KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

Composition API

[KB] IT's Your Life

전까지는 과거 문법, 기초, vue2버전

지금부터가 실전 vue3버전에 새로운 내용.

전꺼는 이제 지원하지 않는다.

획기적인 형태

기존의 방법 createApp({})~~ 이런걸 옵션api라고 한다.

문제가 많았음

흩어져있는 느낌이 들고

타이핑 량이 많ㄹ음

KB 국민은행

1 Composition API란?

Composition API

- 대 규모 Vue 애플리케이션에서의 컴포넌트 로직을 효과적으로 구성하고 재사용할 수 있도록 만든 함수 기반의 API
- Vue3에서 새롭게 추가된 기능

- 지금까지 컴포넌트 작성 방법
  - data, methods, computed, watch와 같은 옵션들을 작성

→ 옵션 API(Options API)

1 Composition API란?

Options API 방식

- 논리적 관심사 코드가 분리됨
- 로직 재사용의 불편함

```
<script>
export default {
    name : "OptionsAPI",
    data() {
        return {
            name: "",
            x: 0, y: 0
        }
    },
    computed : {
        result() {
            return parseInt(this.x, 10) + parseInt(this.y, 10)
        }
    },
    mounted() {
        this.name = "john";
        this.x = 10;
        this.y = 20;
    },
    methods: {
        changeX(strX) {
            let x = parseInt(strX, 10);
            this.x = isNaN(x) ? 0 : x;
        },
        changeY(strY) {
            let y = parseInt(strY, 10);
            this.y = isNaN(y) ? 0 : y;
        },
        changeName(name) {
            this.name = name.trim().length < 2 ? "" : name.trim();
        }
    }
}
</script>
```

name 관련 데이터

calc 관련 데이터

calc 관련 계산된 속성

name 데이터 초기화

calc 데이터 초기화

calc 관련 메서드

name 관련 메서드

2 setup 메서드를 이용한 초기화 : 첫번째

setup() 메서드

- data, methods, computed 옵션이 사라짐
- 초기화 작업을 setup()에서 정의
- beforeCreate, created 단계에서 setup() 메서드가 호출

위 작업을 모두 setup에서 정의

- 반응성을 가진 상태 데이터, 계산된 속성, 메서드, 생명주기 훅을 작성
- 이 들을 객체 형태로 리턴

→ 작성한 데이터나 메서드를 템플릿에서 이용

프록시

vue2가
proxy만드는 책임이
vue임.

vue3에서는
무엇을 proxy로 만들어야 할 지 모름.
만드는 책임이 우리한테 있음

vue2에서는 지역 컴포넌트를 사용하려면 임포트를 해야하고, 컴포넌트 속성에 다가 등록을 해야함. 수정 실수하거나 불편함.

vue3에서는 임포트만 하면돼! 지역컴포넌트 쓰는게 편해졌

```
import { ref } from 'vue';
export default {
    name : "Calc",
    setup() {
        const x = ref(10);
        const y = ref(20);
        return { x, y }
    }
}
```

proxy 만드는코드
ref함수
반환은 proxy객체
근데 이제
해당 데이터를
감싼,,,

2 setup 메서드를 이용한 초기화

실습 프로젝트

```
npm init vue calc-component-test
cd  calc-component-test
npm install
```

2 setup 메서드를 이용한 초기화

src/components/Calc.vue

```
<template>
  <div>
    X : <input type="text" v-model.number="x" /><br />
    Y : <input type="text" v-model.number="y" /><br />
  </div>
</template>

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

export default {
  name: 'Calc',
  setup() {
    const x = ref(10);
    const y = ref(20);
    return { x, y };
  },
};
</script>
```

프록시를 생성하기 모듈은 항상 사용할 것이다

setup함수의 반환값은 템플릿에서도 사용해도 된다! 를 나타냄

2 setup 메서드를 이용한 초기화

src/App.vue

```
<template>
  <div>
    <Calc />
  </div>
</template>

<script>
import Calc from './components/Calc.vue';

export default {
  name:"App",
  components : { Calc }, 여기선 적용 안함
}
</script>
```

부모->자식
정보 전달 역할을하는
prop과

자식->부모의
event는
어떻게 바뀔까?

setup() 메서드의 매개변수

- 두 개의 인자
  - 부모 컴포넌트로부터 전달받는 속성(props)
  - 컴포넌트 컨텍스트(component context)
    - 기존 옵션 API에서 this에 해당
    - vue 인스턴스의 속성에 접근가능(예, emit())

vue2 -> vue3는

옵션 기반 -> 함수 기반
data: {}, props: {}, emits:{}, computed:{}
->
setup() ...

data옵션 => 변수 선언
method옵션=> 함수선언
props옵션=> 매개변수로
computed옵션 => computed 함수으로
watched옵셔 => 특정 형식인 watched함수로

3 반응성을 가진 상태 데이터

반응성 데이터 == proxy객체를 만드는 함수 2가지

- **ref() 함수를 이용한 상태 데이터 생성**
  - data 옵션에 해당
  - 기본 데이터 타입에 대한 반응성 데이터 생성

ref함수는 기본형 데이터를,
참조형 데이터를,
래핑하여
proxy객체를 만들때

number boolean string object...

- **reactive()를 이용한 상태**
  - 참조 데이터 타입에 대한 반응성 데이터 생성

주로 객체 또는 배열
데이터를 래핑하여
proxy객체를 만들때

참조형은 2가지 다
사용 가능한데
어떤걸로?

ref()

- 기본 타입(primitive type)의 값을 이용해 반응성을 가진 참조형 데이터 생성
- 해당 데이터를 스크립트에서 사용할 때는 x.value로 접근해야 함
  - .value를 사용하지 않고 바로 값을 대입하면 반응성을 잃어 버림

let x = ref(10); x는 참조형. proxy객체.
10이라는 데이터는 .value라는 프로퍼티에 들어감

해당 x변수에 대한 표현은

템플릿 파트와

스크립트 파트가

다르게 표현해야 한다.

템 : x
스크립트 : x.value

주의
스크립트에서
x로 접근하면 참조 잃겠쪼??
반응성도 잃겠죠?

src/components/Calc2.vue

```
<template>
  <div>
    X : <input type="text" v-model.number="x" />
    <br />
    Y : <input type="text" v-model.number="y" />
    <br />
    <button @click="calcAdd">계산</button><br />
    <div>결과 : {{ result }}</div>
  </div>
</template>
```

```
<script>
import { ref } from 'vue';

export default {
  name: 'Calc2',
  setup() {
    const x = ref(10);
    const y = ref(20);
    const result = ref(30);

    const calcAdd = () => {
      result.value = x.value + y.value;
    };  result=x+y 하면 안돼

    return { x, y, result, calcAdd };
  },
};
</script>
```

원래 옵션기반이었다면
methods :{
 calcAdd()~~
}

근데 이거는 method 말고 computed가 더 적당 computed는 어떻게 바뀔까

문맥들은 클로저에 저장되고 내보내져도 잘 작동되죠? 반환됐다는 것은 템플릿에서 사용가능 한다는 것

- App.vue

  import Calc from './components/Calc2.vue';

reactive()

- 배열, 객체 등 참조형에 대한 반응성을 가지도록 함

let state = reactive({x:10, y:3, result:30});

템플릿에서 스크립트에서 프로퍼티에 접근하려면 x,y,result

그냥 state.x state.y state.result처럼 객체처럼 접근하면 돼

하지만 state = 처럼 접근하면 안되겠쪼?

그래서 const 처리

const처리에 대한 것은 ref함수 반환값에도 적용하는 것도 마찬가지

const는 권장사항이
아니라 필수.

src/components/Calc3.vue

```
<template>
  <div>
    X : <input type="text" v-model.number="state.x" />
    <br />
    Y : <input type="text" v-model.number="state.y" />
    <br />
    <button @click="calcAdd">계산</button><br />
    <div>결과 : {{ state.result }}</div>
  </div>
</template>
```

템플릿에서도
객체처럼 표현

```
<script>
import { reactive } from 'vue';

export default {
  name: 'Calc3',
  setup() {
    const state = reactive({ x: 10, y: 20, result: 30 });

    const calcAdd = () => {
      state.result = state.x + state.y;
    };
    return { state, calcAdd };
  },
};
</script>
```

객체 리터럴을 래핑
proxy객체화

이거는 .value가 아닌
직접적으로 프로퍼티 명으로 접근

하지만 state= 로 접근하면 참조 잃겠쪄?
반응성도 잉겠쪄?

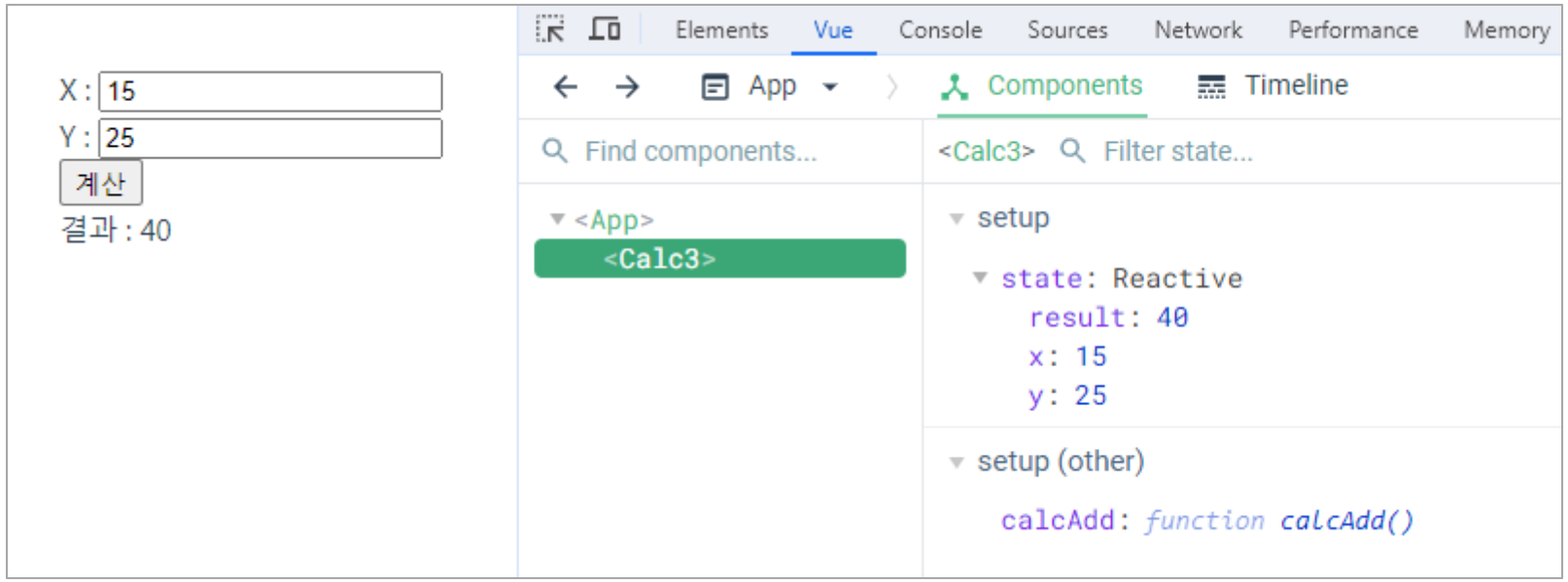
그래서 const로 지정한것도 있음

같은 작업을 한다면 ref reactive 중 뭘 사용할까
참조형 데이터 대상으로,,,

개인 취향이란다 ㅋㅋ

단 const x = ref({ a,b,c });면
x.value.a 이런식으로 접근해야함.
하지만 객체 데이터를 ref로 대하는게 편한경우가
가끔 있어 ㅇ App.vue

```
import Calc from './components/Calc3.vue';
```


X : 15
Y : 25
계산
결과 : 40
Elements
Vue
Console
Sources
Network
Performance
Memory
App
Components
Timeline
Find components...
<Calc3>
Filter state...
<App>
<Calc3>
setup
state: Reactive
result: 40
x: 15
y: 25
setup (other)
calcAdd: function calcAdd()

computed()

- 옵션 API에서 computed 옵션에 해당(계산된 속성)
- Composition API에서 계산형 속성을 만들 때 사용

```
import { computed } from 'vue';

const 속성명 = computed(()=> {
    ...
   return 값;
});
```

setup함수의 반환값에 포함시켜

src/components/Calc4.vue

```
<template>
  <div>
    X : <input type="text" v-model.number="state.x" />
    <br />
    Y : <input type="text" v-model.number="state.y" />
    <br />
    <div>결과 : {{ result }}</div>
  </div>
</template>
```

```
<script>
import { reactive, computed } from 'vue';

export default {
  name: 'Calc4',
  setup() {
    const state = reactive({ x: 10, y: 20 });
    const result = computed(() => {
      return state.x + state.y;
    });
    return { state, result };
  },
};
</script>
```

언제 또 자동으로 computed가 실행되어 바뀌어 적용되느냐

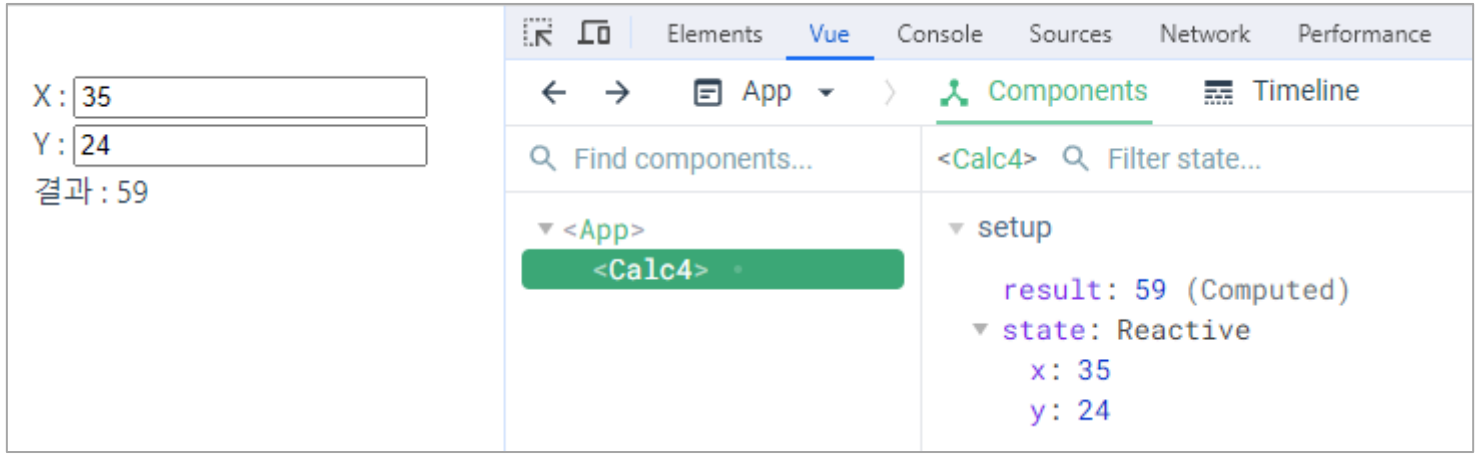
내부에서 사용되는 x, y 값이 바뀔때

- App.vue

  import Calc from './components/Calc4.vue';

또 다른 computed가 필요하면 같은 방식으로 하나 더 만들면 돼

4 computed()


X : 35
Y : 24
결과 : 59
Elements
Vue
Console
Sources
Network
Performance
App
Components
Timeline
Find components...
<Calc4>
Filter state...
<App>
<Calc4>
setup
result: 59 (Computed)
state: Reactive
x: 35
y: 24

watch

- watch() 함수를 통해서 제공
- 형식

첫번째 인자가 ref로 래핑된 것이면, current, old 인자값은 .value값이 대입

```
watch(data, (current, old) => {
    // 처리하려는 연산 로직
})
```

- 첫 번째 인자, data : 감시하려는 대상 반응성 데이터, 속성, 계산된 속성
- 두 번째 인자: 핸들러 함수
  - current: 변경된 값
  - old: 변경되기 전 값

  → current, old는 ref 객체가 아닌 ref.value에 해당하는 값 주의

src/components/Calc5.vue

```
<template>
  <div>
    x : <input type="text" v-model.number="x" />
    <br />
    결과 : {{result}}
  </div>
</template>
```

물론 해당 예시는
computed가 더 절절

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

export default {
    name:"Calc5",
    setup () {
        const x = ref(0);
        const result = ref(0);
        watch(x, (current, old)=>{
            console.log(`${old} -> ${current}`);
            result.value = current * 2;
        })
        return { x, result }
    }
}
</script>
```

current변수와
old변수 자체가
이미 .value이므로

- App.vue

  import Calc from './components/Calc5.vue';

src/components/Calc6.vue

넘어가 생명주기로

```
<template>
  <div>
    x : <input type="text" v-model.number="state.x" />
    <br />
    결과 : {{ state.result }}
  </div>
</template>
```

```
<script>
import { reactive, watch } from 'vue';

export default {
  name: 'Calc6',
  setup() {
    const state = reactive({ x: 0, result: 0 });
    watch(
      () => state.x,
      (current, old) => {
        console.log(`${old} -> ${current}`);
        state.result = current * 2;
      }
    );
    return { state };
  },
};
</script>
```

watch의 첫번째 인자를 주의해서 살펴보자

ref반환변수나 getter함수, effect함수, 등 특정된다

- App.vue

  import Calc from './components/Calc6.vue';

watchEffect

- Vue3에서 반응성 데이터 의존성을 추적하는 기능을 제공하는 새로운 방법

구분	watch	watchEffect
감시대상(의존성) 지정	필요함. 지정된 감시 대상 데이터가 변경되면 핸들러 함수가 실행됨	불필요함. 핸들러 함수 내부에서 이용하는 반응성 데이터가 변경되면 함수가 실행됨
변경전 값	이용가능. 핸들러 함수의 두 번째 인자값을 이용함.	이용 불가. 핸들러 함수의 인자 없음
감시자 설정 후 즉시 실행 여부	즉시 실행되지 않음	즉시 실행

- 기본 형식

```
watchEffect(()=>{
    // 반응성 데이터를 사용하는 코드 작성
})
```

src/components/Calc8.vue

```
<template>
  <div>
    X : <input type="text" v-model.number="x" /><br />
    Y : <input type="text" v-model.number="y" /><br />
    <div>결과 : {{ result }}</div>
  </div>
</template>

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

export default {
  name: 'Calc8',
  setup() {
    const x = ref(10);
    const y = ref(20);
    const result = ref(30);

    watchEffect(() => {
      result.value = x.value + y.value;
      console.log(`${x.value} + ${y.value} =
                   ${result.value}`);
    });

    return { x, y, result };
  },
};
</script>
```

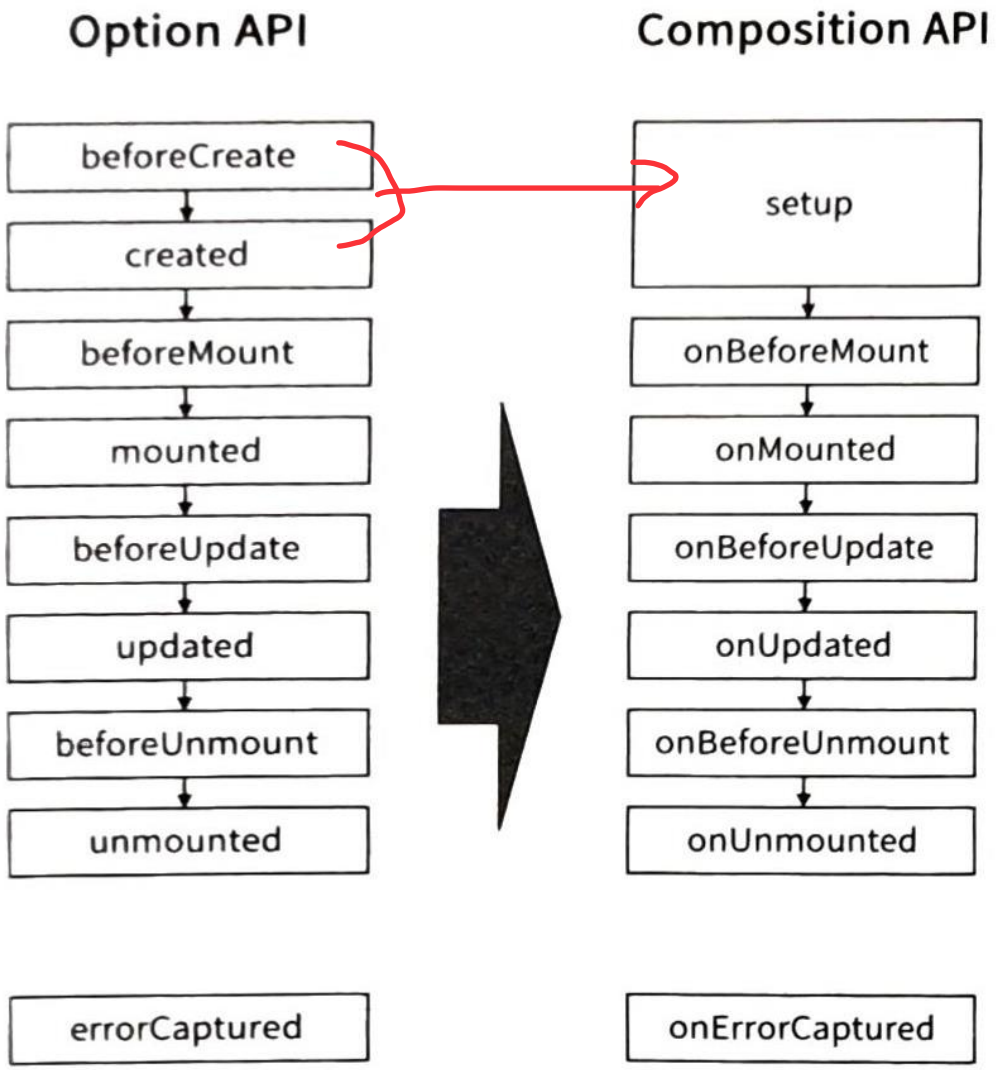
- App.vue

  import Calc from './components/Calc8.vue';

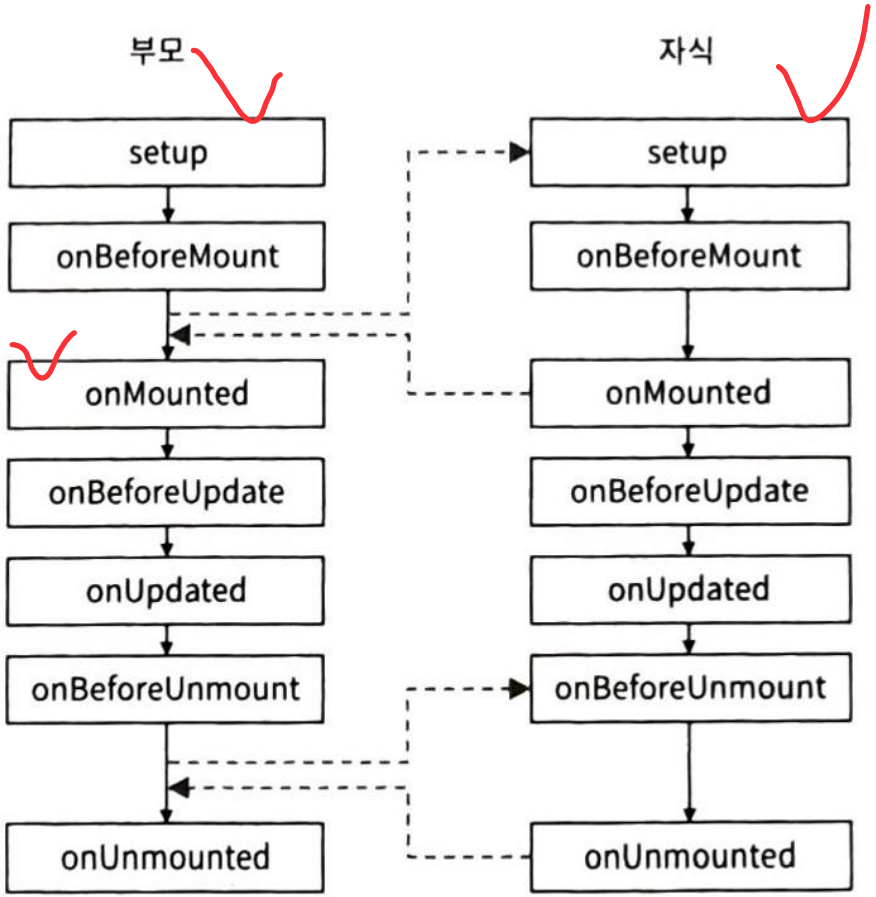
Composition API의 생명주기

vue3
함수기반

기존
vue2



부모 - 자식 관계일 때



옵션 API의 생명주기 메서드와 다른점

- beforeCreate(), created() 메서드의 기능을 setup()으로 대체
- 나머지 생명주기 메서드는 앞에 on 접두어를 붙인 함수로 변경됨
- 실행할 함수를 매개변수로 전달

```
setup() {
   // mounted
   onMounted(()=>{
     ...
   });
   ...
}
```

```
import { onMounted } from 'vue';
```

오늘 우리가 앞에서
했던 것은 setup만 만들었지?
그럴꺼면 간단하게 바꿀 수 없나?
setup구조를 다르게 하자 => script setup

```
template ...
style ...

<script>
import ~~

export default  {
...
setup{ ... } ...
}
```

->

```
<script setup>
...
```

한 단계 더 표현하는
방법

7 <script setup> 사용하기

KB 국민은행

- **<script setup>** setup 함수의 내부를 기재하는 것. 변환기가 알아서 동적으로 다시 만들어냄
  - 단일 파일 컴포넌트 내부에서 Composition API를 좀 더 편리한 문법적 작성 기능 제공
  - setup() 함수 내부 코드로 이용됨

  - 장점
    - 적은 상용구 코드 사용으로 간결한 코드 작성
    - 런타임 성능이 더 좋음
    - IDE에서의 타입 추론 성능이 더 뛰어남
    - 순수 타입스크립트 언어를 사용해 props, 이벤트를 선언할 수 있음

7 <script setup> 사용하기

태그 사이에서는
return이라는 개념이 없잖아.

script setup에서는

모든 식별자
(함수, 변수 등)
자동으로 모두 반환됨. return 됨.

그리고 임포트하고
사용할 컴포넌트들을
component로 등록해야하는데 그럴필요 없이
자동으로 등록해줌

<script setup>

- 템플릿에서 사용하는 값
  - 최상위의 변수, 함수는 직접 템플릿에서 사용 가능
- 지역 컴포넌트 등록
  - import만 하면 됨, components 속성 필요 없음
- 속성과 이벤트 처리 props와 emits 어떻게 바뀔까

```
//기존 방식
setup(props, context) {
   //이벤트를 발신할 때
   context.emit('add-todo', todo)
```

```
// <script setup> 방식
const props = defineProps({
   todoItem : { type : Object, required: true }
})
const emit = defineEmits(['delete-todo','toggle-completed'])
//이벤트를 발신할 때는 다음과 같이
emit('delete-todo', id)특정이벤트 호출
```

템플릿에서는 todoItem으로,
스크립트에서는 props.todoItem로 접근해야함