Reactivity Transform

실험적 기능 제거

Reactivity Transform은 실험적 기능이었으며, 최신 **3.4** 릴리스에서 제거되었습니다. 이유에 대해서는 여기에서 확인하십시오.

여전히 사용하고자 한다면, 이제 **Vue Macros** 플러그인을 통해 사용할 수 있습니다.

Composition API에 특화됨

Reactivity Transform은 **Composition API**에 특화된 기능이며 빌드 단계를 필요로 합니다.

Refs vs. Reactive Variables

Composition API 도입 이후, 주요한 미해결 질문 중 하나는 refs와 reactive 객체의 사용입니다. Reactive 객체를 구조분해할 때 리액티비티가 손실되기 쉽지만, refs를 사용할 때마다 `.value` 를 일일이 입력하는 것이 귀찮을 수 있습니다. 또한, 타입 시스템을 사용하지 않을 경우 `.value` 를 빠뜨릴 수도 있습니다.

Vue Reactivity Transform은 컴파일 시간에 변환되는 기능으로, 다음과 같은 코드를 작성할 수 있게 해줍니다:

vue

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

console.log(count)

function increment() {
  count++
}
</script>

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

여기서 `$ref()` 메소드는 **컴파일 시간 매크로**입니다. 이는 런타임에서 실제로 호출되는 메소드가 아닙니다. 대신 Vue 컴파일러는 이를 힌트로 사용하여 생성되는 `count` 변수를 **리액티브 변수**로 취급합니다.

리액티브 변수는 일반 변수처럼 접근하고 다시 할당할 수 있지만, 이러한 작업은 `.value` 가 포함된 refs로 컴파일됩니다. 예를 들어, 위 컴포넌트의 `<script>` 부분은 다음과 같이 컴파일됩니다:

js

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

let count = ref(0)

console.log(count.value)

function increment() {
  count.value++
}
```

refs를 반환하는 모든 리액티비티 API에는 `$` 로 시작하는 매크로도 있습니다. 이러한 API는 다음과 같습니다:

- **ref** -> `$ref`
- **computed** -> `$computed`

shallowRef -> $shallowRef
customRef -> $customRef
toRef -> $toRef

이러한 매크로는 전역적으로 사용할 수 있으며 Reactivity Transform이 활성화되면 가져올 필요가 없지만, 명시적으로 가져오고 싶다면 vue/macros 에서 가져올 수도 있습니다:

```js
import { $ref } from 'vue/macros'

let count = $ref(0)
```

$() 를 사용한 구조분해

컴포지션 함수가 refs 개체를 반환하고 구조분해를 사용하여 이러한 refs를 검색하는 것은 일반적입니다. 이를 위해 리액티비티 트랜스폼은 $() 매크로를 제공합니다:

```js
import { useMouse } from '@vueuse/core'

const { x, y } = $(useMouse())

console.log(x, y)
```

컴파일된 결과는 다음과 같습니다:

```js
import { toRef } from 'vue'
import { useMouse } from '@vueuse/core'

const __temp = useMouse(),
  x = toRef(__temp, 'x'),
  y = toRef(__temp, 'y')

console.log(x.value, y.value)
```

x 가 이미 ref인 경우 toRef(__temp, 'x') 는 그대로 반환되고 추가적인 ref는 생성되지 않습니다. 구조분해된 값이 ref가 아닌 경우(예: 함수)에도 동작합니다. 값은 ref로 감싸져 코드의 나머지 부분이 예상대로 작동합니다.

$() 구조분해는 리액티브 객체와 ref를 포함한 일반 객체 모두에서 작동합니다.

$() 를 사용하여 기존의 Ref를 리액티브 변수로 변환하기

일부 경우에는 Ref를 반환하는 함수를 가지고 있을 수 있습니다. 그러나 Vue 컴파일러는 함수가 Ref를 반환할 것임을 사전에 알 수 없습니다. 이러한 경우 $() 매크로를 사용하여 기존의 Ref를 리액티브 변수로 변환할 수도 있습니다:

```js
function myCreateRef() {
  return ref(0)
}

let count = $(myCreateRef())
```

리액티브 Props 구조분해

<script setup> 에서 defineProps() 를 사용하는 경우 두 가지 문제점이 있습니다:

.value 와 마찬가지로 리액티비티를 유지하기 위해 항상 props.x 로 접근해야 합니다. 이는 구조분해 결과 변수가 리액티브가 아니므로 구조분해할 수 없습니다.

타입만 있는 **Props** 선언을 사용할 때 props의 기본값을 선언하는 방법이 없습니다. 이를 위해 withDefaults() API를 도입했지만 여전히 사용하기 불편합니다.

이러한 문제를 해결하기 위해 구조분해와 함께 defineProps 가 사용될 때 컴파일 타임 변환을 적용할 수 있습니다. 앞서 $() 로 본 것과 유사하게:

html

```
<script setup lang="ts">
  interface Props {
    msg: string
    count?: number
    foo?: string
  }

  const {
    msg,
    // 기본 값은 잘 작동합니다.
    count = 1,
    // 로컬 별칭도 잘 작동합니다.
    // 여기서는 `props.foo`를 `bar`로 별칭 짓고 있습니다.
    foo: bar
  } = defineProps<Props>()

  watchEffect(() => {
    // 프롭이 변경될 때마다 로그가 기록됩니다.
    console.log(msg, count, bar)
  })
</script>
```

위의 코드는 다음과 같은 런타임 선언으로 컴파일됩니다:

js

```
export default {
  props: {
    msg: { type: String, required: true },
    count: { type: Number, default: 1 },
    foo: String
  },
  setup(props) {
    watchEffect(() => {
      console.log(props.msg, props.count, props.foo)
    })
  }
}
```

함수 경계를 넘어선 리액티비티 유지

리액티브 변수는 .value 를 모든 곳에 일일이 사용해야 하는 번거로움을 줄여줍니다. 하지만 리액티브 변수를 함수 경계를 넘어서 전달할 때 "리액티비티 손실" 문제가 발생합니다. 이는 두 가지 경우에 발생할 수 있습니다:

함수 인수로 전달하기

Ref를 인수로 사용하는 함수가 있다고 가정해보겠습니다:

```ts
function trackChange(x: Ref<number>) {
  watch(x, (x) => {
    console.log('x changed!')
  })
}

let count = $ref(0)
trackChange(count) // 동작하지 않음!
```

위의 경우는 예상한 대로 작동하지 않습니다. 이는 다음과 같이 컴파일됩니다:

```ts
let count = ref(0)
trackChange(count.value)
```

여기서 `count.value` 가 숫자로 전달되는 반면, `trackChange` 는 실제 ref를 기대합니다. 이를 해결하기 위해 `count` 를 전달하기 전에 `$$()` 로 감싸야 합니다:

```diff
let count = $ref(0)
- trackChange(count)
+ trackChange($$(count))
```

위의 코드는 다음과 같이 컴파일됩니다:

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

let count = ref(0)
trackChange(count)
```

`$$()` 는 리액티브 변수 내부에서는 `.value` 가 추가되지 않습니다.

함수 범위에서 반환하기

리액티비티는 리액티브 변수를 반환하는 표현식에서도 손실될 수 있습니다:

```ts
function useMouse() {
  let x = $ref(0)
  let y = $ref(0)

  // mousemove에 대한 이벤트를 청취...

  // 동작하지 않음!
  return {
    x,
    y
  }
}
```

위의 반환 문은 다음과 같이 컴파일됩니다:

```ts
return {
  x: x.value,
  y: y.value
}
```

리액티비티를 유지하려면 현재 값이 아닌 실제 refs를 반환해야 합니다.

다시 말해, $$() 를 사용하여 이 문제를 해결할 수 있습니다. $$() 는 리액티브 변수 내부에서 직접 사용할 수 있습니다. 리액티브 변수에 대한 참조는 소유 중인 ref에 대한 참조를 유지합니다:

ts

```ts
function useMouse() {
  let x = $ref(0)
  let y = $ref(0)

  // mousemove에 대한 이벤트를 청취...

  // 수정된 코드
  return $$({
    x,
    y
  })
}
```

구조분해된 props에서 $$() 사용하기

구조분해된 props에서도 $$() 를 사용할 수 있습니다. 구조분해된 props는 리액티브 변수이므로 잘 동작합니다. 효율성을 위해 컴파일러는 toRef 로 변환합니다:

ts

```ts
const { count } = defineProps<{ count: number }>()

passAsRef($$(count))
```

다음과 같이 컴파일됩니다:

js

```js
setup(props) {
  const __props_count = toRef(props, 'count')
  passAsRef(__props_count)
}
```

TypeScript 통합

Vue는 이러한 매크로에 대한 타이핑을 제공합니다(전역으로 사용 가능). 모든 타입은 예상대로 작동하기 때문에 표준 TypeScript 구문과 호환됩니다. 따라서 기존 도구에서도 작동합니다.

이는 유효한 JS / TS 파일이 있는 모든 곳에서 작동할 수 있는 매크로로, Vue SFC 내부뿐만 아니라 모든 파일에서 작동할 수 있습니다.

매크로가 전역으로 사용 가능하기 때문에, 그들의 타입을 명시적으로 참조해야 합니다(예: env.d.ts 파일에서):

ts

```ts
/// <reference types="vue/macros-global" />
```

vue/macros 에서 매크로를 명시적으로 가져올 때는 전역 변수를 선언하지 않아도 됩니다.

명시적으로 사용하기

코어에서 지원 중단

다음 내용은 Vue 버전 3.3 이하까지만 적용됩니다. Vue 코어 3.4 이상 및 @vitejs/plugin-vue 5.0 이상에서는 지원이 제거되었습니다. 변환을 계속 사용하려면 Vue Macros로 마이그레이션하는 것이 좋습니다.

Vite

@vitejs/plugin-vue@>=2.0.0 가 필요합니다.
SFC 및 js(x)/ts(x) 파일에 적용됩니다. 매크로를 사용하지 않는 파일에 대해 변환 적용 전에 빠른 사용 검사가 수행되므로 성능에 영향을 미치지 않습니다.
reactivityTransform 은 이제 SFC에만 중첩된 script.refSugar 대신 플러그인 루트 수준의 옵션으로 작동합니다.

```js
// vite.config.js
export default {
  plugins: [
    vue({
      reactivityTransform: true
    })
  ]
}
```

vue-cli

현재 SFC에만 영향을 미칩니다.
vue-loader@>=17.0.0 가 필요합니다.

```js
// vue.config.js
module.exports = {
  chainWebpack: (config) => {
    config.module
      .rule('vue')
      .use('vue-loader')
      .tap((options) => {
        return {
          ...options,
          reactivityTransform: true
        }
      })
  }
}
```

일반적인 webpack + vue-loader

현재 SFC에만 영향을 미칩니다.
vue-loader@>=17.0.0 가 필요합니다.

```js
// webpack.config.js
module.exports = {
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
        options: {
          reactivityTransform: true
        }
      }
    ]
```

```
  }
}
```