

Vue 3 at Keap

Making the world a better place by keeping up
with modern frontend frameworks /s



New to Vue 3

- Composition API
- Other new features
 - Teleport
 - Fragments (multiple root elements)
- Performance
- First-class Typescript support

New features

- Teleport
 - Similar to `portal-vue`, can move elements/components through the DOM
- Fragments
 - Multiple root elements in component
- Composition API
- Many others

Fragments example

BEFORE

```
<template>
  <div>
    <AppHeader />
    <AppMain />
    <AppFooter />
  </div>
</template>
```

AFTER

```
<template>
  <AppHeader />
  <AppMain />
  <AppFooter />
</template>
```

Composition API

Example Component

```
<template>
  <button @click="increment">+</button>
  <span>Count^2 {{ countSquared }}</span>
</template>
```

Multiple root nodes! 🎉

Options API vs. Composition API

OPTIONS API

```
export default {
  data() {
    return { count: 0 }
  },
  computed: {
    countSquared() {
      return this.count ** 2
    }
  },
  methods: {
    increment() {
      this.count += 1
    }
  }
}
```

COMPOSITION API

```
import { computed, ref } from 'vue'
export default {
  setup() {
    const count = ref(0)
    const increment = () => count.value += 1
    const countSquared = computed(
      () => count.value ** 2
    )

    return {
      count,
      countSquared,
      increment
    }
  }
}
```

Sure, that's great, it can do the
same thing.

What makes it better?

Composables

Consider a pattern that is used frequently: managing tooltip behavior.

```
export default {
  data() {
    return { isOpen: false }
  },
  methods() {
    toggleTooltip() {
      this.isOpen = !this.isOpen
    }
  }
}
```

```
// tooltip.js
export const useTooltip = () => {
  const isOpen = ref(false)
  const toggle = () => isOpen.value = !isOpen.value

  return { isOpen, toggle }
}

// Component.vue
import useTooltip from './tooltip'
export default {
  setup() {
    const { isOpen, toggle } = useTooltip

    // Other component logic

    return { isOpen, toggle }
  }
}
```

Separation of concerns

Imagine a component that has a header with a tooltip and also needs to render a list of things based on an API call.

OPTIONS API

```
data() {  
  return {  
    isOpen: false,  
    loading: false,  
    items: []  
  }  
}
```

```
computed: {  
  sortedItems() { ... }  
  tooltipTitle() { ... }  
}
```

```
methods: {  
  async loadItems() { ... }  
  toggleTooltip() { ... }  
}
```

COMPOSITION API

```
setup() {  
  // ITEMS  
  const loading = ref(false)  
  const items = ref([])  
  const sortedItems = computed(() => { ... })  
  const loadItems = () => { ... }  
  mounted(() => loadItems())  
  
  // TOOLTIP  
  const isOpen = ref(false)  
  const tooltipTitle = computed(() => { ... })  
  const toggleTooltip = () => { ... }  
  
  // Only return the items used in the template  
  // e.g. Don't return loadItems or items  
  return { ... }  
}
```

`<script setup>`

more info

```
<script setup>
import { ref, computed } from 'vue'

const count = ref(0)
const countDoubled = computed(() => count.value * 2)
const increment = () => count.value += 1
</script>
```

All of these constants become available in the template.

Better Vue Tooling

- Enhanced Vue DevTools extension
- Better intelliense
- Vetur
- Volar

Performance

- Smaller bundle size due to treeshakeability
- Mounts quicker
- Updates quicker
- Uses less memory
- Spreadsheet with details

First-class Typescript support

Why Typescript?

- More reliable/higher confidence in code
- Typing acts as documentation
- Better tooling/intellisense
- Easier to maintain with high amount of developers

Sure, but why Typescript in
Vue?

Better type safety for props

VUE 2

```
props: {
  broadcast: {
    type: Object,
    required: true,
    // default: () => ({}),
  },
}
```

```
props: {
  broadcast: {
    type: Object,
    required: true,
    validator: (value) => {
      ['id', 'scheduleDate'].forEach((key) => {
        broadcast.hasOwnProperty(key)
      })
    },
    // But what about the type of the values in broadcast?
  },
}
```

VUE 3 + TYPESCRIPT

```
import { PropType } from 'vue'

type Broadcast = {
  id: string | number;
  scheduleDate: string;
}
```

```
props: {
  broadcast: {
    type: Object as PropType<Broadcast>,
    required: true,
  }
}
```

```
// Using <script setup> syntactic sugar
type Broadcast = {
  id: string | number;
  scheduleDate: string;
}
```

```
const { broadcast } = defineProps<{

  broadcast: Broadcast
}
```

Type safety for event emissions

Consider a component that emits a value that is selected by the user

```
export default defineComponent({
  emits: {
    select(payload: { value: string; label: string; }) { ... }
  },
  setup(props, { emit }) {
    const onClick = (() => {
      emit('select', { somethingElse: 0 }) // throws compilation error
    })

    return { onClick }
  }
})
```

Great, sign me up!

Migration

1. Remove deprecated syntax for slots
2. Upgrade to compat version (Vue 3.1)
3. Resolve incompatible and partially compatible issues
4. Upgrade related dependencies (vuex, vue-router, vue-i18n)
5. Refactor app and components over time to remove features removed from Vue 3
6. Update eslint plugins to vue3 plugins

Full migration guide

Additional information for Keap's migration