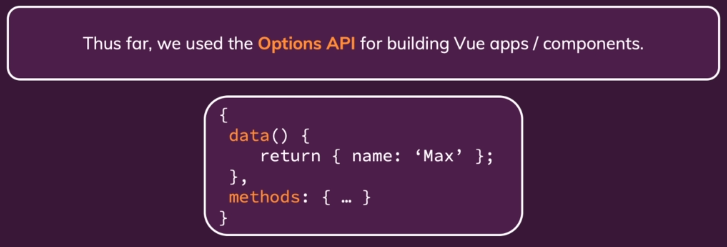
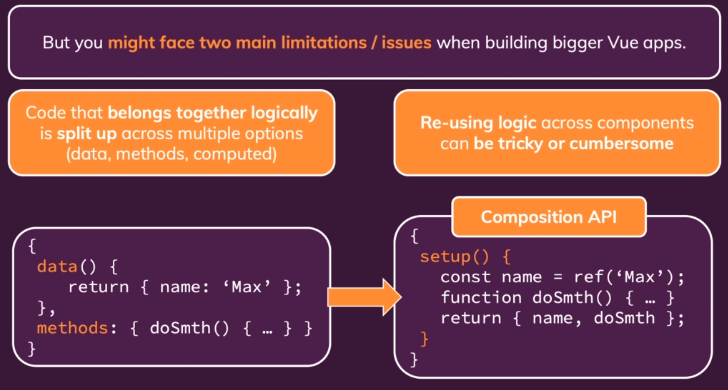
**Section 19 – The Composition API Replacing The Options API**

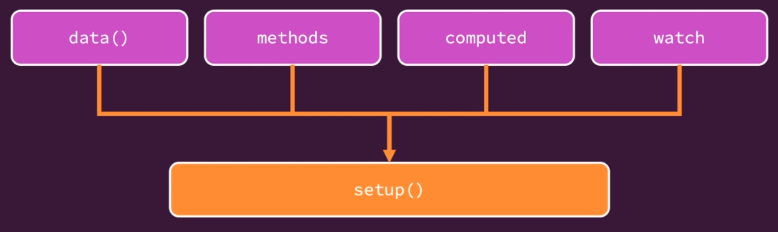
Chapter 279 – Which Problem Does the Composition API Solve ?

1. What is Composition API





1. From Options API to Composition API



Chapter 280 – Replacing “data” with “refs”

1. Ref is the first method will be processed in App.vue. Ref function will create an object and reactive value. Using ref if we want to change using timer, it can be change after several second.

<template>

  <section class="container">

    <h2>{{ userName }}</h2>

  </section>

</template>

<script>

import { ref } from 'vue';

export default {

  // setup will be processed by Vue in the early, before other process

  setup() {

    const uName = ref('Maximilian');

    return { userName: uName };

  }

  // data() {

  //   return {

  //     userName: 'Maximilian',

  //   };

  // },

};

</script>

Chapter 281 – Building “reactive” Objects

1. Return 2 value using ref.

<template>

  <section class="container">

    <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3>

  </section>

</template>

<script>

import { ref } from 'vue';

export default {

  // setup will be processed by Vue in the early, before other process

  setup() {

    const uName = ref('Daivalentineno');

    const uAge = ref(25);

    setTimeout(function() {

      uName.value = 'Daiva';

      uAge.value = 22;

    }, 2000);

    return { userName: uName, age: uAge };

  }

  // data() {

  //   return {

  //     userName: 'Maximilian',

  //   };

  // },

};

</script>

1. Return variable with object.

<template>

  <section class="container">

    <!-- <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3> -->

    <h2>{{ user.name }}</h2>

    <h3>{{ user.age }}</h3>

  </section>

</template>

<script>

import { ref } from 'vue';

export default {

  // setup will be processed by Vue in the early, before other process

  setup() {

    // const uName = ref('Daivalentineno');

    // const uAge = ref(25);

    const user = ref({

      name: 'Daivalentineno',

      age: 25

    });

    setTimeout(function() {

      // uName.value = 'Daiva';

      // uAge.value = 22;

      user.value.name = 'Daiva';

      user.value.age = 22;

    }, 2000);

    return { user: user };

  }

  // data() {

  //   return {

  //     userName: 'Maximilian',

  //   };

  // },

};

</script>

1. We also can use import reactive, it is like ref but it is for object.

<template>

  <section class="container">

    <!-- <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3> -->

    <h2>{{ user.name }}</h2>

    <h3>{{ user.age }}</h3>

  </section>

</template>

<script>

// import { ref } from 'vue';

import { reactive } from 'vue';

export default {

  // setup will be processed by Vue in the early, before other process

  setup() {

    // const uName = ref('Daivalentineno');

    // const uAge = ref(25);

    // const user = ref({

    //   name: 'Daivalentineno',

    //   age: 25

    // });

    const user = reactive({

      name: 'Daivalentineno',

      age: 25

    });

    setTimeout(function() {

      // uName.value = 'Daiva';

      // uAge.value = 22;

      // user.value.name = 'Daiva';

      // user.value.age = 22;

      user.name = 'Daiva';

      user.age = 22;

    }, 2000);

    return { user: user };

  }

  // data() {

  //   return {

  //     userName: 'Maximilian',

  //   };

  // },

};

</script>

Chapter 282 – Reactivity: A Deep Dive

Skip

Chapter 283 – Replacing “methods” with Regular Functions

1. We can use function to run a method in our Vue App.

<template>

  <section class="container">

    <h2>{{ user.name }}</h2>

    <h3>{{ user.age }}</h3>

    <button @click="setAge">Change Age</button>

  </section>

</template>

<script>

// import { ref } from 'vue';

import { reactive } from 'vue';

export default {

  setup() {

    const user = reactive({

      name: 'Daivalentineno',

      age: 25

    });

    function setNewAge() {

      user.age = 22;

    }

    return { user: user, setAge: setNewAge};

  }

};

</script>

Assignment 7: Time to Practice: Data And Functions

Chapter 284 – Replacing “Computed Properties” with the “computed” Function

1. We can use computed to monitor the input value. uName can be filled with the value of first name and last name without using uName.value because it used computed. We can use function to declare a method.

<h2>{{ userName }}</h2>

export default {

  setup() {

    // const uName = ref('Daivalentineno');

    const firstName = ref('');

    const lastName = ref('');

    const uAge = ref(25);

    const uName = computed(function() {

      return firstName.value + ' ' + lastName.value;

    });

    function setNewAge() {

      uAge.value = 22;

    }

    function setFirstName() {

      firstName.value = event.target.value;

    }

    function setLastName() {

      lastName.value = event.target.value;

    }

    return {

      userName: uName,

      age: uAge,

      setAge: setNewAge,

      setFirstName,

      setLastName

    };

  }

};

Chapter 285 – Two Way Binding And The Composition API

1. We can use two way binding when we want to read and listen a variable like below.

<template>

  <section class="container">

    <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3>

    <button @click="setAge">Change Age</button>

    <div>

      <input type="text" placeholder="First Name" v-model="firstName"/>

      <input type="text" placeholder="Last Name" v-model="lastName"/>

    </div>

  </section>

</template>

<script>

import { ref,computed } from 'vue';

export default {

  setup() {

    // const uName = ref('Daivalentineno');

    const firstName = ref('');

    const lastName = ref('');

    const uAge = ref(25);

    const uName = computed(function() {

      return firstName.value + ' ' + lastName.value;

    });

    function setNewAge() {

      uAge.value = 22;

    }

    return {

      userName: uName,

      age: uAge,

      setAge: setNewAge,

      firstName,

      lastName

    };

  }

};

</script>

Chapter 286 – Working with Watchers

1. We can add watch to monitor the changes of variable value like below.

<template>

  <section class="container">

    <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3>

    <button @click="setAge">Change Age</button>

    <div>

      <input type="text" placeholder="First Name" v-model="firstName"/>

      <input type="text" placeholder="Last Name" v-model="lastName"/>

    </div>

  </section>

</template>

<script>

import { ref,computed, watch } from 'vue';

export default {

  setup() {

    // const uName = ref('Daivalentineno');

    const firstName = ref('');

    const lastName = ref('');

    const uAge = ref(25);

    const uName = computed(function() {

      return firstName.value + ' ' + lastName.value;

    });

    // watch(uAge, function(newValue, oldValue) {

    //     console.log('Old age: ' + oldValue);

    //     console.log('New age: ' + newValue);

    // });

    watch([uAge, uName], function(newValues, oldValues) {

        console.log('Old age: ' + oldValues[0]);

        console.log('New age: ' + newValues[0]);

        console.log('Old name: ' + oldValues[1]);

        console.log('New name: ' + newValues[1]);

    });

    function setNewAge() {

      uAge.value = 22;

    }

    return {

      userName: uName,

      age: uAge,

      setAge: setNewAge,

      firstName,

      lastName

    };

  }

};

</script>

Assignment 8: Time to Practice: Composition API Core Building Blocks

Chapter 287 – A First Summary

1. We still use interpolation, v-bind, listen to events and same with before also use v-model.
2. Component and props will not change.

Chapter 288 – How To Use Template Refs

1. If we want to use ref=”” in our HTML template, we must write like below to get the data. Using this way, we can not watch the changes of lastNameInput.

<input type="text" placeholder="Last Name" ref="lastNameInput"/>

function setLastName() {

      lastName.value = lastNameInput.value.value;

}

Chapter 289 – Components, Props, And The Composition API

1. We can separate our file into some components and using props as passing parameter like below.

<template>

  <section class="container">

    <user-data :first-name="firstName" :last-name="lastName" :age="age"></user-data>

    <button @click="setAge">Change Age</button>

    <div>

      <input type="text" placeholder="First Name" v-model="firstName"/>

      <input type="text" placeholder="Last Name" ref="lastNameInput"/>

      <button @click="setLastName">Set Last Name</button>

      <!-- v-model="lastName" -->

    </div>

  </section>

</template>

<template>

    <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3>

</template>

<script>

import { computed } from 'vue';

export default {

   props: ['firstName', 'lastName', 'age'],

   setup(props) {

        const uName = computed(function() {

           return props.firstName + ' ' + props.lastName;

        });

        return {

           userName: uName

        }

   }

}

</script>

Chapter 290 – Emitting Custom Events

1. We can use context to emit some method in Composition API

<template>

    <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3>

</template>

<script>

import { computed } from 'vue';

export default {

   props: ['firstName', 'lastName', 'age'],

   setup(props, context) {

        const uName = computed(function() {

           return props.firstName + ' ' + props.lastName;

        });

        console.log(context);

        // this.$emit('save-data', 1);

        context.emit('save-data', 1);

        return {

           userName: uName

        }

   }

}

</script>

Chapter 291 – Working With Provide / Inject

1. We want to send a dynamic variable which can be changed after we do some function like clicking a button. The code will be like below.

<template>

  <section class="container">

    <user-data :first-name="firstName" :last-name="lastName"></user-data>

    <button @click="setAge">Change Age</button>

    <div>

      <input type="text" placeholder="First Name" v-model="firstName"/>

      <input type="text" placeholder="Last Name" ref="lastNameInput"/>

      <button @click="setLastName">Set Last Name</button>

    </div>

  </section>

</template>

<script>

import { ref, provide } from 'vue';

import UserData from './components/UserData.vue'

export default {

  components: {

    UserData

  },

  setup() {

    // const uName = ref('Daivalentineno');

    const firstName = ref('');

    const lastName = ref('');

    const lastNameInput = ref(null);

    const uAge = ref(25);

    provide('userAge', uAge);

    function setNewAge() {

      uAge.value = 22;

    }

    function setLastName() {

      lastName.value = lastNameInput.value.value;

}

    return {

      setAge: setNewAge,

      firstName,

      lastName,

      lastNameInput,

      setLastName

    };

  }

};

</script>

<template>

    <h2>{{ userName }}</h2>

    <h3>{{ age }}</h3>

</template>

<script>

import { computed, inject } from 'vue';

export default {

   props: ['firstName', 'lastName'],

   setup(props, context) {

        const uName = computed(function() {

           return props.firstName + ' ' + props.lastName;

        });

        const age = inject('userAge');

        console.log(context);

        // this.$emit('save-data', 1);

        context.emit('save-data', 1);

        return {

           userName: uName,

           age

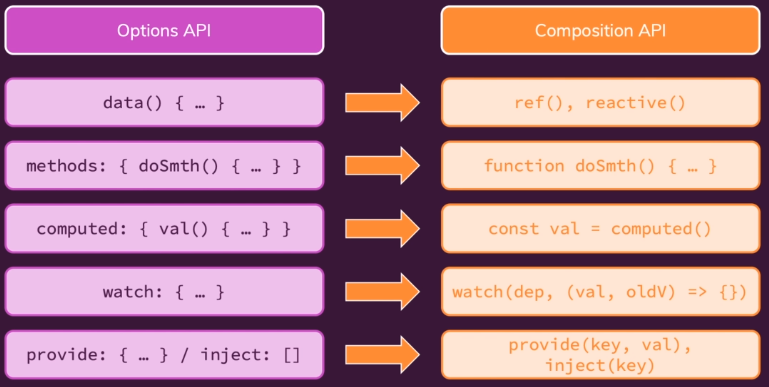
        }

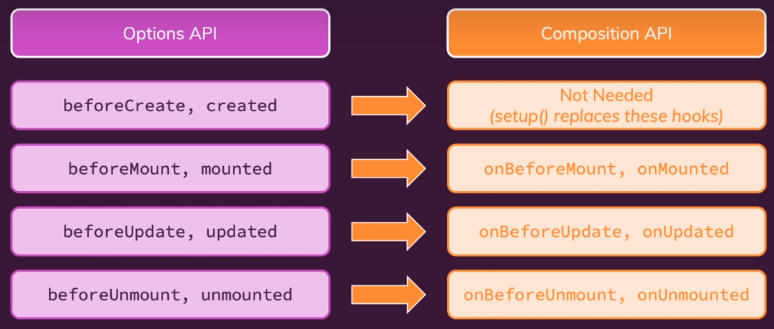
   }

}

</script>

Chapter 292 – Lifecycle Hooks in the Composition API





1. The code in UserData.vue will be like below.

<script>

import {

    computed,

    inject,

    onBeforeMount,

    onMounted,

    onBeforeUpdate,

    onUpdated,

    onBeforeUnmount,

    onUnmounted

} from 'vue';

export default {

   props: ['firstName', 'lastName'],

   setup(props, context) {

        const uName = computed(function() {

           return props.firstName + ' ' + props.lastName;

        });

        const age = inject('userAge');

        console.log(context);

        onBeforeMount(function() {

            console.log('onBeforeMount');

        });

        onMounted(function() {

            console.log('onMounted');

        });

        onBeforeUpdate(function() {

            console.log('onBeforeUpdate');

        });

        onUpdated(function() {

            console.log('onUpdated');

        });

        onBeforeUnmount(function() {

            console.log('onBeforeUnmount');

        });

        onUnmounted(function() {

            console.log('onUnmounted');

        });

        // this.$emit('save-data', 1);

        context.emit('save-data', 1);

        return {

           userName: uName,

           age

        }

   }

}

</script>

Chapter 293 – Migrating from Options API to Composition API – An Example Project

1. Try to change from options API to composition API.

Chapter 294 – Migrating A First Component

<template>

  <main>

    <user-list :users="activeUsers" @list-projects="selectUser"></user-list>

    <projects-list :user="selectedUser"></projects-list>

  </main>

</template>

<script>

import { ref } from 'vue';

import USER\_DATA from './dummy-data.js';

import UserList from './components/users/UserList.vue';

import ProjectsList from './components/projects/ProjectsList.vue';

export default {

  components: {

    UserList,

    ProjectsList,

  },

  setup () {

    const selectedUser = ref(null);

    // no need ref() because the data will not be changed

    const activeUsers = USER\_DATA;

    function selectUser(uid) {

      selectedUser.value = activeUsers.find((usr) => usr.id === uid);

    }

    return {

      selectedUser,

      activeUsers,

      selectUser

    }

  },

  // data() {

  //   return {

  //     selectedUser: null,

  //     activeUsers: USER\_DATA,

  //   };

  // },

  // methods: {

  //   selectUser(uid) {

  //     this.selectedUser = this.activeUsers.find((usr) => usr.id === uid);

  //   },

  // },

};

</script>

Chapter 295 – Migrating A Big Component

<template>

  <base-container>

    <h2>Active Users</h2>

    <base-search @search="updateSearch" :search-term="enteredSearchTerm"></base-search>

    <div>

      <button @click="sort('asc')" :class="{selected: sorting === 'asc'}">Sort Ascending</button>

      <button @click="sort('desc')" :class="{selected: sorting === 'desc'}">Sort Descending</button>

    </div>

    <ul>

      <user-item

        v-for="user in displayedUsers"

        :key="user.id"

        :user-name="user.fullName"

        :id="user.id"

        @list-projects="$emit('list-projects', $event)"

      ></user-item>

    </ul>

  </base-container>

</template>

<script>

import { ref, computed, watch, toRefs } from 'vue';

import ProjectItem from './ProjectItem.vue';

export default {

  components: {

    ProjectItem,

  },

  props: ['user'],

  setup(props) {

    const enteredSearchTerm = ref('');

    const activeSearchTerm = ref('');

    function updateSearch(val) {

      enteredSearchTerm.value = val;

    }

    const hasProjects = computed(function() {

        return props.user.projects && availableProjects.value.length > 0;

    });

    const availableProjects = computed(function() {

        if (activeSearchTerm.value) {

        return props.user.projects.filter((prj) =>

            prj.title.includes(activeSearchTerm.value)

          );

        }

        return props.user.projects;

    });

    watch(enteredSearchTerm, function(newValue) {

      setTimeout(() => {

        if (newValue === enteredSearchTerm.value) {

          activeSearchTerm.value = newValue;

        }

      }, 300);

    });

    // We can declare props like below

    // Method 1

    // const propsWithRefs = toRefs(props);

    // const user = propsWithRefs.user;

    // Method 2

    const { user } = toRefs(props);

    // Just use props because the props value only 1

    // watch(props.user, function(newValue) {

    // You can use below if you didnt define the user using toRefs

    // watch(props, function() {

    // You can use below code (commented) if you use toRefs

    watch(user, function() {

      enteredSearchTerm.value = '';

    });

    return {

      enteredSearchTerm,

      availableProjects,

      hasProjects,

      updateSearch

    }

  },

  // data() {

  //   return {

  //     enteredSearchTerm: '',

  //     activeSearchTerm: '',

  //   };

  // },

  // computed: {

  //   hasProjects() {

  //     return this.user.projects && this.availableProjects.length > 0;

  //   },

  //   availableProjects() {

  //     if (this.activeSearchTerm) {

  //       return this.user.projects.filter((prj) =>

  //         prj.title.includes(this.activeSearchTerm)

  //       );

  //     }

  //     return this.user.projects;

  //   },

  // },

  // methods: {

  //   updateSearch(val) {

  //     this.enteredSearchTerm = val;

  //   },

  // },

  // watch: {

  //   enteredSearchTerm(val) {

  //     setTimeout(() => {

  //       if (val === this.enteredSearchTerm) {

  //         this.activeSearchTerm = val;

  //       }

  //     }, 300);

  //   },

  //   user() {

  //     this.enteredSearchTerm = '';

  //   },

  // },

};

</script>

Chapter 296 – Migrating the Remaining Components

1. We can use toRefs to define the props if the props value only one.

<template>

  <base-container v-if="user">

    <h2>{{ user.fullName }}: Projects</h2>

    <base-search v-if="hasProjects" @search="updateSearch" :search-term="enteredSearchTerm"></base-search>

    <ul v-if="hasProjects">

      <project-item v-for="prj in availableProjects" :key="prj.id" :title="prj.title"></project-item>

    </ul>

    <h3 v-else>No projects found.</h3>

  </base-container>

  <base-container v-else>

    <h3>No user selected.</h3>

  </base-container>

</template>

<script>

import { ref, computed, watch, toRefs } from 'vue';

import ProjectItem from './ProjectItem.vue';

export default {

  components: {

    ProjectItem,

  },

  props: ['user'],

  setup(props) {

    const enteredSearchTerm = ref('');

    const activeSearchTerm = ref('');

    function updateSearch(val) {

      enteredSearchTerm.value = val;

    }

    const hasProjects = computed(function() {

        return props.user.projects && availableProjects.value.length > 0;

    });

    const availableProjects = computed(function() {

        if (activeSearchTerm.value) {

        return props.user.projects.filter((prj) =>

            prj.title.includes(activeSearchTerm.value)

          );

        }

        return props.user.projects;

    });

    watch(enteredSearchTerm, function(newValue) {

      setTimeout(() => {

        if (newValue === enteredSearchTerm.value) {

          activeSearchTerm.value = newValue;

        }

      }, 300);

    });

    const propsWithRefs = toRefs(props);

    const user = propsWithRefs.user;

    const { user } = toRefs(props);

    // Just use props because the props value only 1

    // watch(props.user, function(newValue) {

    // You can use below if you didnt define the user using toRefs

    // watch(props, function() {

    // You can use below code (commented) if you use toRefs

    watch(user, function() {

      enteredSearchTerm.value = '';

    });

    return {

      enteredSearchTerm,

      availableProjects,

      hasProjects,

      updateSearch

    }

  },

  // data() {

  //   return {

  //     enteredSearchTerm: '',

  //     activeSearchTerm: '',

  //   };

  // },

  // computed: {

  //   hasProjects() {

  //     return this.user.projects && this.availableProjects.length > 0;

  //   },

  //   availableProjects() {

  //     if (this.activeSearchTerm) {

  //       return this.user.projects.filter((prj) =>

  //         prj.title.includes(this.activeSearchTerm)

  //       );

  //     }

  //     return this.user.projects;

  //   },

  // },

  // methods: {

  //   updateSearch(val) {

  //     this.enteredSearchTerm = val;

  //   },

  // },

  // watch: {

  //   enteredSearchTerm(val) {

  //     setTimeout(() => {

  //       if (val === this.enteredSearchTerm) {

  //         this.activeSearchTerm = val;

  //       }

  //     }, 300);

  //   },

  //   user() {

  //     this.enteredSearchTerm = '';

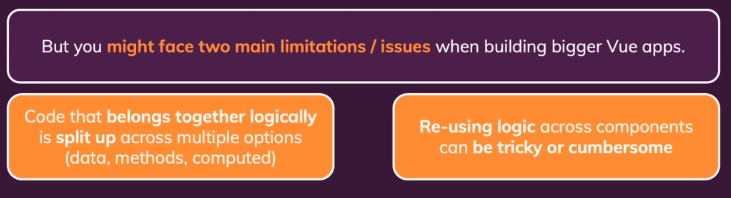
  //   },

  // },

};

</script>

1. Summary



Chapter 297 – Routing, Params, and The Compositions AP

1. If we have a parameter we need to send to another link, we need to add props in our router.

Main.js

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/', redirect: '/products' },

    { path: '/products', component: AllProducts },

    { path: '/products/:pid', component: ProductDetails, props: true },

    { path: '/products/add', component: AddProduct }

  ]

});

We have provided products in App.vue

App.vue

<script>

import { ref, provide } from 'vue';

import TheHeader from './components/TheHeader.vue';

export default {

  components: {

    TheHeader,

  },

  setup() {

    const products = ref([

      {

        id: 'p1',

        title: 'A Carpet',

        description: 'A nice looking, maybe a little bit used carpet.',

        price: 15.99,

      },

      {

        id: 'p2',

        title: 'A Book',

        description: 'You can read it. Maybe you should read it.',

        price: 12.99,

      },

    ]);

    function addProduct(productData) {

      const newProduct = {

        id: new Date().toISOString(),

        title: productData.title,

        description: productData.description,

        price: productData.price,

      };

      products.value.push(newProduct);

    }

    provide('products', products);

    provide('addProduct', addProduct);

  },

};

</script>

Therefore, in AllProducts.vue, we need to inject the products.

<template>

  <section>

    <h2>All products</h2>

    <ul>

      <li v-for="product in products" :key="product.id">

        <h3>{{ product.title }}</h3>

        <h4>${{ product.price }}</h4>

        <p>{{ product.description }}</p>

        <router-link :to="'/products/' + product.id">View Details</router-link>

      </li>

    </ul>

  </section>

</template>

<script>

import { inject } from 'vue';

export default {

  setup() {

    const loadedProducts = inject('products');

    return { products: loadedProducts };

  },

};

</script>

In ProductDetails.vue, we also inject products

<template>

  <section>

    <h2>{{ title }}</h2>

    <h3>${{ price }}</h3>

    <p>{{ description }}</p>

    <router-link to="/products/p2">Product 2</router-link>

  </section>

</template>

<script>

import { inject, computed } from 'vue';

export default {

  props: ['pid'],

  setup(props) {

    const products = inject('products');

    const selectedProduct = computed (() =>

      products.value.find((product) => product.id === props.pid)

    );

    const title = computed(() => selectedProduct.value.title);

    const price = computed(() => selectedProduct.value.price);

    const description = computed(() => selectedProduct.value.description);

    return { title, price, description };

  },

};

</script>

1. We need to change the id in AllProducts.vue

<template>

  <section>

    <h2>All products</h2>

    <ul>

      <li v-for="product in products" :key="product.id">

        <h3>{{ product.title }}</h3>

        <h4>${{ product.price }}</h4>

        <p>{{ product.description }}</p>

        <router-link :to="'/products/' + product.id">View Details</router-link>

      </li>

    </ul>

  </section>

</template>

Chapter 298 – The Route and Router Objects and the Composition API

1. useRoute allows us to get access to this route object. It will print a proxy with all that route data in there. We can use in computer props to update any data that depends on it, whenever route changes. The result will be similar to props.id

ProductDetails.vue

import { use } from 'vue-router';

export default {

  props: ['pid'],

  setup(props) {

    const products = inject('products');

    const route = useRoute();

    console.log(route);

    // const selectedProduct = computed (() =>

    //   products.value.find((product) => product.id === props.pid)

    // );

    const selectedProduct = computed (() =>

      products.value.find((product) => product.id === route.params.pid)

    );

    const title = computed(() => selectedProduct.value.title);

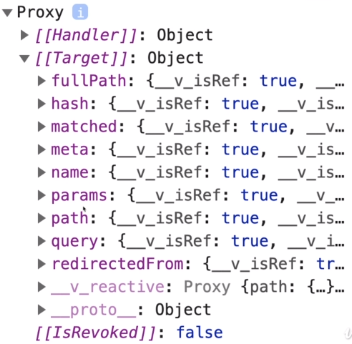
    const price = computed(() => selectedProduct.value.price);

    const description = computed(() => selectedProduct.value.description);

    return { title, price, description };

  },

};



UseRouter to access another link.

import { ref, inject } from 'vue';

import { useRouter } from 'vue-router';

export default {

  setup() {

    const addProduct = inject('addProduct');

    const router = useRouter();

    const enteredTitle = ref('');

    const enteredPrice = ref(null);

    const enteredDescription = ref('');

    function submitForm() {

      addProduct({

        title: enteredTitle,

        description: enteredDescription,

        price: enteredPrice,

      });

      router.push('/products');

    }

    return {

      enteredTitle,

      enteredPrice,

      enteredDescription,

      submitForm,

    };

  },

};

Chapter 299 – Using Vuex with the Composition API

1. We can use useStore to get and set the data.

App.vue

<template>

  <the-counter></the-counter>

  <control-center></control-center>

</template>

<script>

import ControlCenter from './components/ControlCenter.vue';

import TheCounter from './components/TheCounter.vue';

export default {

  components: {

    ControlCenter,

    TheCounter,

  },

};

</script>

TheCounter.vue

<template>

  <h2>{{ counter }}</h2>

</template>

<script>

import { computed } from 'vue';

import { useStore } from 'vuex';

export default {

  setup() {

    const store = useStore();

    // const counter = 0;

    const counter = computed(function() {

      return store.getters.counter;

    });

    return { counter };

  },

};

</script>

ControlCenter.vue

<template>

  <button @click="inc">Increment</button>

</template>

<script>

import { useStore } from 'vuex';

export default {

  setup() {

    const store = useStore();

    function inc() {

      store.dispatch('increment');

    }

    return { inc };

  },

};

</script>

Chapter 300 – Summary

