관련된 초기화 작업을 구현하기 적당한 곳이다.

최초 한 번 호출된다.

### beforeMount

컴포넌트 객체가 생성되고, 컴포넌트의 속성들도 만들어지고,

Virutal DOM 객체도 만들어진 후에 자동으로 호출된다.

이 메소드를 호출하여 어떤 기능을 구현하는 것은 매우 고급 작업들 뿐이다. 즉 아직 이 메소드는 몰라도 된다.

### mounted

컴포넌트 객체가 생성되고, 컴포넌트의 속성들도 만들어지고,

Virutla DOM 객체도 만들어지고, DOM 객체와 Virtual DOM 객체가 연결된(mounted) 후에 자동으로 호출된다.

DOM 객체와 관련된 초기화 작업을 구현하기 적당한 곳이다.

최초 한 번 호출된다.

### beforeUpdate

렌더링 작업 후이고 업데이트 작업 직전에 호출된다.

화면이 변화할 때 마다 호출된다.

화면이 변화하기 직전에 뭔가 추가로 해야할 일이 있다면, 여기에 구현한다.

### updated

업데이트 작업 직후에 호출된다.

화면이 변화할 때 마다 호출된다.

화면이 변화한 직후에 뭔가 추가로 해야할 일이 있다면, 여기에 구현한다.

### beforeUnmount

컴포넌트가 파괴되기 직전에 호출된다.

컴포넌트가 파괴될 때 어떤 마무리 작업이 필요하다면, 여기에 구현한다.

### unmounted

컴포넌트가 파괴된 직후에 호출된다.

### 요약

초기화 작업 mounted 메소드에 구현

마무리 작업 beforeUnmount 메소드에 구현

## 메모리 누출 확인

### MyClock.vue 수정

|  |
| --- |
| <template>    <div>{{ time }}</div>  </template>  <script>  import moment from "moment";  export default {    name: "MyClock",    data() {      return { time: null };    },    mounted() {      let callback = () => {        this.time = moment().format("HH:mm:ss");        console.log(this.time);      }      callback();      setInterval(callback, 1000);    }  }  </script>  <style scoped>  div { display: inline-block; font-size: 15pt; padding:3px 12px; border: 1px solid gray; }  </style> |

callback 함수가 호출될 때 마다 console에 현재 시각을 출력하는 코드를 추가함.

### App.vue 수정

src/App.vue

|  |
| --- |
| <template>    <div id="app">      <h1>시계</h1>      <MyClock v-if="showTimer"></MyClock>      <div>        <button type="button" @click="showTimer = true"                              :disabled="showTimer">시계 보이기</button>        <button type="button" @click="showTimer = false"                              :disabled="!showTimer">시계 감추기</button>      </div>    </div>  </template>  <script>  import MyClock from './MyClock.vue'  export default {    name: "App",    data: function() {      return {        showTimer: true      }    },    components: { MyClock }  }  </script>  <style scoped>  div#app { padding: 0 30px 30px 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }  button { padding: 0.5em 1.5em; margin: 20px 5px 0 0; }  </style> |

시계 보이기, 시계 감추기 버튼을 구현하였다.

### v-if vs v-show

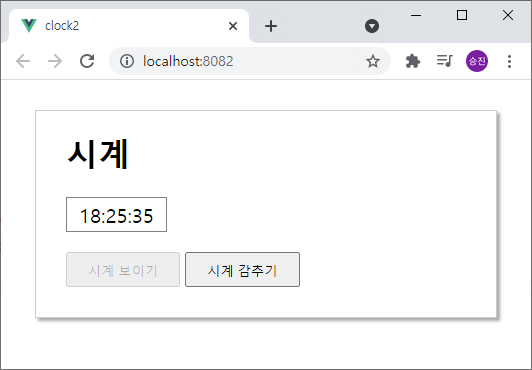
태그의 보이는 상태를 동적으로 변경하는 기능을 구현할 때, v-if 지시자나 v-show 지시자를 사용한다.

v-show 지시자는 해당 태그 객체의 보이는 상태만 변경한다.

v-if 지시자는 보이는 상태가 변경될 때 마다, 해당 태그 객체를 생성하고 파괴한다.

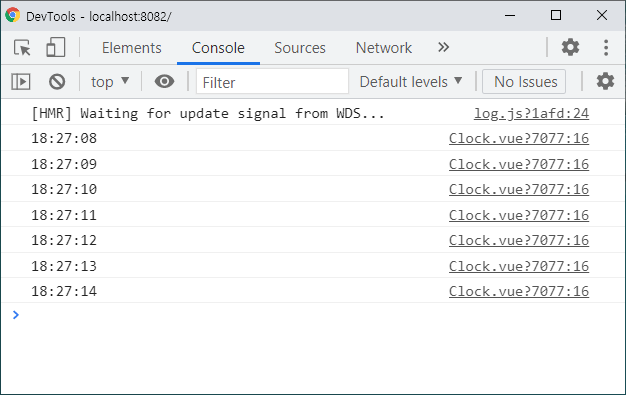
즉 보이지 않게되면 태그 객체를 파괴하고, 보이는 상태가 되면, 태그 객체를 다시 생성한다.

### 실행



F12 키를 눌러서 개발자 창을 열자.

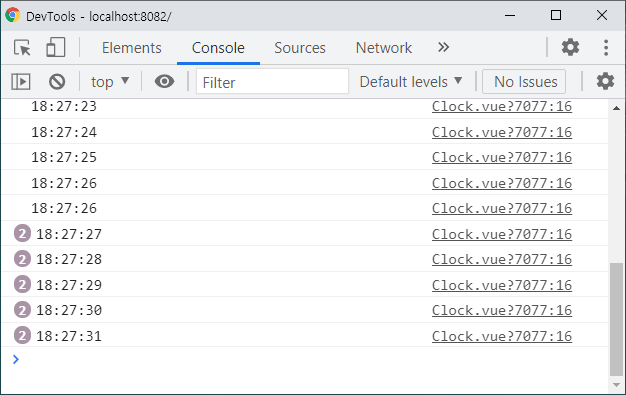
그리고 위 화면에서 '시계 감추기' 버튼을 클릭하자.



시계 감추기를 누르면 MyClock 컴포넌트가 메모리에서 제거되어야 하는데,

여전히 콜백 함수가 호출되고 있다.

위 화면에서 '시계 보이기' 버튼을 클릭하자.



시계 보이기를 누르면 MyClock 컴포넌트가 새로 생성되어서

콜백 함수가 1초에 한 번씩 호출되기 시작한다.

아까 제거되어야 할 컴포넌트도 제거되지 않고 콜백 함수가 호출되고 있기 때문에

지금 메모리에는 MyClock 컴포넌트 객체가 두 개이고, 각각 1초에 한 번씩 콜백 함수가 호출되고 있다.

그래서 로그 메시지 출력도 1초에 두 개씩 출력된다.

## 메모리 누출 해결

### MyClock.vue 수정

|  |
| --- |
| <template>    <div>{{ time }}</div>  </template>  <script>  import moment from "moment";  export default {    name: "MyClock",    data() {      return { time: null };    },    mounted() {      let callback = () => {        this.time = moment().format("HH:mm:ss");        console.log(this.time);      }      callback();      this.timerId = setInterval(callback, 1000);    },    beforeUnmount() {      clearInterval(this.timerId);    }  }  </script>  <style scoped>  div { display: inline-block; font-size: 15pt; padding:3px 12px; border: 1px solid gray; }  </style> |

this.timerId = setInterval(callback, 1000);

지금부터 1000 밀리초 간격으로 callback 함수를 반복 호출하도록 등록한다.

등록 결과로 리턴된 id 값을 컴포넌트의 timerId 속성에 대입한다.

javascript 문법에서는 객체의 속성에 값을 처음 대입할 때, 그 속성이 생성된다.

beforeUnmount() {

  clearInterval(this.timerId);

}

컴포넌트가 파괴될 때, 콜백 함수 반복 호출 등록도 같이 취소한다.

이렇게 취소하지 않으면, 메모리 누출이 발생한다.

콜백 함수 등록이 취소되지 않으면, 반복 호출 엔진이 콜백 함수를 계속 참조하게 된다.

콜백 함수 본문에서 컴포넌트의 time 속성을 사용하고 있으므로,

콜백 함수가 컴포넌트를 참조하고 있다.

참조되고 있는 것은 가비지 콜렉션 되지 않기 때문에,

콜백 함수도, 컴포넌트도 가비지 콜렉션 될 수 없게 된다. => 메모리 누출

따라서 컴포넌트 내부에서 setInterval 함수를 사용했다면, beforeUnmounted 에서 clearInterval 해야 한다.

웹브라우저 창을 새로 고침하거나, 웹브라우저 창이 다른 페이지로 넘어가면, 메모리 누출 문제는 해결된다.

그런데 고급 Vue 앱은 보통, 앱의 모든 기능을 한 페이지에 다 구현하는

Single Page App 형태로 구현되기 때문에,

다른 페이지로 넘어갈 일이 없다.

그리고 Single Page App에서 새로 고침은, 앱의 재시작을 의미하기 때문에, 새로 고침을 거의 하지 않는다.

따라서 Single Page App 개발에서는 메모리 누출 문제를 신경써야 한다.

### 실행

시계 감추기 버튼을 누르면 MyClock 객체가 메모리에서 완전히 제거되기 때문에,

로그 메시지가 출력되지 않는다.

시계 보이기 버튼을 누르면 MyClock 객체가 새로 한 개 생성되어,

로그 메시지가 1초에 한 개씩 출력된다.

## 메모리 누출 요약

내가 구현하지 않은 어떤 서비스에, 콜백 함수를 등록했다면,

그 콜백 함수 등록을 취소하는 코드를 beforeUnmount 메소드에 구현하자.

# 재사용성 향상

## 개요

재사용은 소스 코드를 수정하지 않고 그대로 재사용해야 한다.

MyClock 컴포넌트를 다른 프로젝트서 재사용할 때, 아마도 CSS 서식이 달라질 것이다.

그리고 시각 출력 포멧도 달라질 수 있다.

이렇게 앞으로 수정될 가능성이 있는 부분을 컴포넌트 외부로 추출하자.

재사용할 때 수정될 가능성이 거의 없는 부분만 컴포넌트 내부에 구현하자.

## MyClock.vue 수정

|  |
| --- |
| <template>    <div>{{ time }}</div>  </template>  <script>  import moment from "moment";  export default {    name: "MyClock",    props: {      format: { type: String, default: "HH:mm:ss" }    },    data() {      return { time: null };    },    mounted() {      let callback = () => this.time = moment().format(this.format)      callback();      this.timerId = setInterval(callback, 1000);    },    beforeUnmount() {      clearInterval(this.timerId);    }  }  </script>  <style scoped>  div { display: inline-block; }  </style> |

### 시각 출력 포멧

수정될 가능성이 있는 시각 출력 포멧을 props로 구현하였다.

시각 출력 포멧을 변경하려면, 컴포넌트 소스 코드를 수정할 필요없이,

새 포멧 문자열을 props 값으로 전달하면 된다.

format props 값 전달을 생략한 경우, 디폴트 값은 "HH:mm:ss" 이다.

### 컴포넌트 서식

수정될 가능성이 없는 공통 서식만 컴포넌트 내부에 구현하였다.

## App.vue 수정

|  |
| --- |
| <template>    <div id="app">      <h1>clock</h1>      <MyClock v-if="showTimer" class="clockStyle1" format="YYYY-MM-DD HH:mm:ss"></MyClock>      <MyClock v-if="showTimer" class="clockStyle2"></MyClock>      <div>        <button type="button" @click="showTimer = true"                              :disabled="showTimer">시계 보이기</button>        <button type="button" @click="showTimer = false"                              :disabled="!showTimer">시계 감추기</button>      </div>    </div>  </template>  <script>  import MyClock from './MyClock.vue'  export default {    name: "App",    data() {      return {        showTimer: true      }    },    components: { MyClock }  }  </script>  <style scoped>  div#app { padding: 0 30px 30px 30px; margin: 30px auto; max-width: 400px;        border: 1px solid #ccc; box-shadow: 3px 3px 3px #aaa; }  button { padding: 0.5em 1.5em; margin: 20px 5px 0 0; }  .clockStyle1, .clockStyle2 { font-size: 15pt; border: 1px solid gray; }  .clockStyle1 { padding: 3px 12px; margin-right: 5px; background-color: #ffa; }  .clockStyle2 { padding: 3px 30px; background-color: #bfb;  }  </style> |

format="YYYY-MM-DD HH:mm:ss"

format props 값을 전달한다.

<MyClock v-if="showTimer" class="clockStyle2"></MyClock>

컴포넌트 태그의 class 애트리뷰트는, 컴포넌트 내부의 루트 태그에 상속된다.

MyClock 컴포넌트의 루트 태그는 div 태그이다.

이 div 태그에 class="clockStyle2" 애트리뷰트가 상속된다.

따라서 <div class="clockStyle2"> 형태가 된다.

컴포넌트에 대한 서식 중 수정될 가능성이 높은 서식은 컴포넌트를 사용하는 부모 쪽 소스코드에 구현하였다.

## 실행

