Vue3

Vue3 기초

Vue3 실행하기

1. CDN을 이용한 실행
2. npm이용한 프로젝트 생성

CDN을 이용한 Vue 실행

CDN

- 외부에서 제공되는 패키지의 주소를 참조하여 실행
- 작성중인 HTML 문서에 다음 코드 작성

CDN을 이용한 Vue 실행

CDN을 이용한 Vue 실행

렌더링 방식

- jquery : 명령형 렌더링(Imperative Rendering)
- vue, react : 선언형 렌더링(Declarative Rendering)

선언형 렌더링

- 데이터 변수로 선언된 값을 템플릿 내에 특정 방식으로 접근
- DOM 엘리먼트를 무조건 다시 렌더링하지 않고 DOM과 연결된 상태와 속성이 변경될 때 자동으로 DOM 엘리먼트를 업데이트

NPM / Vite를 이용한 프로젝트 생성

패키지 설치를 이용한 프로젝트 생성

- 실제 프로젝트 개발에 사용
- npm을 이용하여 관련 라이브러리들을 프로젝트 코드들과 함께 컴파일 한 후 코드 축소
- 결과물은 하나의 js파일로 생성
- 결과물의 크기가 큰 경우 비동기식으로 필요할 때 필요한 코드를 불러오는 Lazy 로딩 방식 이용
- vue ES 패키지를 다운로드 받아 개발

NPM / Vite를 이용한 프로젝트 생성

vue 설치

```
> npm install vue
```

- -g 옵션(전역 설정)
  - 지정하지 않으면 node_modules 폴더 생성하고 해당 폴더에 다운로드한 라이브러리 이동
  - 지역설치는 하드웨어 인프라에서 독립된 개발 환경 조성
  - 하나의 서버에서 여러 개발자들이 동시에 다양한 프로젝트 개발 진행
- npm을 이용하여 vue 설치 시 Webpack이나 Rollup 같은 다양한 모듈 번들러를 이용해 프로젝트 컴파일 가능
- vue3에서는 기존 번들러로도 컴파일 가능하며 새로운 빌드툴인 vite 제공

NPM / Vite를 이용한 프로젝트 생성

vite

- vue3에서 별다른 번들 생성 없이 ES Modules를 바로 웹 브라우저에 렌더링 할 수 있도록 만든 개발 툴
- 매우 빠른 HMR(Hot Module Replacement) 제공
- 번들 생성 과정이 필요 없어 서버의 시작 속도 빠름
- 개발자가 번들 없이 모듈화된 컴포넌트의 수정사항을 브라우저로 확인 가능

vite를 이용한 프로젝트 생성

```
> npm init vite [프로젝트명]
```

실습

실습 준비

- 실습 폴더 생성
- 콘솔 또는 터미널에서 cd 명령어 사용하여 폴더 선택
- 코드 작성

```
> npm init vite [프로젝트명]
```

```
? Ok to proceed? (y) : [ Y ]
? Project name : [ Project name ]
? Select a framework : [ Vue ] - ↓ or ↑ 이동, [Enter] 선택
? Select a variant : [ Javascript ]
```

실습

디렉토리 이동

- 생성된 프로젝트 디렉토리로 이동

```
> cd [프로젝트명]
```

의존성 도구 설치

- 프로젝트 개발을 위한 의존성 패키지 설치
- 설치 후 “node_modules” 폴더 생성

```
> npm install
```

프로젝트 실행

- 개발 서버를 실행시켜 localhost 주소로 접속

```
> npm run dev
```

[추가]

vite없이 vue를 이용한 프로젝트 생성

```
> npm init vue
```

```
✔ Project name: … <your-project-name>
✔ Add TypeScript? … No / Yes [ No ]
✔ Add JSX Support? … No / Yes [ No ]
✔ Add Vue Router for Single Page Application development? … No / Yes [ No ]
✔ Add Pinia for state management? … No / Yes [ No ]
✔ Add Vitest for Unit testing? … No / Yes [ No ]
✔ Add an End-to-End Testing Solution? … No / Cypress / Playwright [ No ]
✔ Add ESLint for code quality? … No / Yes [ No ]
✔ Add Prettier for code formatting? … No / Yes [ No ]
```

Vue3 핵심 문법

1. 컴포넌트 구조
2. Composition 함수
3. 컴포넌트 생명주기
4. 선언적 렌더링
5. 핵심 디렉티브

SFC(Single File Component)

SFC

- Vue의 컴포넌트를 하나의 파일로 제작
- 코드가 간결하고 관리가 쉬움
- <template>, <script>, <style>로 구분

template

- 렌더링 될 html 코드

SFC(Single File Component)

script

- 컴포넌트에 적용될 스크립트 코드 작성
- setup 속성 사용 시 LOC(Line of Code) 절약 가능

style

- 컴포넌트에 적용될 스타일 작성
- scoped 속성 추가 시 해당 컴포넌트에만 스타일 적용

setup(컴포지션 함수)

기존 vue 버전

- Options API만 제공
- 컴포넌트 생성시 data, methods, computed 옵션 작성

setup

- 기존 자바스크립트 문법과 동일
- setup 함수는 객체 반환
- 반환된 객체에는 HTML에서 사용할 변수가 포함되어야 함

```
setup() {
    const data = 1
    return { data }
}
```

```
setup() {
    const rtn = () => { data = 2 }
    return { rtn }
}
```

컴포넌트 Life Cycle(생명주기)

생명주기

- 컴포넌트 생성 – DOM 노드에 마운트 – 불필요한 엘리먼트를 제거하는 일련의 과정
- 생명주기 훅(Hook) 제공 – 각 생명주기를 후킹(Hooking)할 수 있는 방법 제공

예)

Options API

```
update(){
    // Update Action
}
```

Composition API

```
setup() {
    onUpdated( ( ) => { // Update Action } )
}
```

컴포넌트 Life Cycle(생명주기)

beforeCreate

- 컴포넌트를 생성하기 전에 호출
- 컴포지션 API의 setup() 함수는 beforeCreate를 대체
- 컴포넌트 생성 전에 호출되므로 생성된 data, data 관찰을 위한 watch 등이 동작하지 않음

created

- 컴포넌트가 생성되면 호출
- 컴포지션 API의 setup() 함수가 beforeCreate와 create를 함께 대체
- 컴포넌트의 옵션에 접근이 가능하므로 data 옵션에 선언한 데이터 초기화 시 많이 사용

컴포넌트 Life Cycle(생명주기)

mounted(onMounded)

- 실제로 컴포넌트의 구성요소들이 DOM 엘리먼트로 마운트 된 후에 호출
- 실제 엘리먼트 참조 가능
- onRenderTracked 라는 생명주기 훅에서 관찰 가능
- 실제 엘리먼트에 동적으로 변화를 주고자 하는 경우 활용
- 이후 onRenderTriggered 훅 호출됨

beforeUpdate(onBeforeUpdate)

- 데이터가 변경되었지만 아직 DOM에 반영되지 않은 경우 호출
- 변경 사항이 DOM에 반영되지 않았으므로 실제 엘리먼트를 참조하는 변수값 사용 불가

컴포넌트 Life Cycle(생명주기)

updated(onUpdated)

- 데이터가 변경되어 DOM이 적용된 시점에 호출
- 실제 DOM이 업데이트 되면서 참조된 변수를 이용한 다양한 역할 수행 가능
- 해당 컴포넌트만 수정을 보장하며 자식 노드들의 업데이트를 보장하지는 않음
- 자식 컴포넌트들의 수정까지 보장하기 위해 nextTick 사용

```
update(){
    this.$nextTick( function() {
        // 모든 자식 요소 업데이트 완료
    } )
}
```

beforeUnmount(onBeforeUnmount)

- 컴포넌트가 탈착되기 직전에 호출
- 모든 기능의 사용 가능한 상태이므로 컴포넌트가 분리되기 전에 수행해야 될 내용 작성

컴포넌트 Life Cycle(생명주기)

activated(onActivated)

- keep-alive 태그 : 컴포넌트가 다시 렌더링 되는 것을 방지하고 상태를 유지 시킴
- v-is와 함께 사용되며 v-is 디렉티브가 컴포넌트를 변경할 때 기존 컴포넌트의 형태가 사라지지 않게 하기 위해 사용
- keep-alive 태그로 컴포넌트의 상태가 보존되기 시작하면 onActivated 훅 호출

```
<keep-alive>
    <compoment v-is="currentComponent" />
</keep-alive>
```

deactivated(onDeactivated)

- keep-alive로 상태가 유지되던 컴포넌트가 효력 상실 시 호출
- 소스코드 수정 후 저장하면 Vite의 HMR이 해당 컴포넌트를 다시 렌더링하는데 이 때 keep-alive로 activated 된 컴포넌트에 deactivated 호출 확인 가능

컴포넌트 Life Cycle(생명주기)

renderTracked(onRenderTracked)

- Virture DOM이 변경될 때마다 관찰을 목적으로 호출
- 이 함수로 DebuggerEvent 객체를 확인하여 Virture DOM의 변경 이유 확인 가능
- DebuggerEvent는 target 속성으로 Virture DOM 변경 추적

```
renderTracked( e ) {
    console.log(e.target)
}
```

renderTriggered(onRenderTriggered)

- Virture DOM이 실제 DOM에 반영되어야 할 때 호출
- onMounted, onActivated, onUpdated와 같이 실제 DOM이 변경되기 직전에 호출
- DebuggerEvent로 호출이유 파악 가능
  예) 새로운 값이 추가된 경우 - DebuggerEvent의 type 속성에 “add”, newValue 속성에 추가된 값 입력 확인 가능

컴포넌트 Life Cycle(생명주기)

errorCaptured(onErrorCaptured)

- 자손 컴포넌트에 에러가 발생한 경우 어느 컴포넌트에서 발생했는지 확인
- 실제 동작 중에 에러가 발생되면 안되므로 개발 중 에러를 캡처하기 위해 사용

선언적 렌더링

선언적 렌더링

- 변수를 선언하고 값을 입력하면 자동으로 DOM 갱신

Options API

- data 옵션에 변수 선언

Composition API

- setup 함수 생성 후 내부에 자바스크립트 방식으로 변수 선언
- 선언된 변수는 반드시 객체 형식으로 반환

머스태시(이중 중괄호{{ }}) 표기법

- 템플릿 객체 내에서 변수의 값 출력

ES6 단축 속성(Shorthand Property)

단축 속성

- Setup 함수에서 반환할 때 객체의 키 값과 동일한 변수명을 값으로 사용하는 경우 값을 생략하여 짧은 코드로 작성 가능

```
setup() {
    const date2 = Date().toString()
    return {
      date2 : data2,
    }
  }
```

```
setup() {
    const date2 = Date().toString()
    return {
      date2,
    }
  }
```

v-html 디렉티브

머스태시 & v-text

- 머스태시와 v-text 디렉티브는 HTML 엘리먼트의 textContent를 업데이트
- 문자열을 변수로 대입하더라도 변수에 저장된 값이 아닌 변수명이 문자열로 반환

v-html

- HTML 엘리먼트의 innerHTML로 값 전달되므로 문자열을 HTML 마크업 언어로 표현
- 변수의 값이 반드시 HTML 평문이어야 함
- Vue 문법을 사용해도 컴파일 되지 않음

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

데이터 결합 디렉티브

v-bind

- 단방향 결합
- 변수의 값이 템플릿으로만 결합
- 템플릿의 HTML 태그가 변경한 값이 변수에 반환되지 않음
- 디렉티브 생략 가능

```
<엘리먼트 v-bind:속성="변수명"></div>
```

```
<엘리먼트 :속성="변수명"></div>
```

데이터 결합 디렉티브

v-model

- 양방향 결합
- 변경된 변수의 값이 DOM에 업데이트
- 템플릿에서 변경된 값이 변수의 값을 변경
- 양방향이 가능한 모든 태그에서 사용 가능
- 별도의 속성을 정의하지 않으며 반드시 vaule 속성과 결합
- Vue3 에서는 하나의 엘리먼트에 여러 개의 v-model 디렉티브 사용 가능
- 디렉티브 생략 불가능

```
<엘리먼트 v-model="변수명"></div>
```