Vue3

Vue3 기초

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>
```

데이터 결합 디렉티브

v-bind VS v-model

```
<엘리먼트 v-bind:속성="변수명"></div>
```

```
<엘리먼트 v-model="변수명"></div>
```

데이터 결합 디렉티브

ref 함수

- vue에서 함수를 import 한 후 사용

```
import { ref } from 'vue'
```

- setup 함수에서는 반응형 변수 선언 시 값을 ref 함수의 인자로 입력
- ref 또는 reactive 함수를 이용하여 데이터를 프록시 객체로 생성하면 데이터의 변경이 프록시 객체가 아닌 원래의 데이터에서 수행
- const로 선언된 변수도 프록시의 변화는 없이 데이터만 변경 가능
- 반환된 값은 템플릿과 결합하기 위해 return 사용

데이터 결합 디렉티브

ref 함수의 문법적 슈가

- <script> 선언문에 setup 속성을 추가
- components 옵션 생략
- setup 함수 생략

```
<script setup>
    import { ref } from 'vue'
    const title = ref('DIY')
    const change() => { title = 'Do It Yourself' }
</script>
```

이벤트 리스너

v-on 디렉티브

- HTML 태그가 발생시키는 이벤트를 캡처하여 저장된 스크립트를 수행하거나 함수를 호출하는 디렉티브
- 사용자 정의 컴포넌트에서 발생된 이벤트에도 사용 가능

```
v-on:click="스크립트코드 또는 함수"
```

- "@"로 줄여서 사용 가능

```
@click="스크립트코드 또는 함수"
```

이벤트 리스너

Composition API

- 함수 선언 시 return 문을 통해 값 반환
- ref함수를 통해 선언된 변수는 프록시(반응성) 객체로 변환됨
- 함수 값을 변경하기 위해 value 속성 사용

```
counter2.value++
```

- setup 함수에서 생성된 데이터 변수는 setup 함수 내에서만 사용 가능

Options API

- methods 옵션 내부에 함수 생성
- methods 옵션에 선언된 함수는 data 옵션에 선언된 변수만 접근 가능

이벤트 리스너

이벤트 수식어

- 이벤트의 동작을 변형하기 위한 함수
- v-on 디렉티브는 이벤트 함수 호출을 이벤트 핸들러 메서드에서 하지 않고

```
예) event.preventDefault()
```

이벤트를 받는 태그에서 직접 사용 가능

```
<a href="#" @click.prevent="">
```

이벤트 리스너

```
<button @click="onClick">값 증가 : 함수 호출</button>

methods: {
    onClick: function (event) {
      if (event) {
        event.preventDefault() // 버튼의 기본 동작 제한
      }
      this.counter++
    },
  },
```

```
<button @click.prevent="onClick">값 증가 : 함수 호출</button>
```

이벤트 리스너

이벤트 수식어

이벤트 수식어	동작
.stop	이벤트 전파 방지, stopPropagation()과 동일
.prevent	브라우저의 기본 동작 제한, preventDefault()와 동일
.capture	이벤트리스너의 capture 옵션 활성화
.self	이벤트가 자식 엘리먼트가 아닌 현재 엘리먼트에서 발생했을 때만 핸들러 호출
.once	최대 한번만 클릭 허용, .once.prevent와 같이 사용 가능
.passive	이벤트 리스너의 passive 옵션 활성화
.exact	정확하게 해당 키를 누른 경우에만 핸들러 호출, @click.ctrl.exact
.left	마우스의 왼쪽 버튼을 눌렀을 때 핸들러 호출
.right	마우스의 오른쪽 버튼을 눌렀을 때 핸들러 호출
.middle	마우스의 가운데 버튼을 눌렀을 때 핸들러 호출

조건문

v-if 디렉티브

- 일반적인 스크립트 문법 사용
- 디렉티브를 큰따옴표로 구성한 경우 내부의 문자열에는 작은따옴표 사용
- v-if
  v-else-if
  v-else
- 빠르게 애플리케이션의 그림을 그려주지만 조건이 변경될 때마다 해당 엘리먼트를 다시 그림 (<-> v-show 디렉티브)
- 조건이 자주 변경되지 않는 경우에 유리

```
v-if="count > 0"
v-if="text == text"
```

조건문

v-show 디렉티브

- v-if와 유사한 역할 수행
- 모든 조건의 DOM 엘리먼트를 그린 후 조건에 맞지 않는 엘리먼트를 hide 처리
- 처음 렌더링은 느린 편이지만 조건이 변경되는 경우 빠르게 전환됨
- 조건이 자주 변경되는 경우 유리

반복문

v-for 디렉티브

- 동일한 문장 또는 규칙을 가지는 문장을 반복하는 경우에 사용
- 배열, 객체, 데이터베이스 등 많은 데이터를 가지고 있는 변수의 값을 출력할 때 용이
- 자바스크립트의 loop를 이용해 HTML 태그를 포함한 문자열을 반환하고 해당 문자열을 v-html 디렉티브로 표현 가능

```
v-for="값 in 배열"
v-for="(값, 인덱스) in 배열"
```

```
v-for="값 in 객체"
v-for="(값, 키) in 객체"
v-for="(값, 키, 인덱스) in 객체"
```

- key 속성과 같이 사용하는 것이 좋음
- 변수를 반응성으로 만들기 위해 프록시 객체를 생성하는 ref를 사용하지만 배열이나 객체는 reactive 함수 사용

반복문

computed 속성

- 반응형 애플리케이션 구현의 중요한 기능 중 하나
- ref, reactive, watcher는 실시간으로 데이터의 변경을 감시하지만 computed는 원하는 대로 데이터를 변경
- 간단한 데이터의 조작은 v-if, v-for 등으로 비슷하게 구현 가능하지만 computed는 복잡한 연산이나 기능의 구현 가능
- 호출이 되면 반드시 새롭게 계산을 진행하고 DOM을 업데이트하는 함수와는 다르게
  내부 반응성 변수의 값이 변하지 않으면 결과를 캐시에서 가져와 사용하며 DOM에 업데이트 하지 않음
- Options API - computed 속성 사용
  Composition API – vue 패키지에서 computed 컴포지션 API 호출하여 사용

[참고] 배열의 .map() 메소드와 .filter() 메소드

map() 메서드

- 배열 내의 모든 요소를 돌면서 주어진 함수의 결과를 모아 새로운 배열을 리턴
- 결과를 return으로 추출
- 실행문이 한 개만 반환될 경우, return 키워드와 {} 를 생략 가능

```
const a = numbers.map(number => {
  return number < 3
})

// 리팩토링(간결한 코드)
const a = numbers.map(number => number < 3)
```

[참고] 배열의 .map() 메소드와 .filter() 메소드

map() 구문

```
array.map(callbackFunction(current value[, index[, array]])[, thisArg])
```

- currentValue : 배열 내 현재 값
- index : 배열 내 현재 값의 인덱스
- array : 현재 배열
- thisArg : callbackFunction 내에서 this로 사용될 값

[참고] 배열의 .map() 메소드와 .filter() 메소드

화살표 함수

- 정한 실행문이 한 개만 반환될 경우, return 키워드와 {} 를 생략
- 소괄호[()]로 한번 중괄호[{}]를 감싸 주어야 객체 데이터가 화살표 문법에 의해서 반환

```
const colors = ["yellow", "red", "orange", "green", "red"]
const a = colors.map((color, index) => ({
    id: index,
    name: color
  })
```

[참고] 배열의 .map() 메소드와 .filter() 메소드

filter() 메서드

- 배열 내의 모든 요소를 돌면서 주어진 함수의 조건에 맞는 요소만을 모아 새로운 배열을 리턴

filter() 구문

```
let newArray = arr.filter(callback(currentValue[, index, [array]]) {
  // return element for newArray, if true
}[, thisArg]);
```

- currentValue : 배열 내 현재 값
- index : 배열 내 현재 값의 인덱스
- array : 현재 배열
- thisArg : callbackFunction 내에서 this로 사용될 값