Vue.js Стартовий

Основи роботи з компонентами: watch, props, emit

Vue.js Стартовий

Introduction

Кінаш Станіслав

Front-end dev

stanislav.kinash

stanislav.kinash

MCID: 9210561

Тема уроку

Основи роботи з компонентами: watch, props, emit

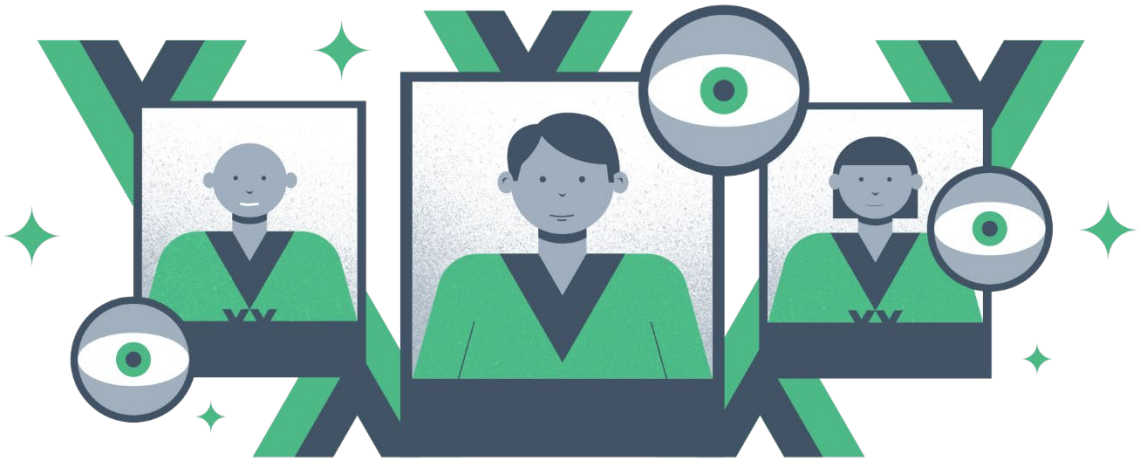
План уроку

1. Що таке watch?
2. Принцип роботи watch.
3. Що таке watchEffect?
4. Принцип роботи watchEffect.
5. Що таке props?
6. Що таке emit?
7. Приклад роботи з emit та props.

Що таке watch?

Watch — це функція, яка дозволяє спостерігати за змінами властивості даних і реагувати на ці зміни. Коли властивість даних, за якою спостерігають, змінюється, виконується функція зворотного виклику (callback function).

Спостерігач watch корисний для обробки побічних ефектів, виконання перевірки даних або обчислень, а також ініціювання дій на основі змін у даних.



Принцип роботи watch?

У цьому прикладі ми маємо простий компонент Vue.js із властивістю message та вхідним елементом, який прив'язаний до властивості newMessage за допомогою v-model.

Коли значення newMessage змінюється, функція зворотного виклику спостерігача буде виконана, і властивість повідомлення буде оновлено, щоб включити нове значення newMessage.

```
<template>
  <div>
    <p>{{ message }}</p>
    <input v-model="newMessage">
  </div>
</template>
```

```
<script>
export default {
  data() {
    return {
      message: "Hello, World!",
      newMessage: "",
    };
  },
  watch: {
    newMessage(newValue) {
      this.message = `You typed: ${newValue}`;
    },
  },
};
</script>
```

Що таке watchEffect?

WatchEffect — це новий API, який дозволяє створювати реактивний ефект, який автоматично відстежує залежності коду всередині наданої вами функції. Кожного разу, коли будь-яка реактивна залежність функції змінюється, ефект автоматично запускається повторно.

На відміну від watch, який вимагає від вас вказати певну властивість для перегляду, watchEffect динамічно відстежує всі реактивні залежності, що використовуються у функції зворотного виклику, що може спростити ваш код.

Принцип роботи watchEffect?

Приклад, який демонструє, як використовувати **watchEffect** для відстеження реактивної залежності та відповідного оновлення інтерфейсу користувача:

```
<template>
    <div>
      <p>Counter: {{ counter }}</p>
      <button @click="incrementCounter">Increment</button>
    </div>
</template>
```

```
<script>
import { ref, watchEffect } from 'vue';

export default {
  setup() {
    const counter = ref(0);

    watchEffect(() => {
      console.log('Counter changed:', counter.value);
    });

    function incrementCounter() {
      counter.value++;
    }

    return { counter, incrementCounter };
  },
};
</script>
```

Принцип роботи watchEffect?

У цьому прикладі ми створюємо реактивну змінну лічильника за допомогою функції ref.
Ми використовуємо watchEffect для відстеження змін у змінній лічильника.

Кожного разу, коли змінна лічильника змінюється, функція зворотного виклику всередині watchEffect буде повторно виконана, і консоль буде реєструвати повідомлення про те, що лічильник змінився.

Коли ви натискаєте кнопку «Збільшити» в шаблоні, змінна лічильника буде збільшена, а консоль запише повідомлення про те, що лічильник змінився. Шаблон також буде повторно відтворено, щоб показати оновлене значення лічильника.

```
<script>
import { ref, watchEffect } from 'vue';

export default {
  setup() {
    const counter = ref(0);

    watchEffect(() => {
      console.log('Counter changed:', counter.value);
    });

    function incrementCounter() {
      counter.value++;
    }

    return { counter, incrementCounter };
  },
};
</script>
```

Що таке props?

Props — це механізм для передачі даних від батьківського компонента до дочірнього компонента. Коли ви визначаєте властивість у дочірньому компоненті, ви можете використовувати її так само, як і будь-яку іншу властивість даних у компоненті.

Що таке props?

```vue
<!-- ParentComponent.vue -->
<template>
  <div>
    <child-component :message="parentMessage"></child-component>
  </div>
</template>

<script>
import ChildComponent from './ChildComponent.vue';

5+ usages  neprostostas +1 *
export default {
  data() {
    return {
      parentMessage: 'Hello from the parent component!',
    };
  },
  components: {
    ChildComponent,
  },
};
</script>
```

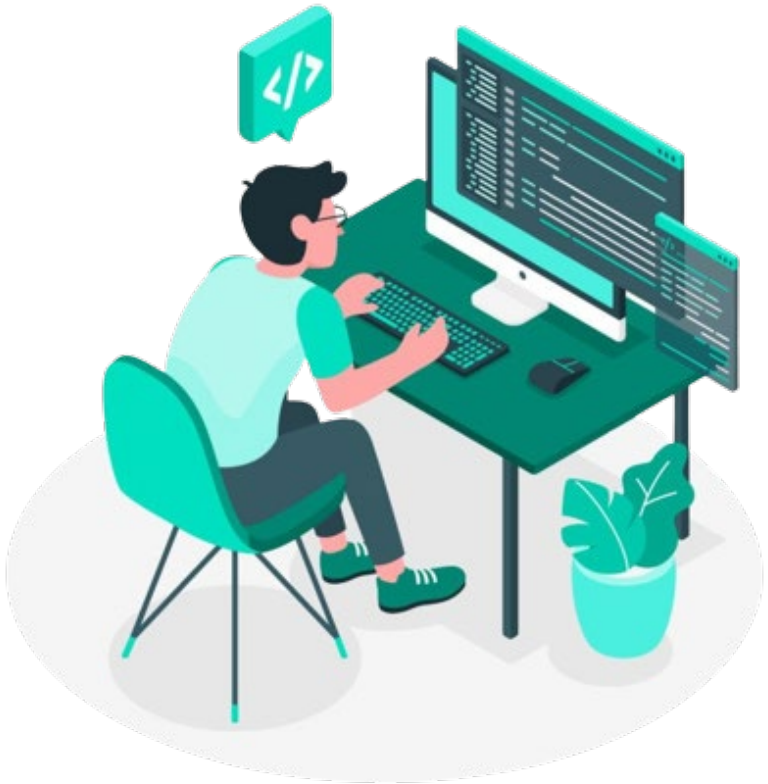
```vue
<!-- ChildComponent.vue -->
<template>
  <div>
    <p>{{ message }}</p>
  </div>
</template>

<script>
no usages  new *
export default {
  props: ['message'],
};
</script>
```

Що таке props?

Щоб передати дані parentMessage у дочірній компонент, ми використовуємо властивість message в дочірньому компоненті.
Ми передаємо значення parentMessage пропу повідомлення, використовуючи : скорочення для прив'язаних пропсів.

У дочірньому компоненті ми визначаємо атрибут повідомлення за допомогою параметра props. Ми використовуємо синтаксис масиву, щоб визначити один проп під назвою message.

Після визначення атрибута повідомлення в дочірньому компоненті ми можемо використовувати його, як і будь-яку іншу властивість даних у компоненті.



Що таке emit?

Emit — метод, який дозволяє дочірнім компонентам спілкуватися з батьківськими компонентами, випромінюючи події. Коли подія надходить від дочірнього компонента, вона може бути зафіксована батьківським компонентом за допомогою директиви v-on.

Що таке emit?

У цьому прикладі ми маємо батьківський і дочірній компоненти.

Дочірній компонент має кнопку, яка при натисканні збільшує властивість даних лічильника.

Коли лічильник збільшується, дочірній компонент видає подію під назвою counter-incremented за допомогою методу $emit.

```vue
<!-- ChildComponent.vue -->
<template>
  <div>
    <button @click="incrementCounter">Increment</button>
  </div>
</template>

<script>
5+ usages  new *
export default {
  data() {
    return {
      counter: 0,
    };
  },
  methods: {
    incrementCounter() {
      this.counter++;
      this.$emit('counter-incremented', this.counter);
    },
  },
};
</script>
```

Що таке emit?

Батьківський компонент прослуховує подію counter-incremented за допомогою директиви v-on і викликає метод handleCounterIncrement із новим значенням властивості counter.

У методі handleCounterIncrement ми оновлюємо значення властивості лічильника в батьківському компоненті новим значенням, переданим із дочірнього компонента.

```vue
<!-- ParentComponent.vue -->
<template>
  <div>
    <p>Counter: {{ counter }}</p>
    <child-component @counter-incremented="handleCounterIncrement">
    </child-component>
  </div>
</template>

<script>
import ChildComponent from './ChildComponent.vue';

5+ usages  new *
export default {
  data() {
    return {
      counter: 0,
    };
  },
  components: {
    ChildComponent,
  },
  methods: {
    handleCounterIncrement(counter) {
      this.counter = counter;
    },
  },
};
</script>
```

Що таке emit?

Підводячи підсумок, emit у Vue.js — це спосіб для дочірніх компонентів спілкуватися з батьківськими компонентами шляхом випромінювання подій.

Це забезпечує кращий склад компонентів і полегшує створення багаторазових компонентів, які можна використовувати в різних контекстах.

Приклад роботи з emit та props.

```
<!-- ChildComponent.vue -->
<template>
  <div>
    <input type="text" v-model="newItemText">
    <button @click="addItem">Add Item</button>
  </div>
</template>

<script>
no usages  new *
export default {
  data() {
    return {
      newItemText: '',
    };
  },
  methods: {
    addItem() {
      this.$emit('item-added', this.newItemText);
      this.newItemText = '';
    },
  },
};
</script>
```

У цьому прикладі ми маємо батьківський і дочірній компоненти. Дочірній компонент має поле введення та кнопку, яка дозволяє користувачеві додати новий елемент до списку. Коли користувач натискає кнопку «Додати елемент», дочірній компонент видає подію під назвою item-added із новим текстом елемента як параметром.

Приклад роботи з emit та props.

```vue
<!-- ParentComponent.vue -->
<template>
  <div>
    <ul>
      <li v-for="item in items" :key="item">{{ item }}</li>
    </ul>
    <child-component @item-added="addItem"></child-component>
  </div>
</template>
```

Батьківський компонент прослуховує подію item-added за допомогою директиви v-on і викликає метод addItem із новим текстом елемента. У методі addItem ми оновлюємо значення масиву items у батьківському компоненті новим елементом.

```vue
<script>
import ChildComponent from './ChildComponent.vue';

no usages  new *
export default {
  data() {
    return {
      items: [],
    };
  },
  components: {
    ChildComponent,
  },
  methods: {
    addItem(newItemText) {
      this.items.push(newItemText);
    },
  },
};
</script>
```

Приклад роботи з emit та props.

У підсумку цей приклад показує, як використовувати **props** для передачі даних від батьківського компонента до дочірнього компонента та **emit** для передачі даних від дочірнього компонента до батьківського компонента.

Він демонструє, як ці механізми можна використовувати разом для створення динамічного інтерактивного інтерфейсу користувача.


ITVDN
IT VIDEO DEVELOPERS NETWORK


TestProvider
Pearson
CyberBionic
systematics
Microsoft Partner
Windows Azure
EBA

Перевірка знань

TestProvider.com

TestProvider – це online-сервіс перевірки знань з інформаційних технологій. За його допомогою ви можете оцінити свій рівень та виявити слабкі місця. Він буде корисним як у процесі вивчення технології, так і для загальної оцінки знань IT-спеціаліста.

Перевірте, як ви засвоїли даний матеріал на TestProvider.com

Успішне проходження фінального тестування дозволить отримати відповідний Сертифікат.

Vue.js Стартовий

Дякую за увагу! До нових зустрічей!

Кінаш Станіслав
Front-end dev

MCID: 9210561