Vue.js: The Progressive JavaScript Framework

Press Space for next page →

Vue is easy

Reactive web development concepts

Templates and Data Binding

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" langage="JavaScript"></script>

<div id = "app">
{{message}}
</div>
```

```
var app = new Vue({
  el: '#app',
  data:{
    message : 'hello Vue!'
  }
})
```

Result JavaScript HTML CSS

Edit in JSFiddle

hello Vue!

Playing with the DOM

```
<script src="https://cdnjs.cloudflare.com/ajax
/libs/vue/2.4.2/vue.js" langage="JavaScript"></script>

<ul id="example-1">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

Result JavaScript HTML CSS Edit in JSFiddle

- Foo
- Bar

Two-Way Data Binding

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" langage="JavaScript"></script>

<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

Result JavaScript HTML CSS

Edit in JSFiddle

Foo

Message is: Foo

Two-Way Data Binding

```
<select v-model="selected" multiple>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
<br>
<span>Selected: {{ selected }}</span>
```

Result JavaScript HTML CSS Edit in JSFiddle

A
B
C

Selected:

Transition Effects

```
<script src="https://cdnjs.cloudflare.com/
ajax/libs/vue/2.4.2/vue.js"
 langage="JavaScript"></script>

<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

```
new Vue({
  el: '#demo',
  data: {
    show: true
  }
})
```

```
.fade-enter-active, .fade-leave-active {
  transition: opacity .5s
}
.fade-enter, .fade-leave-to
```

Result JavaScript HTML CSS Edit in JSFiddle

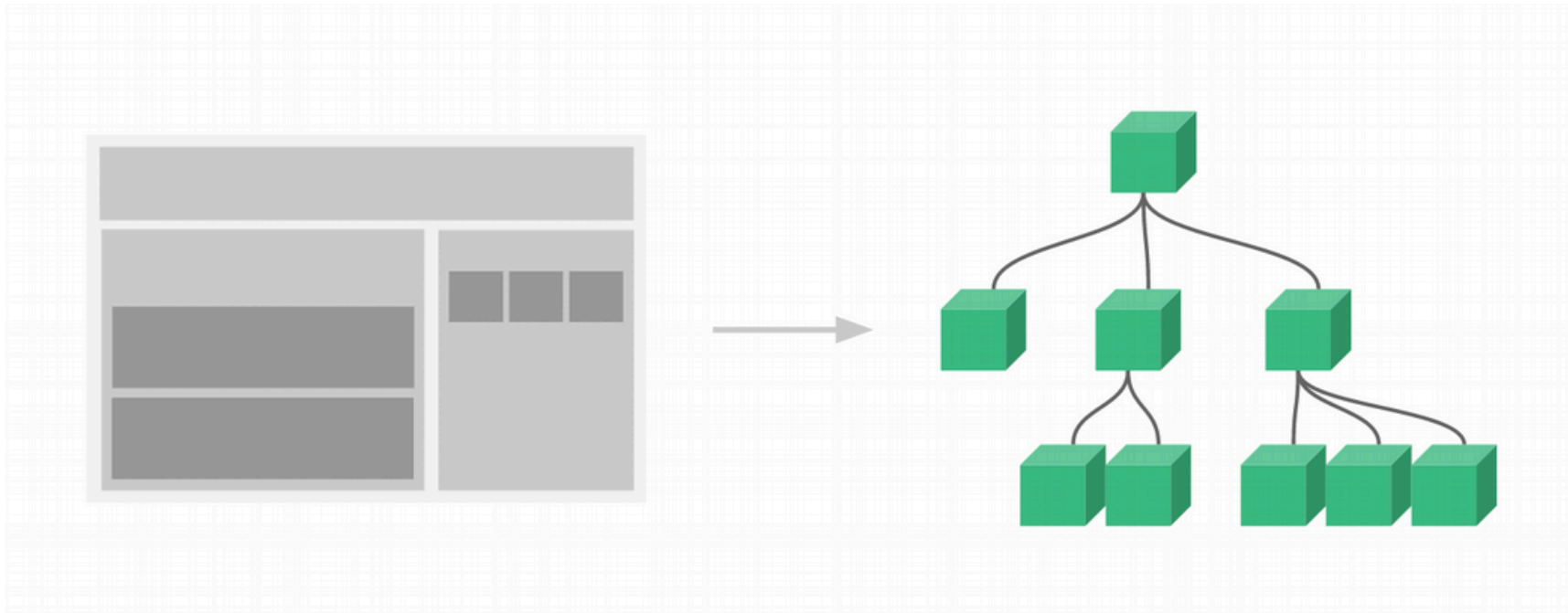
Toggle

hello

Vue can be easily integrated

LARGE or small

Components



Components

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" langage="JavaScript"></script>
<div id="example">
  <my-component></my-component>
</div>
```

```
// register
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
// create a root instance
new Vue({
  el: '#example'
})
```

Clean and Simple

Clean and Simple

- Vue was created by Evan You after working for Google on AngularJS. You later summed up his thought process,

> I figured, what if I could just ** extract the part that I really liked** about Angular and build something **really lightweight** without all the extra concepts involved?

Concepts très proches d’Angular

- composant
- binding modèle vue
- directive
- props <=> input port
- emit <=> output port
- Ref, reactive, computed, watch <=> signal, effect
- Lifecycle hooks
- Router

Compatible avec TypeScript

- Demo

```
npm create vite@latest
cd demovue
npm i
npm run dev
```

Un modèle de développement propre

1 composant = 1 fichier .vue avec trois parties (*SFC (single file component)*)

```
<script setup lang="ts">
import HelloWorld from './components/HelloWorld.vue'
</script>

<template>
  <div>
    <a href="https://vitejs.dev" target="_blank">
      <img src="/vite.svg" class="logo" alt="Vite logo" />
    </a>
    <a href="https://vuejs.org/" target="_blank">
      <img src="./assets/vue.svg" class="logo vue" alt="Vue logo" />
    </a>
  </div>
  <HelloWorld msg="Vite + Vue" />
</template>

<style scoped>
.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
```

Partie 1 Script

Code TS permettant de

- définir les **props** et les **emits** (équivalent input output port d'angular)
- definir les **ref** (signal en angular), les **reactive** (signal pour types complexes), les **computed** (computed signal), les **watch** (effect en angular) => Base de la programmation reactive
- définir les **Lifecycle Hooks**
- définir les **provide** / **inject**

⚠️: Attention dans la suite nous utiliserons que l'API composition (la nouvelle) en comparaison de l'API Options plus proche de Vue 2

Props et Emits

Props

```
defineProps({
  greetingMessage: String
})
```

```
<BlogPost title="My journey with Vue" />
```

You've also seen props assigned dynamically with v-bind or its : shortcut, such as in:

```
<BlogPost :title="post.title" />
```

Emit

```
const emit = defineEmits<{
  increase-by: [id: number]
  update: [value: string]
}>()
```

Emit an event from code (to the parent)

```
emit("increase-by",23)
```

Emit an event from template (to the parent)

```
<button @click="$emit('increase-by', 1)"> emit 1 </button>
```

Capture the event from the parent

```
<MyButton @increase-by="(n) => count += n" />
```

Ref, reactive, computed, watch

Ref

In Composition API, the recommended way to declare reactive state is using the ref() function:

```
import { ref } from 'vue'

const count = ref(0)
console.log(count) // { value: 0 }
console.log(count.value) // 0
count.value++
console.log(count.value) // 1
```

Close to signal in Angular

Reactive

There is another way to declare reactive state, with the reactive() API. Unlike a ref which wraps the inner value in a special object, *reactive*() makes an object itself reactive:

```
import { reactive } from 'vue'

const state = reactive({ count: 0 })
```

Reactive is only for complex object

Refs, reactive, computed, watch

Without computed property

In Composition API, you can declare computed property

```
const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})
```

And we want to display different messages depending on if author already has some books or not:

```
<p>Has published books:</p>
<span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
```

With computed property

```
<script setup lang="ts">
import { reactive, computed } from 'vue'

const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})

// a computed ref
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
})
</script>

<template>
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

Refs, reactive, computed, watch

watch

Computed properties allow us to declaratively compute derived values. However, there are cases where we need to perform "side effects" in reaction to state changes - for example, mutating the DOM, or changing another piece of state based on the result of an async operation.

With Composition API, we can use the watch function to trigger a callback whenever a piece of reactive state changes:

```
<script setup lang="ts">
import {  ref, watch } from 'vue'
const count = ref(0)
const increment = ()=>  count.value = count.value  +2
watch(count, async (newValue, oldValue) => {
  console.error('newValue',newValue);
  }
)
</script>
<template>
    <button type="button" @click="increment()">count is {{ count }}</button>
</template>
```

Lifecycle Hooks

For example, the onMounted hook can be used to run code after the component has finished the initial rendering and created the DOM nodes:

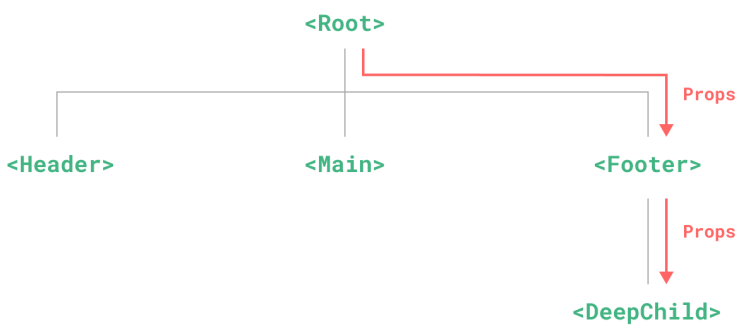
```
<script setup lang="ts">
import { onMounted } from 'vue'

onMounted(() => {
  console.log(`the component is now mounted.`)
})
</script>
```

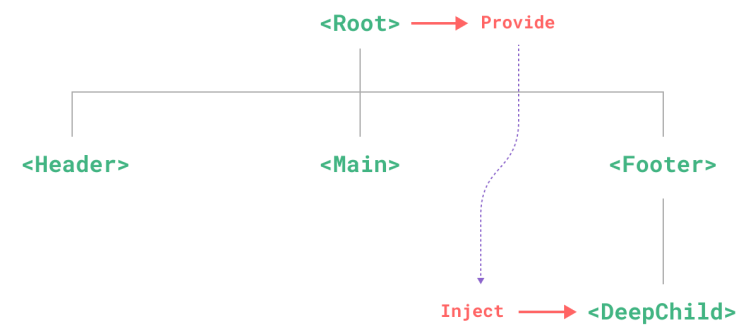
- onMounted()
- onUpdated()
- onUnmounted()
- onBeforeMount()
- onBeforeUpdate()
- onBeforeUnmount()
- onErrorCaptured()
- onRenderTracked()
- onRenderTriggered()
- onActivated()
- onDeactivated()
- onServerPrefetch()

provide / inject

With props



With provide inject



provide / inject

provide

```
<script setup lang="ts">
import { provide } from 'vue'

provide(/* key */ 'message', /* value */ 'hello!')
</script>
```

App-level Provide

In addition to providing data in a component, we can also provide at the app level:

```
import { createApp } from 'vue'
const app = createApp({})
app.provide(/* key */ 'message', /* value */ 'hello!')
```

inject

```
<script setup lang="ts">
import { inject } from 'vue'
const message = inject('message')
</script>
```

Component registration

Global Registration

```
import MyComponent from './App.vue'
const app = createApp({})

app.component('MyComponent', MyComponent)
  .component('ComponentA', ComponentA)
  .component('ComponentB', ComponentB)
  .component('ComponentC', ComponentC)
```

```
<!-- this will work in any component inside the app -->
<ComponentA/>
<ComponentB/>
<ComponentC/>
```

Local Registration

When using SFC (single file component) with *script setup*, imported components can be locally used without registration:

```
<script setup lang="ts">
import ComponentA from './ComponentA.vue'
</script>

<template>
  <ComponentA />
</template>
```

Better to use local registration