Vue.js Стартовий

Option/Composition API. Slots.

Introduction

Кінаш Станіслав
Front-end dev

stanislav.kinash

stanislav.kinash

MCID: 9210561

Тема уроку

Option/Composition API. Slots.

План уроку

1. Як працює реактивність у Vue?
2. Різниця між Option та Composition API
3. Що таке слоти?
4. Приклад застосування слотів в компонентах.

Як працює реактивність у Vue?

У Vue 3 реактивність обробляється новим **Composition API**, який дозволяє розробникам створювати реактивні дані та функції за допомогою функцій **ref** і **reactive**.

Як працює реактивність у Vue?

Функція ref

Створює реактивну змінну, яку можна використовувати в шаблонах і коді JavaScript. Коли значення змінної ref змінюється, будь-які компоненти, які використовують цю змінну, будуть повторно відображені, щоб відобразити нове значення.

Ось приклад створення змінної ref і використання її в компоненті:

```
import { ref } from 'vue';

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

    function increment() {
      count.value++;
    }

    return { count, increment };
  }
}
```

Як працює реактивність у Vue?

У наведеному вище прикладі ми створюємо змінну підрахунку за допомогою функції ref із початковим значенням 0. Ми також визначаємо функцію збільшення, яка збільшує значення підрахунку на 1. Нарешті, ми повертаємо і підрахунок, і приріст із функції налаштування, так що їх можна використовувати в шаблоні компонента.

```
import { ref } from 'vue';

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

    function increment() {
      count.value++;
    }

    return { count, increment };
  }
}
```

Як працює реактивність у Vue?

У шаблоні компонента ми можемо використовувати змінну count так:

```
<template>
    <div>
    <p>Count: {{ count }}</p>
    <button @click="increment">Increment</button>
    </div>
</template>
```

Щоразу, коли викликається функція збільшення, значення count буде оновлено, а компонент буде повторно відображено, щоб відобразити нове значення.

Як працює реактивність у Vue?

Функція reactive

Створює реактивний об’єкт, який можна використовувати подібно до ref. Коли властивість реактивного об’єкта змінюється, будь-які компоненти, які використовують цю властивість, будуть повторно відтворені, щоб відобразити нове значення.

Ось приклад створення реактивного об’єкта та використання його в компоненті:

```
import { reactive } from 'vue';

export default {
  setup() {
    const user = reactive({
      name: 'John',
      age: 30,
      address: {
        street: '123 Main St',
        city: 'Anytown',
        state: 'CA'
      }
    });

    function updateAddress() {
      user.address.street = '456 Maple Ave';
      user.address.city = 'Sometown';
      user.address.state = 'NY';
    }

    return { user, updateAddress };
  }
}
```

Як працює реактивність у Vue?

У наведеному вище прикладі ми створюємо об’єкт користувача за допомогою реактивної функції з трьома властивостями: ім’я, вік та адреса. Властивість адреси сама по собі є об’єктом із трьома властивостями: вулиця, місто та штат.

Ми також визначаємо функцію updateAddress, яка оновлює властивості вулиці, міста та штату об’єкта адреси. Нарешті, ми повертаємо користувача та updateAddress із функції налаштування, щоб їх можна було використовувати в шаблоні компонента.

```
import { reactive } from 'vue';

export default {
  setup() {
    const user = reactive({
      name: 'John',
      age: 30,
      address: {
        street: '123 Main St',
        city: 'Anytown',
        state: 'CA'
      }
    });

    function updateAddress() {
      user.address.street = '456 Maple Ave';
      user.address.city = 'Sometown';
      user.address.state = 'NY';
    }

    return { user, updateAddress };
  }
}
```

Як працює реактивність у Vue?

У шаблоні компонента ми можемо використовувати об’єкт користувача так:

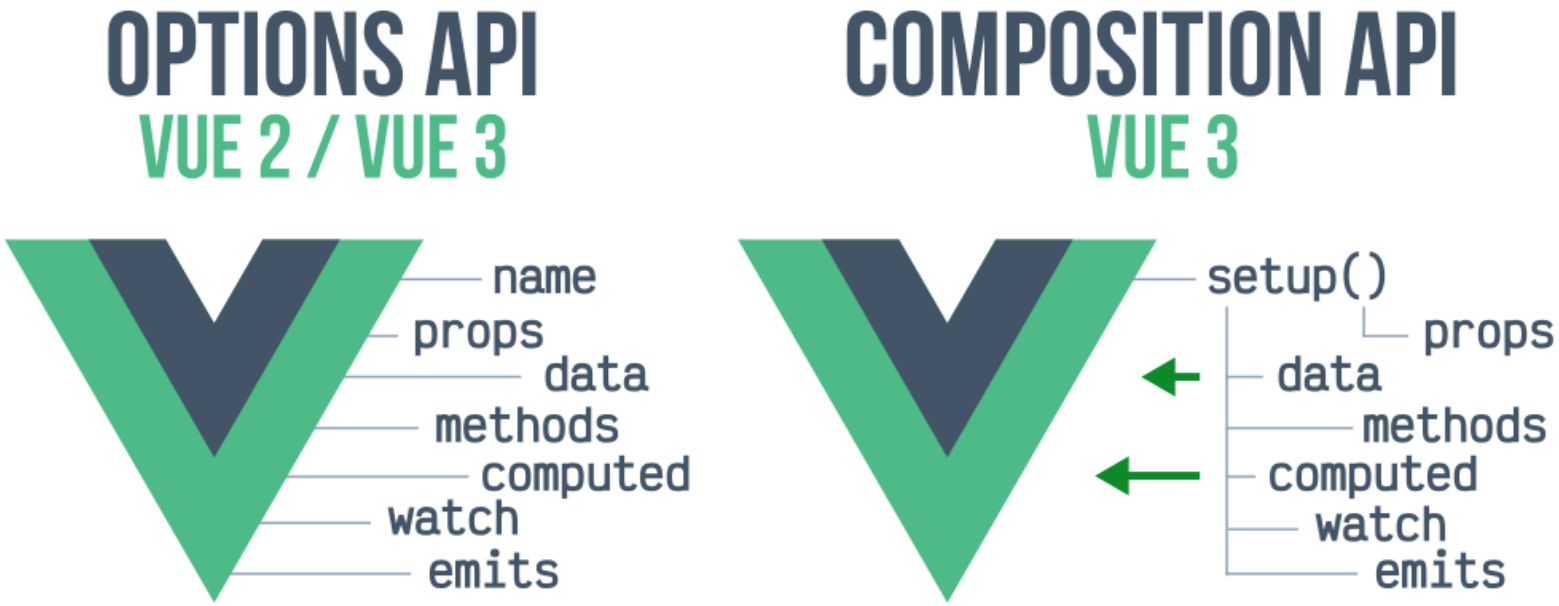
Щоразу, коли викликається функція updateAddress, значення властивостей вулиці, міста та штату об’єкта user.address буде оновлено, а компонент буде повторно відображено, щоб відобразити нові значення.

```
<template>
    <div>
        <p>Name: {{ user.name }}</p>
        <p>Age: {{ user.age }}</p>
        <p>Address: {{ user.address.street }},
            {{ user.address.city }},
            {{ user.address.state }}</p>
        <button @click="updateAddress">
            Update Address
        </button>
    </div>
</template>
```

Різниця між Option та Composition API

Vue 3 надає два способи визначення компонентів: **Option API** та **Composition API.**

Хоча обидва підходи можна використовувати для створення реактивних компонентів, вони мають деякі фундаментальні відмінності в тому, як вони працюють.



Різниця між Option та Composition API

1. Option API визначає параметри як властивості об’єкта, тоді як Composition API визначає реактивні дані та функції за допомогою функцій.

1. Option API може призвести до створення тісно пов’язаного коду, тоді як Composition API дозволяє створити більш модульний код.

1. Option API вимагає використання `this` для доступу до властивостей і методів компонентів, тоді як Composition API використовує деструктуровані об’єкти.

Загалом Composition API є більш гнучким і потужним способом створення реактивних компонентів порівняно з Options API. Це дозволяє використовувати більш модульний код, полегшує керування поведінкою складних компонентів і покращує підтримку TypeScript.

Різниця між Option та Composition API

Option API

Це традиційний спосіб визначення компонентів Vue, який передбачає створення об’єкта з різними параметрами, які визначають поведінку компонента.

Різниця між Option та Composition API

```
<template>
    <div>
        <p>Count: {{ count }}</p>
        <button @click="increment">
            Increment
        </button>
    </div>
</template>

<script>
export default {
    data() {
        return {
            count: 0
        };
    },
    methods: {
        increment() {
            this.count++;
        }
    }
};
</script>
```

Ось приклад використання **Options API** для створення простого компонента:

У цьому прикладі параметр `data` використовується для визначення початкового стану компонента. Параметр `methods` використовується для визначення методів, які можуть змінювати стан компонента. У шаблоні компонента ми використовуємо властивість даних count та метод increment для відображення та оновлення стану компонента.

Різниця між Option та Composition API

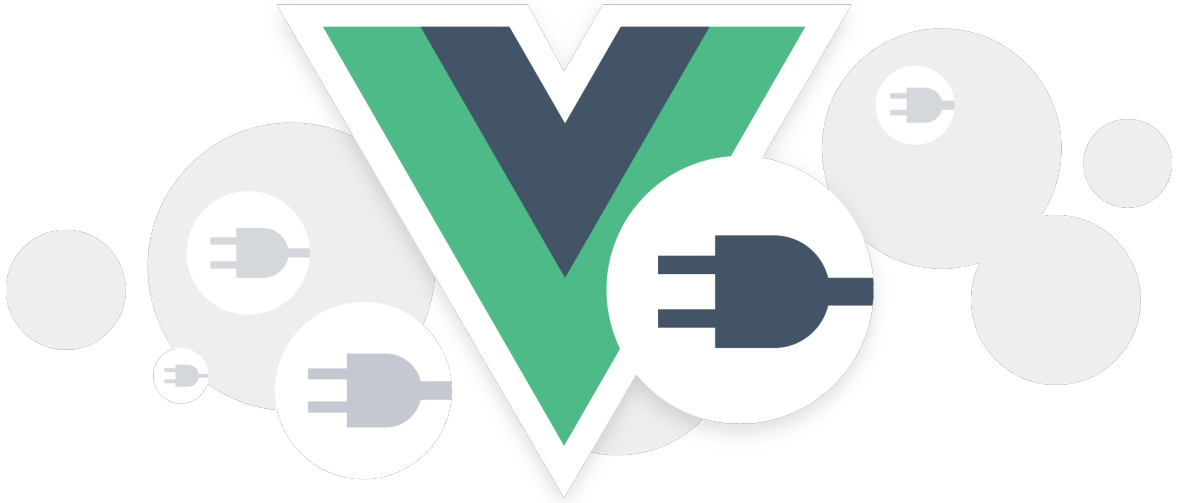
Хоча Options API можна використовувати для створення реактивних компонентів, ним може стати важко керувати, оскільки розмір компонента зростає.

Наприклад, коли до параметра «Методи» додається більше методів, стає важко відстежувати, які методи змінюють властивості даних.

Різниця між Option та Composition API

Composition API

Це новий спосіб визначення компонентів Vue, який передбачає створення реактивних даних і функцій за допомогою функцій **ref** і **reactive**.



Різниця між Option та Composition API

```
<template>
  <div>
    <p>Count: {{ count }}</p>
    <button @click="increment">
      Increment
    </button>
  </div>
</template>

<script>
import { ref } from 'vue';

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

    function increment() {
      count.value++;
    }

    return { count, increment };
  }
};
</script>
```

Ось приклад використання **Composition API** для створення простого компонента:

У цьому прикладі ми використовуємо функцію **setup()**, щоб створити змінну реактивного підрахунку count та функцію збільшення increment за допомогою функції **ref**.

Щоразу, коли значення змінної **ref** змінюється, будь-які компоненти, які використовують цю змінну, будуть повторно відображені, щоб відобразити нове значення.

Функція **setup()** повертає об’єкт, що містить змінну підрахунку count та функцію збільшення increment, яку можна використовувати в шаблоні компонента.

Різниця між Option та Composition API

Метод **setup()** використовується для визначення реактивних даних і функцій для компонента за допомогою Composition API.

Метод налаштування викликається перед монтуванням компонента, і він повертає об’єкт, що містить реактивні дані та функції, які можна використовувати в шаблоні компонента та коді JavaScript.

Різниця між Option та Composition API

Composition API дозволяє групувати пов’язані дані та функції разом, полегшуючи керування та міркування щодо поведінки компонента, коли він стає складнішим.

Що таке слоти?

Slot

У Vue 3 **slot** є потужною функцією для створення компонентів і передачі вмісту між ними.

Слоти дозволяють визначити заповнювач у шаблоні компонента, який можна заповнити вмістом із батьківського компонента.

Що таке слоти?

Ми поговоримо про такі типи слотів:

1. default slot → Слоти за замовчуванням дозволяють передавати вміст до основної області вмісту компонента.
2. named slot → Іменовані слоти дозволяють передавати вміст до певного заповнювача в шаблоні компонента.
3. named scoped slot → Іменовані масштабовані слоти дозволяють передавати дані від батьківського компонента до дочірнього компонента за допомогою іменованого слота.
4. dynamic slot name → Динамічні слоти дозволяють використовувати обчислене (динамічне) значення як ім'я слотів.

Що таке слоти?

Підсумуємо:

Слоти є потужною функцією у Vue 3, яка дозволяє створювати гнучкі компоненти для багаторазового використання. Вони дозволяють створити більш модульний код і спрощують керування поведінкою складних компонентів.

Приклад застосування слотів в компонентах.

У цьому прикладі ми визначаємо компонент під назвою Card, який має три іменовані слоти:

- header
- footer
- default slot

```
<!-- Card Component -->
<template>
  <div class="card">
    <div class="header">
      <slot name="header">Default Header</slot>
    </div>
    <div class="content">
      <slot></slot>
    </div>
    <div class="footer">
      <slot name="footer">Default Footer</slot>
    </div>
  </div>
</template>
```

Приклад застосування слотів в компонентах.

Default slot

У цьому прикладі ми використовуємо компонент Card і заповнюємо слот за замовчуванням вмістом за допомогою елемента <button> з атрибутом slot, встановленим як нижній колонтитул.

```
<!-- Default Slots -->
<template>
  <Card>
    <h2>Custom Header</h2>
    <p>Card content goes here.</p>
    <button slot="footer">Submit</button>
  </Card>
</template>
```

Приклад застосування слотів в компонентах.

Named slot

Іменовані слоти визначаються за допомогою елемента <slot> з атрибутом name, встановленим для імені слота.

Слот за замовчуванням визначається за допомогою елемента <slot> без атрибута імені.

```
<!-- Named Slots -->
<template>
  <Card>
    <template v-slot:header>
      <h2>Custom Header</h2>
    </template>
    <p>Card content goes here.</p>
    <template v-slot:footer>
      <button>Submit</button>
    </template>
  </Card>
</template>
```

У цьому прикладі ми використовуємо компонент Card і заповнюємо вказані слоти вмістом за допомогою елемента <template> з директивою v-slot. Директива v-slot використовується для визначення імені слота.

Приклад застосування слотів в компонентах.

Dynamic Named slot

Dynamic Named slot дозволяють використовувати обчислене значення як ім'я слотів. Ви можете використовувати синтаксис [slot], щоб указати назву слота як вираз JavaScript.

У цьому прикладі ми визначаємо компонент із динамічним іменем слота. Ми використовуємо обчислену властивість для генерації імені слота на основі певної логіки. Обчислену властивість slotName повертає функція налаштування.

```
<!-- Dynamic Slot Names -->
<template>
  <div>
    <slot :name="slotName"></slot>
  </div>
</template>

<script>
import { computed } from 'vue';

export default {
  setup() {
    const slotName = computed(() => 'dynamic-slot');

    return { slotName };
  }
};
</script>
```

Приклад застосування слотів в компонентах.

Ви можете використовувати **динамічне ім’я слота**, вказавши ім’я слота за допомогою синтаксису [slot]. Ось приклад:

У цьому прикладі ми використовуємо ім’я динамічного слота для передачі вмісту компоненту DynamicComponent. Обчислена властивість slotName використовується для визначення імені слота.

```
<!-- Dynamic Slot Names -->
<template>
  <DynamicComponent :slot="slotName">
      <p>Slot content goes here.</p>
  </DynamicComponent>
</template>
```

Приклад застосування слотів в компонентах.

Named scoped slot

Іменовані масштабовані слоти дозволяють передавати дані від батьківського компонента до дочірнього компонента за допомогою іменованого слота.

Ви можете визначити іменовані слоти з областю дії за допомогою елемента <slot> з атрибутом name, встановленим як ім'я слота, і директивою v-bind для прив'язки даних до слота.

```
<!-- Named Scoped Slot -->
  <template>
  <div class="list">
      <slot name="item" v-for="item in items" v-bind:item="item"></slot>
  </div>
  </template>
```

У цьому прикладі ми визначаємо компонент під назвою list, який має іменований слот із областю дії під назвою item. Ми використовуємо директиву v-for, щоб перебирати масив елементів і прив'язувати кожен елемент до властивості item слота.

Приклад застосування слотів в компонентах.

Ви можете використовувати іменовані слоти з областю видимості, використовуючи елемент <template> з директивою v-slot для вказівки імені слота та директивою v-bind для прив'язки даних до слота.

```
<!-- Named Scoped Slot -->
<template>
  <List :items="items">
    <template v-slot:item="{ item }">
      <li>{{ item }}</li>
    </template>
  </List>
</template>
```

У цьому прикладі ми використовуємо компонент List і передаємо масив елементів як проп. Ми використовуємо іменований слот із областю дії під назвою item для відтворення кожного елемента в списку. Директива v-slot використовується для визначення назви слота, а директива v-bind використовується для прив'язки поточного елемента до слота.

Іменовані слоти з областю видимості корисні, коли ви хочете передати дані від батьківського компонента до дочірнього компонента без використання реквізитів. Вони також корисні, коли ви хочете налаштувати візуалізацію дочірнього компонента без необхідності змінювати його шаблон.

Приклад застосування слотів в компонентах.

На додаток до іменованих слотів і іменованих слотів з областю видимості, Vue 3 також підтримує скорочений синтаксис для іменованих слотів і слотів з областю видимості.

Ось приклад:

```
<!-- Named Scoped Slot -->
<template>
  <Card>
    <h2 slot="header">Custom Header</h2>
    <p>Card content goes here.</p>
    <button slot="footer">Submit</button>
  </Card>
</template>

<template>
  <List :items="items">
    <li v-for="item in items" v-slot:item="{ item }">{{ item }}</li>
  </List>
</template>
```


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