Vue Composition-API

Introduction to composition-api

Carlos Rodrigues

Freelancer: VueJS & C# Developer

Github: @pikax
Medium: @pikax.dev

What is composition-api?

- “A set of additive, function-based APIs that allow flexible composition of component logic.” - source RFC
- Similar to mixins but without name collision.
- First class @Typescript support

@vue/composition-api

- Library to use the new API in the Vue 2.6
- **Not production** ready
- Limited by Vue 2.6

Fetch example

Implementing simple fetching component, it should handle loading, show error and result.

```
<div>
  <div v-if="loading">loading...</div>
  <div v-else-if="error">Error: {{error}}</div>
  <div v-else>{{ result }}</div>
</div>
```

Fetch example

Traditional Vue2.x

Observations

- Boilerplate code
- We need to copy & paste to reuse in other components

```
export default {
  props: {
    url: String
  },
  data: {
    loading: false,
    result: null,
    error: null
  },
  methods: {
    async fetch() {
      this.loading = true;
      this.error = null;
      try {
        this.result = await fetch(this.url);
      } catch (e) {
        this.result = null;
        this.error = e;
      } finally {
        this.loading = false;
      }
    }
  }
};
```

Mixin to the rescue...

- Mixin will solve the previous issues, seem to be the perfect solution for the problem... but is it tho?
  - Multiple `loading` properties in the component.
  - Same property/method name in the component.

```
const fetchMixin = {
  data() {
    return {
      loading: false,
      result: null,
      error: null
    };
  },
  methods: {
    async fetch(url) {
      this.loading = true;
      this.error = null;
      try {
        this.result = await fetch(url);
      } catch (e) {
        this.result = null;
        this.error = e;
      } finally {
        this.loading = false;
      }
    }
  }
};
```

```
const fetchMixin = {
  data() {
    return {
      fetchResult: {
        loading: false,
        result: null,
        error: null
      }
    };
  },
  methods: {
    async fetch(url) {
      this.fetchResult.loading = true;
      this.fetchResult.error = null;
      try {
        this.fetchResult.result = await fetch(url);
      } catch (e) {
        this.fetchResult.result = null;
        this.fetchResult.error = e;
      } finally {
        this.fetchResult.loading = false;
      }
    }
  }
};
```

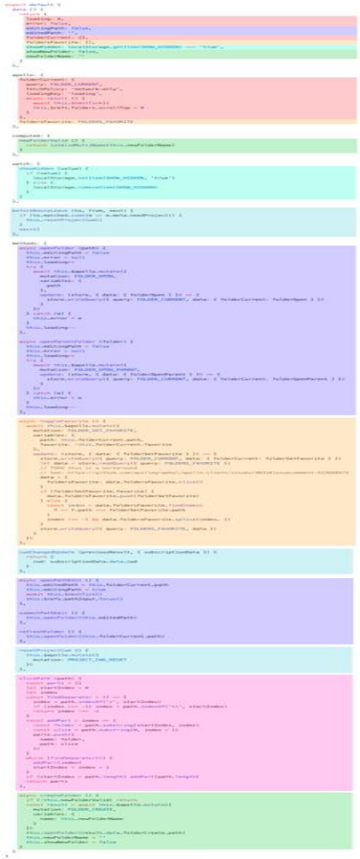
Multiple fetch with mixin?

- No can do

Composition-API motivation

- Logic Reuse & Code Organisation
  - Allow code to be reused across multiple components
  - Allow to restructure large components to have better organise code.

Options API



Composition API

API introduction

- **reactive** creates an reactive object similar to **data** in v2
- **ref** used to create reactive object against non-object variables. E.g: number, string
- **computed** used to create computed properties
- **watch**
- Lifecycle Hooks
  - onMounted
  - onUnmounted
  - onBeforeMount
  - etc...

```
import { ref, reactive, computed, watch } from "@vue/composition-api";
import { ref, reactive, computed, watch, readonly } from "@vue/reactivity";
```

Composition-api **setup** method

- Specifying all the **methods** and **data** required for the template.
- Only runs once
  - After **beforeCreate**
  - Before **created**
- **Params**:
  - Props
  - Context: **Vue instance**

```
import { ref, onMounted, onUnmounted } from "@vue/composition-api";

export default {
  setup(props, context) {
    const x = ref(0);
    const y = ref(0);

    const handleMouseMove = e => {
      x.value = e.x;
      y.value = e.yl;
    };

    onMounted(() => window.addEventListener("mousemove", listener));
    onUnmounted(() => window.removeEventListener("mousemove", listener));

    return {
      x,
      y
    };
  }
};
```

Composition-API fetch example

All the properties and methods used in the template are returned.

- **Ref** will be **unwrapped** in the template.
- Pretty generic code, we can move this logic to other place.

```
import { ref } from "@vue/composition-api";
export default {
  setup(props, context) {
    const loading = ref(false);
    const error = ref(null);
    const result = ref(null);

    const exec = async url => {
      loading.value = true;
      error.value = null;
      result.value = null;

      try {
        result.value = await fetch(url);
      } catch (error) {
        result.value = null;
        error.value = error;
      } finally {
        loading.value = false;
      }
    };

    return {
      loading,
      error,
      exec,
      result
    };
  }
};
```

Composable Fetch

- We can create a generic composable function that we can use in our components.

```
import { ref } from "@vue/composition-api";
function useFetch() {
  const loading = ref(false);
  const error = ref(null);
  const result = ref(null);
  const exec = async url => {
    loading.value = true;
    error.value = null;
    result.value = null;
    try {
      result.value = await fetch(url);
    } catch (error) {
      result.value = null;
      error.value = error;
    } finally {
      loading.value = false;
    }
  };
  return {
    loading,
    error,
    exec,
    result
  };
}
```

useFetch

- Now we can just call our **useFetch** and extract all the needed properties out of it.
- It allows to use it multiple times in the same component.

```
export default {
  setup(props, context) {
    const {
      loading,
      error,
      exec,
      result
    } = useFetch();

    const doFetch = () =>
exec("https://swapi.co/api/people");

    return {
      loading,
      error,
      doFetch,
      result
    };
  }
};
```

How do you use many?

```
export default {
  setup(){
    const fetchPeople = useFetch();
    const peopleExec = () => exec("https://swapi.co/api/people");

    const fetchPlanets = useFetch();
    const planetsExec = () => exec("https://swapi.co/api/planets");

    return {
      people: {
        ...fetchPeople,
        exec: peopleExec
      },
      planets: {
        ...fetchPlanets,
        exec: planetsExec
      }
    }
  }
}
```

Observations

- Composition API is additive, the options API will still be valid
- Typescript support is much better
- The reactivity will allow to do some interesting things
- Some limitations when using **@vue/composition-api**
- Is there any **vue-composable** library!?

Composition-api libraries

- Awesome List: https://github.com/kefranabg/awesome-vue-composition-api
  - vue-use-web - 🕸️ Web APIs implemented as Vue.js composition functions
  - vue-use-form - ✔️ A Vue.js composition API function to validate forms
  - vue-compose-promise - 💝 Promises using vue composition API
  - vue-composition-toolkit - 💚 Vue3 Composition-API toolkit
  - vue-composable - 💚 Vue composition-api composable components
  - vuses - 💚Vue Hooks built with Vue Composition Api

Questions?