# The new composition API

*A world of possibilities*

# The new composition API

A world of possibilities

Thorsten Lünborg

Vue.js Core Team

GitHub: LinusBorg
Twitter: @linus_borg

An warning upfront…

We’re gonna see ….

ALOT of code

© Allie Brosh / hyperboleandahalf.blogspot.com.au

but don’t worry …

# Demo project on Github

Repo will be published after the talk!

http://bit.ly/vue-ldn-demos

**https://github.com/LinusBorg/composition-api-demos**

- base for all demos of this talk
- more extensive code, e.g. error handling
- additional examples

## What's that *composition API*?

that's a lot of functions!

```
import {

  setup,

  ref,
  isRef,
  readonly,
  reactive,
  toRefs,
  computed,
  watch,

  onBeforeMount,
  onMounted,
  onBeforeUpdate,
  onUpdated,
  onBeforeDestroy,
  onDestroyed,

  provide,
  inject,

} from 'vue'

/* Vue 2 Plugin: */
import {
  /* .. */
} from '@vu/composition-api'
```

„Portable" Reactivity

Dynamic Lifecycle methods

Sharing State

## #1 „Portable“ Reactivity

# Is this thing still loading?

1. Always the same pattern
2. properties, lifecycle and method loosely connected
3. **`this`** is all over the place

```
export default {
  data: () => ({
    loading: null,
    error: null,
    result: [],
  }),
  created() {
    this.loadData
  },
  methods: {
    // click handler for a button
    async loadData() {
      this.loading = true
      let result
      try {
        result = await api.getUsers()
      } catch (error) {
        this.error = error
      } finally {
        this.loading = false
      }
      this.result = result
    },
  },
})
```

## Enter: composition

```
import usePromiseFn from './composables/use-
promise-fn'
import api from '../api'

export default {
  setup() {

    const getUsers = usePromiseFn(
      () => api.getUsers()
    )

    return {
      getUsers,
    }
  },
}
```

## Enter: composition

```
import usePromiseFn from './composables/use-
promise-fn'
import api from '../api'

export default {
  setup() {

    /**
     * @type {{
     *   loading: boolean
     *   error: Error<any>
     *   result: any
     *   call: () => Promise<any>
     * }}
     */
    const getUsers = usePromiseFn(
      () => api.getUsers()
    )

    getUsers.use()

    return {
      getUsers,
    }
  },
}
```

./src/components/MyComponent.vue

```
export default {
  data: () ⇒ ({
    loading: null,
    error: null,
    result: [],
  }),
  created() {
   this.loadData
  },
  methods: {
    // click handler for a button
    async loadData() {
      this.loading = true
      let result
      try {
        result = await api.getUsers()
      } catch (error) {
        this.error = error
      } finally {
        this.loading = false
      }
      this.result = result
    },
  },
})
```

./src/composables/use-promise.js

```
import { reactive, toRefs } from '@vue/composition-
api'

export default function usePromise(fn) {
  const state = reactive({
    loading: false,
    error: null,
    result: null,
  })

  return {
    ... toRefs(state),
  }
}
```

./src/components/MyComponent.vue

```js
export default {
  data: () => ({
    loading: null,
    error: null,
    result: [],
  }),
  created() {
   this.loadData
  },
  methods: {
    // click handler for a button
    async loadData() {
      this.error = null
      this.loading = true
      this.result = []
      let result
      try {
        result = await api.getUsers()
      } catch (error) {
        this.error = error
      } finally {
        this.loading = false
      }
      this.result = result
    },
  },
})
```

./src/composables/use-promise.js

```js
import { reactive, toRefs } from '@vue/composition-api'

export default function usePromise(fn) {
  const state = reactive({
    loading: false,
    error: null,
    result: null,
  })

  const use = async (...args) => {
    state.error = null
    state.loading = true
    state.result = []
    try {
      const result = await fn(...args)
      state.result = result
    } catch (e) {
      state.error = e
    } finally {
      state.loading = false
    }
  }
  return {
    ...toRefs(state),
    use,
  }
}
```

# The Advantages:

1. reactive properties available with one line
2. Properties and method namespaces in one object
3. no lifecycle necessary, just call it
4. **`this`** is nowhere to be seen

```js
import usePromiseFn from './composables/use-promise-fn'
import api from '../api'

export default {
  setup() {

    /**
     * @type {{
     *   loading: boolean
     *   error: Error<any>
     *   result: any
     *   call: () => Promise<any>
     * }}
     */
    const getUsers = usePromiseFn(
      () => api.getUsers()
    )

    getUsers.use()

    return {
      getUsers,
    }
  },
}
```

and here's the thing…..

## These functions can be used *everywhere*!

- Components
- directives
- state management (more on that later)
- your own Javascript module

```
import { reactive } form '@vue/reactivity'
```

## More Examples

# Pagination

./src/components/Paginated.vue

```
import usePagination from './use-pagination'
export default {
  setup() {
    /**
     * @type {{
     *   perPage: Ref<number>
     *   total: Ref<number|null>
     *   currentPage: ReadOnly<number>
     *   lastPage: ReadyOnly<number>
     *   offset: Readonly<number>
     *   next: () ⇒ void
     *   prev: () ⇒ void
     *   first: () ⇒ void
     *   last: () ⇒ void
     *   set: (number) ⇒ void
     * }}
     */
    const pagination = usePagination({
      perPage: 10,
    })

    return { pagination }
```

# Form Validation

./src/components/Form.vue

```
import useValidation from './use-validation'
const rules = {
  // … left out for brevity
}
export default {
  setup() {
    const person = reactive({
      firstName: null,
      lastName: null,
    })
    /**
     * @type {{
     *   dirty: boolean
     *   valid: boolean
     *   errors: object
     * }}
     */
    const validation = useValidation(person, rules)
    return {
      person,
      ...toRefs(personValidation),
    }
```

Implementation on Github

## #2 Dynamic Lifecycle hooks

## Scroll Handling

### A typical usecase

1. verbose and repetitive
2. Code in 3 different „locations“
3. we need to do this quite often:
    - Global Events
    - Timers & Intervals
    - Wrapping 3rd-party libs

```
export default {

  mounted() {
    window.addEventListener('scroll',
     this.handleScroll
    )
  },

  beforeDestroy() {
    window.removeEventListener('scroll',
      this.handleScroll
    )
  },

  methods: {
    handleScroll(event) {
      /* ... */
    },
  },

}
```

# Scroll Handling

## A typical usecase

1. short and concise
2. handler doesn’t have to be a component method
3. easily extendable

```
import useEvent from './composables/use-event'

export default {
  setup() {
    useEvent('scroll', event => {
      /* ... */
    })
  },
}
```

# Extracting reusable behaviour

./src/components/MyComponent.vue

```
export default {

  mounted() {
    window.addEventListener('scroll',
      this.handleScroll
    )
  },

  beforeDestroy() {
    window.removeEventListener('scroll',
      this.handleScroll
    )
  },

  methods: {
    handleScroll(event) {
      /* ... */
    },
  },

}
```

./src/composables/use-event.js

```
import {
  onMounted,
  onBeforeDestroy,
} from '@vue/composition-api'

export function useEvent(name, handler, el = window) {

}
```

# Extracting reusable behaviour

./src/components/MyComponent.vue

```js
export default {

  mounted() {
    window.addEventListener('scroll',
     this.handleScroll
    )
  },

  beforeDestroy() {
    window.removeEventListener('scroll',
      this.handleScroll
    )
  },

  methods: {
    handleScroll(event) {
      /* ... */
    },
  },

}
```

./src/composables/use-event.js

```js
import {
  onMounted,
  onBeforeDestroy,
} from '@vue/composition-api'

export function useEvent(name, handler, el = window) {

  onMounted(
    () => el.addEventListener(name, handler)
  )

  onBeforeDestroy(
    () => el.removeEventListener(name, handler)
  )
}
```

## Usage

./src/components/MyComponent.vue

```
import useEvent from './composables/use-event'

export default {
  setup() {
    useEvent('scroll', event => {
      /* ... */
    })
  },
}
```

## Implementation

./src/composables/use-event.js

```
import {
  onMounted,
  onBeforeDestroy,
} from '@vue/composition-api'

export function useEvent(name, handler, el = window) {

  onMounted(
    () => el.addEventListener(name, handler)
  )

  onBeforeDestroy(
    () => el.removeEventListener(name, handler)
  )
}
```

# Extending by Composition

1. Listen to scroll event

2. Watch scroll position

3. Do something when position is reached

```
import useScroll from './composables/use-scroll'
import { watch, reactive } from 'vue'

export default {
  setup() {
    const stuff = reactive([])

    const { scrollY } = useScroll()

    watch(scrollY, y => {
      //checking if we reached end of page
      if (isBottomOfPage(y)) {

        stuff.push( /* ... */)

      }
    })
    return {
      stuff,
    }
  },
}
```

# Extending by composition

./src/composables/use-scroll.js

```js
import useEvent from './use-event'
import { throttle } from 'lodash-es'
import { ref } from 'vue'

export default function useScroll() {

  const scrollY = ref(null)
  const scrollX = ref(null)

  const doc = document.documentElement
  const handler = throttle(() => {
    scrollY.value = doc.scrollTop
    scrollX.value = doc.scrollLeft
  }, 50)

  useEvent('scroll', handler, window)

  return {
    scrollX,
    scrollY,
  }
}
```

1️⃣ Set up our state

2️⃣ Define event handler

3️⃣ usEvent with that handler

# Extending by composition

./src/composables/use-scroll.js

```javascript
import useEvent from './use-event'
import { throttle } from 'lodash-es'
import { ref } from 'vue'

export default function useScroll() {

  const scrollY = ref(null)
  const scrollX = ref(null)

  const doc = document.documentElement
  const handler = throttle(() => {
    scrollY.value = doc.scrollTop
    scrollX.value = doc.scrollLeft
  }, 50)

  useEvent('scroll', handler, window)

  return {
    scrollX,
    scrollY,
  }
}
```

Code is „lifecycle-aware“

=

No more worrying about lifecycle hooks!

## #3 Composition in components

## Demo Time

Home | Form Validation | Paginated Posts | Infinite Images | Upload Dropzone

Load more Images

# Infinite Scroll

## Ugly Pinterest

```
<template>
  <div>
    <div class=„grid">

      <article v-for="(photo, i) in photos" :key="i">
        <DemoImage :src="photo.url" />
      </article>

    </div>

    <div v-if="loading"><Spinner /></div>

    <button
      v-else-if="!loading"
      @click=„next">
      Load more Images
    </button>

  </div>
</template>
```

# Infinite Scroll

## Ugly Pinterest

```
export default {
  setup() {

  },
}
```

# Infinite Scroll

## Ugly Pinterest

```
export default {
  setup() {

          1️⃣ Set up paginated
             API call

          2️⃣ Prepare Tracking
             of API Promise

          3️⃣ Set up Pagination

          4️⃣ Fn to call API

          5️⃣ Detect end of Page
             -> next()

  },
}
```

./src/composables/InfiniteImages.vue

```
setup() {
  const photos = reactive([])

  const _loadImages = async (offset, perPage) ⇒ {
    const result = await api.photos.get({
      start: offset,
      limit: perPage,
    })
    photos.push( ... result)
  }

  const { loading, error, use: loadImages } = usePromiseFn(_loadImages)

  const pagination = usePagination({ perPage: 9 })

  function next() {
    if (loading.value) return
    pagination.next()
    loadImages(pagination.offset.value, pagination.perPage.value)
  }

  useEndOfPage(next, 150 /* px from bottom */)

  return {
    photos,
    currentPage: pagination.currentPage,
    error,
    loading,
```

Implementation on Github

1. Set up paginated API call
2. Prepare Tracking of API Promise
3. Set up Pagination
4. Fn to call API
5. Detect end of Page -> next()

## #4 Sharing state

## Provide/inject
## Now with more awesome

- Work quite like in Vue 2 in principle
- But: portable Reactivity improves usefulness exponentially
- Together: custom state management easy as 🎂

## Writing our own „DIY Vuex“

```
import {
  reactive,
  readonly,
  computed,
} from 'vue'

const state = reactive({
  messages: [],
})

const actions = {
  addMessage: message ⇒ {
    state.messages.push(message)
  },
}

const getters = {
  unread: computed(() ⇒
    state.messages.filter(message ⇒ !message.read)
  ),
}


export default {
  state: readonly(state),
  ...actions,
  ...getters,
}
```

## Using our store

./src/components/App.vue

```
<template>
  <div><router-view /></div>
</template>

<script>
import store from './store'
import { provide } from 'vue'

export default {
  setup() {
    provide(Symbol.for('MessageStore'), store)
  },
}
</script>
```

./src/composables/MessageIndicator.vue

```
<template>
  <div>You have {{ unread.length }} messages</div>
</template>
<script>
import { inject } from 'vue'
export default {
  setup() {

    const {
      state: { messages },
      addMessage,
      unread,
    } = inject(Symbol.for('MessageStore'))

    return {
      messages,
      addMessage,
      unread,
    }
  },
}
</script>
```

## Wrapping up

# Composition opens up
# a World of possibilities

1. Reactivity can be used everywhere
2. Sharing state is more powerful then ever
3. code can be lifecycle-aware
4. Composition opens up new patterns for writing components

Try it out ...

Get creative....

I HAVE NO IDEA WHAT I'M DOING
DANGER!
PICK ONE

## Demo project on Github

[http://bit.ly/vue-ldn-demos](http://bit.ly/vue-ldn-demos)

[https://github.com/LinusBorg/composition-api-demos](https://github.com/LinusBorg/composition-api-demos)

- ❖ base for all demos of this talk
- ❖ more extensive code, e.g. error handling
- ❖ additional examples

# Thank you!