3.Vue composition API

강사 강태광

1-1. Composition API

1. Options API
   - Vue2에서 애플리케이션을 만들 수 있는 방법
   - 예시

```
<template>
        <div>
                <button @click="increment">Counter: {{ counter }}</button>
        </div>
</template>

<script>
export default {
        data() {
                return {
                        counter: 0,
                };
        },
        methods: {
                increment() {
                        this.counter++;
                },
        },
        mounted() {
                console.log('애플리케이션이 마운트 되었습니다!');
        },
};
</script>

<style lang="scss" scoped></style>
```

1-2. Composition API

2. Composition API
 - Options API의 단점을 보완하기 위해, Vue3에서 애플리케이션을 만들 수 있는 방법
 - Vue3에서는 Composition API 권장
 - 예시

```
<template>
        <div>
                <button>Counter: {{ counter }}</button>
        </div>
</template>

<script>
import { onMounted, ref } from 'vue';

export default {
        setup() {

                const counter = ref(0);
                const increment = () => counter.value++;

                onMounted(() => {
                    console.log('애플리케이션이 마운트 되었습니다!');
                });
                return {
                        counter,
                        increment,
                };
        },
};
</script>
<style lang="scss" scoped></style>
```

[1] **ref** 개념
 - Vue3 이전에는 뷰 템플릿의 DOM 또는 컴포넌트를 가리키는 속성 이었지만, Vue3에서는 **reactive reference**를 의미

[2] ref 사용 방법
① <script>에 import { ref } from "vue" 을 임포트.
② 변수를 선언하고 값을 ref()로 감싸줍니다.
③ 변수 값을 변경할 때 변수.value에 변경할 값을 적용

[3] **const** 의 경우 **배열이나 객체의 값**은 변경 가능

1-3. Composition API

[1] Composition API 개념

- Composition API는 옵션(data, methods, ...)을 선언하는 대신 가져온 함수(ref, onMounted, ...)를 사용하여 Vue 컴포넌트를 작성할 수 있는 API 세트

[2] Composition API 최대 장점 , 코드 재사용성

- Composable 함수의 형태로 로직의 재사용이 가능
- Composable 함수 의미 → Vue 컴포지션 API를 활용하여 **상태 저장 로직**를 **캡슐화**하고 재사용하는 함수
- Options API의 기본 로직 재사용 메커니즘인 Mixins의 모든 단점을 해결
- **hook**를 사용하여 관련 **코드 조각을 함께 그룹화**

[3] Composition API 유형

항목	주요기능
반응형 API (Reactivity API)	ref(), reactive()와 같은 API를 사용하여 reactive state(반응 상태), computed state(계산된 상태), watchers(감시자) 등을 생성
Lifecycle Hooks	onMounted(), onUnmounted()와 같은 API를 사용하여 프로그래밍 방식으로 컴포넌트 라이프 사이클에 접근
Dependency Injection	Provide()와 inject()는 Reactivity API를 사용하는 동안 Vue의 의존성 주입 시스템을 활용

[4] Lifecycle Hooks

① Lifecycle : Vue 인스턴스가 생성된 후 우리 눈에 보여지고, 사라지기까지의 단계
생성(**create**)되고, DOM에 부착(**mount**)되고, 업데이트(**update**)되며, 없어지는(**destroy**) 4가지 과정

② Lifecycle Hooks : 라이프사이클 단계에서 실행되는 함수
- site 참조 : https://vuejs.org/guide/essentials/lifecycle.html

2-1. Composition API Setup Hook

[1] Setup 개념
- setup() 함수(hook)는 **Composition API 사용**을 위한 **진입점**
- setup 함수는 컴포넌트 인스턴스가 생성되기 전에 실행

[2] **<template>**에 노출
- 반응형 API(Reactivity API)를 사용하여 반응형 상태를 선언
- setup()에서 객체를 반환

[3] setup 함수 Parameter

항목	주요기능
props	setup 함수의 **첫 번째** 매개변수 props는 반응형 객체
Setup Context 객체	함수에 전달된 **두번째** 매개변수 setup 함수내에서 유용하게 사용할 수 있는 속성 **Context** 객체는 반응형이 아니며 안전하게 구조 분해 할당

[4] **Setup Context** 객체 주요 속성
- attrs와 slots : 컴포넌트 자체가 업데이트될 때 항상 업데이트되는 상태 저장 객체

항목	주요기능
context.attrs	속성($attrs와 동일한 비반응형 객체)
context.slots	$slots에 해당하는 비반응성 개체
context.emit	이벤트 발생
context.expose	Public한 속성, 함수를 외부에 노출시에 사용, 노출되는 속성을 명시적으로 제한하는 데 사용할 수 있는 함수

2-2-1. Composition API Template Syntax

[1] Template Syntax 개념
- 템플릿 문법을 사용하여 렌더링된 DOM을 **컴포넌트의 인스턴스 데이터**에 선언적으로 바인딩
- **v-** 로 시작되는 요소들을 **디렉티브** 라 함

[2] 텍스트 **보간법**
- 데이터 바인딩의 가장 기본형태는 {{ data }} (이중 중괄호, 콧수염 괄호)를 사용
- v-once 디렉티브를 사용하여 데이터가 변경되어도 갱신(반응)되지 않는 일회성 보간을 수행

```
<p v-once>문자열: {{ message }} </p>
```

[3] HTML (v-html)

항목	내용
개념	이중 중괄호는 데이터를 HTML이 아닌 일반 텍스트로 해석 실제 HTML을 출력하려면 v-html 디렉티브를 사용 HTML 보간법은 반드시 신뢰할 수 있는 콘텐츠에서만 사용하고 사용자가 제공한 콘텐츠에서는 절대 사용금지(XSS취약점)
코드 예시	텍스트: {{ rawHtml }} v-html:

[4] v-bind

항목	내용
개념	이중 중괄호는 HTML 속성에 사용할 수 없습니다. 대신 **v-**bind **디렉티브**를 사용 **v-bind:** 또는 **:** 을 사용하여 **속성을 바인딩**
코드 예시	마우스를 올려보세요. <input type="text" v-bind:disabled="isInputDisabled" /> **또는** <div :title="dynamicTitle">마우스를 올려보세요. <input type="text" :disabled="isInputDisabled" />

2-2-2. Composition API Template Syntax

[5] 다중 속성 바인딩

항목	내용
개념	여러 속성을 한번에 바인딩

코드 예시	const attrs = {
	id: 'password-id',
	type: 'password',
	placeholder: '비밀번호를 입력해주세요'
	}
	
	<input v-bind="attrs" />

[6] JavaScript 표현식 사용

항목	내용
개념	모든 데이터 바인딩 내에서 JavaScript 표현식이 가능 , 함수도 호출
코드 예시	{{ isInputDisabled ? '예' : '아니오' }} {{ message.split('').reverse().join('') }} <input type="text" :placeholder="`입력해주세요 ${isInputDisabled}`" /> // 함수 호출 {{ toDay() }}

2-3-1. Composition API 반응형

[1] 반응형 상태 선언
- JavaScript 객체에서 **반응형 상태**를 생성하기 위해서는 reactive() 함수를 사용

[2] 선언 예시

```
import { reactive } from 'vue'

// 반응형 상태
const state = reactive({ count: 0 })
```

[3] setup에서 반응형 사용

항목	내용
적용	반응형 객체를 사용하려면 setup()함수에서 선언하고 리턴 컴포넌트의 data()에서 객체를 반환할 때, 이것은 내부적으로 reactive()에 의해 반응형으로 생성

코드 예시	import { reactive } from 'vue'
	
	export default {
	  setup() {
	
	  const state = reactive({ count: 0 })
	
	  return {
	  state
	  }
	  }
	}
	// HTML
	 {{ state.count }}

2-3-2. Composition API 반응형

[4] ref로 원시값 반응형 데이터 생성하기

- **reactive() 함수는 객체타입에만** 동작. 그래서 기본타입(number, string, boolean)을 반응형으로 만들고자 할 때 ref 메소드를 사용
- ref 메서드는 **변이가능**한(mutable) 객체를 반환. 이 객체 안에는 value라는 하나의 속성만 포함. value값은 ref() 메서드에서 매개변수로 받은 값을 가지고 있고, 객체는 내부의 value 값에 대한 반응형 참조(reference) 역할

예시	내용
Ref 선언	import { ref } from 'vue' const count = ref(0)

Ref value 예시	import { ref } from 'vue'
	
	const count = ref(0)
	console.log(count.value) // 0
	
	count.value++
	console.log(count.value) // 1

2-3-3. Composition API 반응형

[5] 반응형 객체의 ref Unwarpping
- ref가 반응형 객체의 속성으로 접근할 때, 자동적으로 내부 값으로 벗겨내서, 일반적인 속성과 마찬가지로 동작
- 이때 반응형은 연결

[6] 배열 및 컬렉션의 참조 Unwarpping
- 반응형 객체와 달리 ref가 반응형 배열 또는 Map과 같은 기본 컬렉션 타입의 요소로 접근될 때 수행되는 래핑 해제가 없음

```
const books = reactive([ref('Vue 3 Guide')])
// need .value
console.log(books[0].value)

const map = reactive(new Map([['count', ref(0)]]))
// need .value here
console.log(map.get('count').value)
```

[7] 반응형 상태 구조 분해하기(Destructuring)
- 큰 반응형 객체의 몇몇 속성을 사용하길 원할 때, 원하는 속성을 얻기 위해 ES6 구조 분해 할당을 사용하는 것은 매우 일반적

```
import { reactive } from 'vue'

const book = reactive({
  author: 'Vue Team',
  year: '2020',
  title: 'Vue 3 Guide',
  description: '당신은 이 책을 지금 바로 읽습니다 ;)',
  price: '무료'
})

let { author, title } = book
```

구조 분해로 두 속성은 반응형을 잃게 됨
이런 경우, 반응형 객체를 일련의 ref 들로 변환해야 함.
이러한 ref 들은 소스 객체에 대한 반응형 연결을 유지

2-3-4. Composition API 반응형

[8] 반응형 상태 구조 분해하기(Destructuring) - 반응형 객체의 속성과 동기화

- **toRefs, toRef를 사용**하면 반응형 객체의 **속성과 동기화**

```
import { reactive, toRefs } from 'vue'

const book = reactive({
  author: 'Vue Team',
  year: '2020',
  title: 'Vue 3 Guide',
  description: '당신은 지금 바로 이 책을 읽습니다 ;)',
  price: '무료'
})

let { author, title } = toRefs(book)

title.value = 'Vue 3 상세 Guide' // title 이 ref 이므로 .value 를 사용해야 합니다.
console.log(book.title) // 'Vue 3 Detailed Guide'
```

2-3-5. Composition API 반응형

[9] readonly를 이용하여 반응형 객체의 변경 방지
- 반응형 객체(ref나 reactive)의 변화를 추적하기 원하지만, 또한 특정 부분에서는 변화를 막기를 원하기도 함
- Provide/Inject로 주입된 반응형 객체를 갖고 있을 때, 우리는 그것이 주입된 곳에서는 해당 객체가 변이되는 걸 막고자 할 때 , 원래 객체에 대한 읽기 전용 프록시를 생성

```
import { reactive, readonly } from 'vue'
const original = reactive({ count: 0 })
const copy = readonly(original)

// 원본이 변이되면 복사본에 의존하는 watch 도 트리거됨
original.count++

// 복사본을 변이하려고 하면 경고와 함께 실패할 것 입니다.
copy.count++  // warning: "Set operation on key 'count' failed: target is readonly."
```

[10] 함수 이용한 변경 (compile-time transforms을 사용하면 적절한 위치에 .value를 자동으로 추가)
- compile-time transforms을 사용하면 적절한 위치에 .value를 자동으로 추가하여 인체 공학을 개선

```
<script setup>
let count = $ref(0)

function increment() {
  // no need for .value
  count++
}
</script>

<template>
  <button @click="increment">{{ count }}</button>
</template>
```

2-3-6. Composition API 반응형

[1] **Computed** 적용 개념

- 템플릿 문법({{ }})이 복잡해지며 여러곳에서 반복적으로 사용해야 한다면 더욱 비효율

  이럴 때 사용하는 것이 계산된 속성(computed property)

[2] Computed 적용 예시

```
const hasLecture = computed(() => {
    return teacher.lectures.length > 0 ? 'Yes' : 'No'
})
<p>강의가 존재 합니까?:</p>
<span>{{ hasLecture }}</span>
```

[3] computed와 메서드 공통점 / 차이점

- Computed는 기본적으로 getter전용. **계산된 속성에 새 값을 할당**하려고 하면 **런타임 경고**가 표시.

  새로운 계산된 속성이 필요한 경우에 getter와 setter를 모두 제공하여 속성을 생성해야 함

항목	computed	메서드
차이점	결과가 **캐시**	파라미터가 올 수 있음
장점	반응형 데이터가 변경된 경우에만 다시 계산 컴포넌트 랜더링시 비용이 적게듬	Writable Computed Computed는 기본적으로 getter전용
공통점	동일한 결과를 획득.	

2-3-7. Composition API 반응형

[1] 클래스와 Style 바인딩 적용 개념
- 클래스를 동적으로 바인딩 하기 위해선는 **:class(v-bind:class)**를 사용

[2] 배열에 바인딩 적용 예시

```
const activeClass = ref('active')
const errorClass = ref('text-danger')

<div :class="[activeClass, errorClass]"></div>
```

[3] 인라인 스타일 바인딩 적용 예시

```
const activeColor = ref('red')
const fontSize = ref(30)

<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

[4] 스타일 객체에 직접 바인딩

```
const styleObject = reactive({
  color: 'red',
  fontSize: '13px'
})

<div :style="styleObject"></div>
```

[5] 배열에 바인딩
- :style은 여러 객체 배열에 바인딩

```
<div :style="[baseStyles, overridingStyles]"></div>
```

2-3-8-1. Composition API 반응형

[1] 조건부 렌더링 개념
- v-if 디렉티브는 조건부로 블록을 렌더링 할 때 사용

```
<h1 v-if="visible">Hello Vue3!</h1>
```

항목	적용개념	예시
v-if	조건부로 블록을 렌더링 할 때 사용	<h1 v-if="visible">Hello Vue3!</h1>
v-else	v-if가 거짓(false)일 때 렌더링 하는 블록	<h1 v-if="visible">Hello Vue3!</h1> <h1 v-else>Good bye!</h1>
v-else-if	v-if에 대한 'else if 블록' 여러 조건을 연결	<h1 v-if="type === 'A'">A</h1> <h1 v-else-if="type === 'B'">B</h1> <h1 v-else-if="type === 'C'">C</h1> <h1 v-else>Not A/B/C</h1>
<**template** v-if="">	**여러개의 HTML요**소를 v-if 디렉티브로 연결	<**template** v-if="visible"> Title Paragraph 1 Paragraph 2 </template>
v-show	조건부로 표시하는 또 다른 옵션은 v-show 디렉티브	<h1 v-show="show">Title</h1> <button @click="show = !show">toggle show</button>

2-3-8-2. Composition API 반응형

[2] **v-if 대 v-show** 비교

항목	v-if	v-show
렌더링	실제(real)로 렌더링. 전환 할 때 블록 내부의 컴포넌트들이 제거되고 다시 생성	엘리먼트는 CSS 기반 전환으로 초기 조건과 관계 없이 항상 렌더링
Cost(비용)	전환 비용이 높음	초기 렌더링 비용이 높음
사용시기	런타임 시 조건이 거의 변경되지 않을 때	무언가를 자주 전환할 때

2] **v-if 대 v-for** 비교
- v-if와 v-for를 함께 사용할 때, v-if가 더 높은 우선순위
- v-if와 v-for를 함께 쓰는 것은 **권장 안 함**

2-3-9. Composition API 반응형

[1] 목록 렌더링 개념
- v-for 디렉티브는 배열 과 객체 블록을 렌더링 할 때 사용

항목	적용개념	예시
배열	배열인 목록을 렌더링	// Java Script const items = reactive([{ id: 1, message: 'Java' }, { id: 2, message: 'HTML' }, { id: 3, message: 'CSS' }, { id: 4, message: 'JavaScript' },]);
	// HTML <li v-for="(item, index) in tems" :key="item.id"> {{ item.message }} 	

객체	객체의 속성을 반복	// Java Script
		const myObject = reactive({
		title: '제목입니다.',
		author: '홍길동',
		publishedAt: '2016-04-10',
		});

	<li v-for="(value, key, index) in myObject" :key="key"> {{ key }} - {{ value }} - {{ index }} 	

2-3-10-1. Composition API 반응형

[1] 디렉티브(directives) 개념

- v-접두사가 있는 특수 속성
- 디렉티브 속성 값은 단일 JavaScript 표현식이 된다. (v-for는 예외)
- 디렉티브의 역할은 표현식의 값이 변경될 때 사이드 이펙트를 반응적으로 DOM에 적용

항목	주요기능
template	여러 개의 태그들을 묶어서 처리해야 할 경우에 template를 사용. v-if, v-for, component등과 함께 많이 사용
v-text	자바스크립트의 innerText와 같은 역할을 한다. html 태그가 적용되지 않고 문자열이 그대로 보여짐
v-bind	엘리먼트의 속성과 바인딩 처리를 위해서 사용, 약어로 ":"로 사용이 가능
v-html	자바스크립트의 innerHTML과 같은 역할을 하며 html 태그가 적용된 화면이 보여짐. 이중 중괄호(mustaches)는 HTML이 아닌 일반 텍스트로 데이터를 해석. 실제 HTML을 출력하려면 **v-html 디렉티브**를 사용
v-show	조건에 따라 엘리먼트를 화면에 렌더링. style의 display를 변경
v-for	배열이나 객체의 반복에 사용. 사용법은 v-for="요소변수이름 in 배열"이나 v-for="(요소변수이름, 인덱스) in 배열" 같은 방법으로 사용
v-once	데이터 변경 시 업데이트되지 않는 일회성 보간을 수행
v-cloak	Vue Instance가 준비될 때까지 mustache바인딩을 숨기는 데 사용
v-model	양방향 바인딩 처리를 위해서 사용, form의 input이나 textarea
v-on	DOM 이벤트를 듣고 트리거 될 때 JavaScript를 실행

2-3-10-2. Composition API 반응형

[2] 디렉티브(directives) 구성

예시] v-on:submit.prevent="onSubmit"

항목	표기법예시	주요구성요소
디렉티브(directives) Name	v-on	접두사가 있는 특수 속성으로 디렉티브의 값(value)이 변경될 때 특정 효과를 반응적으로 DOM에 적용
전달인자(Argument)	:submit	디렉티브명 뒤에 콜론(:)으로 표기되는 전달인자 동적 전달인자 : 대괄호를 사용하여 전달인자를 **동적으로 삽입**
수식어(Modifiers)	.prevent	점(.)으로 표시되는 특수 접미사로 **디렉티브**가 특별한 방식으로 바인딩
value	onSubmit	전달인자의 값