



まもなくやってくる

# Vue.js 3

PWA night conference 2020

2020.02.01

@kazupon

# 自己紹介





# kazupon

PLAID, inc.

Vue.js Core Team Member

Vue.js Japan User Group Organizer

Creator of Vue I18n & Intlify

WebAssembly Love ❤️



@kazu\_pon



kazupon

はじめに

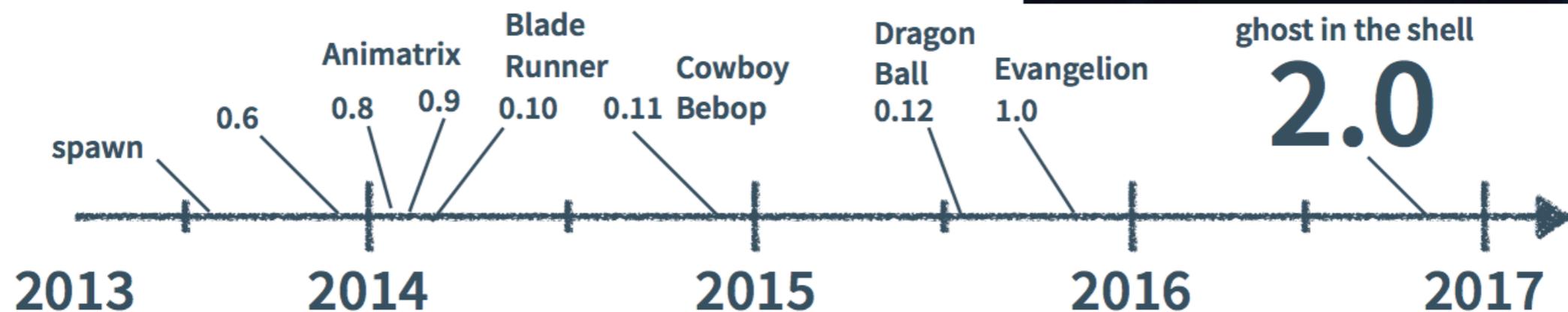
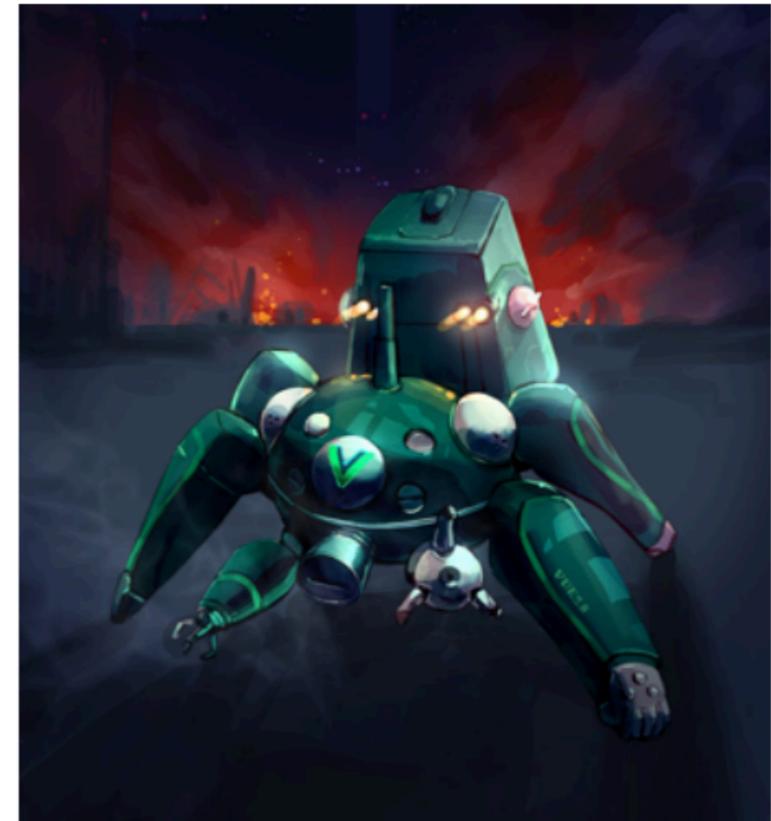
2016年10月

# Vue.js 2.0 リリース

- 2016年10月1日(日本時間)  
コードネーム: Ghost in the Shell  
リリース!

※イラスト著作者: @hashedrock 氏

<https://twitter.com/hashedrock/status/782069763358924800>



それから

3年余りを歴て

2020年 Q1

**Vue.js 3.0**  
**リリース予定**

現在

# RFCで最終仕様を固めつつ着々と開発中

5 Q1 2020

- 2.x-next
  - Added by yyx990803
- 3.0: Migration
  - 2.x Compat build
  - Upgrade guide
  - Migration tools
  - Added by yyx990803
- 3.0: Release Management
  - Regression testing for 3.0
  - Automated nightly release
  - Formalize release lifecycle
  - Added by yyx990803

vuejs / rfcs

Watch 292 Star 1.7k Fork 157

Code Issues 14 Pull requests 13 Security Insights

RFCs for substantial changes / feature additions to Vue core

49 commits 22 branches 0 packages 0 releases 8 contributors

Branch: master New pull request Find file Clone or download

yyx990803 update(slots-unification): fix example Latest commit ea9b219 2 hours ago

|                  |  |               |
|------------------|--|---------------|
| active-rfcs      | update(slots-unification): fix example | 2 hours ago   |
| 0000-template.md | init                                   | 11 months ago |
| README.md        | Fix a typo (#79)                       | 3 months ago  |

README.md

## Vue RFCs

### What is an RFC?

The "RFC" (request for comments) process is intended to provide a consistent and controlled path for new features to enter the framework.

<https://github.com/vuejs/rfcs>

# vue-next-webpack-preview

vuejs / vue-next-webpack-preview

Watch 3 Unstar 100 Fork 18

Code Issues 3 Pull requests 2 Actions Projects 0 Wiki Security Insights

No description, website, or topics provided.

4 commits 1 branch 0 packages 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

yyx990803 bump versions Latest commit f08b0b3 26 days ago

| File              | Commit        | Time        |
|-------------------|---------------|-------------|
| src               | readme        | last month  |
| .gitignore        | init          | last month  |
| README.md         | readme        | last month  |
| index.html        | init          | last month  |
| package.json      | bump versions | 26 days ago |
| webpack.config.js | add alias     | last month  |
| yarn.lock         | bump versions | 26 days ago |

README.md

## vue-next-webpack-preview

Minimal webpack setup for Vue 3 (alpha). This is for preview purposes only. There might be bugs and undocumented behavior differences from v2, which are expected.

Also note that if you are using VSCode, Vetur isn't updated to take advantage of Vue 3's typing yet so intellisense in Vue files may not be fully functional (especially in templates).

<https://github.com/vuejs/vue-next-webpack-preview>

# vue-cli-plugin-vue-next

vuejs / vue-cli-plugin-vue-next

Used by 9 | Unwatch 9 | Unstar 65 | Fork 2

Code | Issues 1 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

No description, website, or topics provided. [Edit](#)

[Manage topics](#)

10 commits | 1 branch | 0 packages | 2 releases | 2 contributors | MIT

Branch: master | [New pull request](#) | [Create new file](#) | [Upload files](#) | [Find file](#) | [Clone or download](#)

rickbeerendonk and posva docs: fix @vue/compiler-sfc typo (#3) Latest commit 156e62c 13 days ago

|                |  |             |
|----------------|--|-------------|
| generator      | feat: use a codemod to transform entryFile, rather than replace it | 24 days ago |
| .gitignore     | Initial commit   | 27 days ago |
| .prettierrc.js | feat: use a codemod to transform entryFile, rather than replace it | 24 days ago |
| LICENSE        | Initial commit   | 27 days ago |
| README.md      | docs: fix @vue/compiler-sfc typo (#3)                              | 13 days ago |
| index.js       | feat: implement minimum functionality                              | 27 days ago |
| package.json   | chore: update repository url                                       | 23 days ago |
| yarn.lock      | feat: use a codemod to transform entryFile, rather than replace it | 24 days ago |

README.md

## vue-cli-plugin-vue-next

A Vue CLI plugin for trying out the Vue 3 alpha.

<https://github.com/vuejs/vue-cli-plugin-vue-next>

# 公式プラグイン/ツール群は対応中

- Vue Router
- Vuex
- Vue DevTools
- Vue Test Utils
- Vetur
- Vue CLI
- Webpack plugin
- ESLint Plugin
- JSX Plugin
- ... etc

メシジャーリリリースなので

新機能 & 変更

たくさんあります

今日話すこと

# 今日話すこと

- Vue.js 3 で入る新機能について
  - コンポジション API
  - フラグメント
  - ポータル
  - サスペンス

# 今日話すこと

- 数ある Vue.js 2.x からの変更の中からは...
  - 単一ファイルコンポーネント
  - スロット
  - フィルタ
  - イベント
  - グローバルAPI

# 今日話さないこと

- Vue.js 本体以外のこと
- Vue 公式プラグイン / ツール系
- Nuxt.js などの関連エコシステム

# ⚠️ 注意事項 ⚠️

- 今日話す内容は、発表時点で Vue.js 本体に実装されているもの
- RFC の状況と今後の alpha、beta バージョンのリリース後の状況次第では変わる可能性はあり

新機能

について

コンポジションAPI

Composition API

# コンポジションAPIとは？

- 関数ベースで提供される API 群



The screenshot shows the 'Vue Composition API' website. The main heading is 'API Reference'. A sidebar on the left lists various API functions: setup, reactive, ref, isRef, toRefs, computed, readonly, watch, Lifecycle Hooks, provide & inject, Template Refs, and defineComponent. A red box highlights this sidebar, and a red arrow points from the text '提供される API たち' (APIs provided) to it. The main content area features a promotional banner for a 'Cheat Sheet' and a section for the 'setup' function, which is described as a new component option for using the Composition API.

Vue Composition API 提供される API たち API Reference

## API Reference

Download the free [Cheat Sheet](#) from Vue Mastery or watch their [Vue 3 Course](#).

### setup

The `setup` function is a new component option. It serves as the entry point for using the Composition API inside components.

- Invocation Timing**

`setup` is called right after the initial props resolution when a component instance is created. Lifecycle-wise, it is called before the `beforeCreate` hook.

<https://vue-composition-api-rfc.netlify.com/api.html>

# 生まれた背景

- Vue.js で大規模開発はできるが問題があった
  - コンポーネントが大きくなるとコードが読みにくくなる
  - テストがしにくい
  - コード再利用のパターン化がしにくい
  - TypeScript による型のサポートが受けにくい

# コンポジションAPIを使うことで...

- コンポーネントのロジックを自由度高く組み合わせ実装することができる

```
<template>
  <button @click="increment">
    カウントは {{ state.count }}、それを2倍すると {{ state.double }}
  </button>
</template>

<script>
import { reactive, computed } from 'vue'

export default {
  setup() {
    const state = reactive({
      count: 0,
      double: computed(() => state.count * 2)
    })

    function increment() {
      state.count++
    }

    return {
      state,
      increment
    }
  }
}
</script>
```

# 大規模開発がより快適に！

- ・ コンポーネントのロジック構成が綺麗になり、テストもしやすくなる！

Options API



The screenshot shows the Options API code structure. It is organized into several distinct, color-coded blocks: a top orange block for the root component, followed by a green block for the first child, a cyan block for the second child, a purple block for the third child, an orange block for the fourth child, a cyan block for the fifth child, a purple block for the sixth child, a cyan block for the seventh child, a pink block for the eighth child, and a green block for the ninth child. Each block contains code for the component's options, data, and methods, demonstrating a clear and consistent organizational pattern.

Composition API



The screenshot shows the Composition API code structure. It is organized into several distinct, color-coded blocks: a top orange block for the root component, followed by a purple block for the first child, an orange block for the second child, a cyan block for the third child, a pink block for the fourth child, and a green block for the fifth child. Each block contains code for the component's options, data, and methods, demonstrating a clear and consistent organizational pattern.

どんなものか  
見ていきましよう！

# 提供される主な API の種類

- ・ 主に以下の内容を提供するものに分類される
  - ・ リアクティブ状態の操作
  - ・ コンポーネントの新エントリポイント
  - ・ ライフサイクルフック
  - ・ DI (Dependency injection: 依存性の注入)

# リアクティブ 状態の操作

# 状態操作とユーティリティが提供される

- reactive
- computed
- watch
- readonly
- ref
- isRef
- toRefs

# 新しい概念 Ref が導入される

- リアクティブな状態をラップした参照オブジェクト



# コード例

```
import { reactive, computed, watch } from 'vue'

// reactive は Proxy を返す
const state = reactive({ count: 1 })
console.log(state)

// computed は ref オブジェクトでラップしたものを返す
const double = computed(() => state.count*2 )
console.log(double.value) // .value にリアクティブな値が入っている

// ここで count を変えると、watch が呼ばれる
watch(() => {
  document.getElementById('app').innerHTML = `countは${state.count}, 2倍すると${double.value}`
})
state.count = 2 // `countは2, 2倍すると4` と app に表示される
```

コンピューターネットワーク

の

新エントリポイント

# setup

- コンポジション APIを使ってコンポーネントのロジックを実装
- return したものがテンプレートのレンダリングコンテキストになる

```
<template>
  <button @click="increment">
    カウントは {{ state.count }}、それを2倍すると {{ state.double }}
  </button>
</template>

<script>
import { reactive, computed } from 'vue'

export default {
  setup() {
    const state = reactive({
      count: 0,
      double: computed(() => state.count * 2)
    })

    function increment() {
      state.count++
    }

    return {
      state,
      increment
    }
  }
}
</script>
```

新しい  
エントリポイント

# render 関数 / JSX も使用できる

```
<script>
import { h, ref, reactive } from 'vue'

export default {
  setup () {
    const count = ref(0)
    const obj = reactive({ msg: 'hello world!' })

    return () => h('div', [
      count.value,
      obj.msg
    ])
  }
}
</script>
```

# 引数に props と context が渡される

- props:  
コンポーネントで定義されたプロパティ
- context:  
コンポーネントでアクセスできるプロパティ
  - attrs: Vue 2.x の \$attrs に相当
  - emit: Vue 2.x の \$emit に相当
  - slots: Vue 2.x の \$slots に相当

# props と context の使用例

```
import { ref } from 'vue'

export default {
  props: {
    name: String
  },
  setup (props, { attrs, slots, emit }) {
    const count = ref(0)

    function onClick() {
      const old = count.value
      count.value++
      emit('increment', { old, val: count.value })
    }

    watch(() => console.log(`name is: ${props.name}`))

    return () => h('div', [
      h('p', [`count is ${count.value}`]),
      h('button', { onClick }, slots)
    ])
  }
}
```

# this はコンポーネントのインスタンスではない

- ・ コンポジション API で setup でコンポーネントを実装する場合は、this はもう使わない

```
export default {
  props: {
    name: String
  },
  setup () {
    function onClick () {
      // Vue 2.x のような使い方は NG !!
      console.log(`name is ${this.name}`)
    }
  }
}
```

ライフサイクル

フック

# Vue 2.x とのマッピング

|               |   |                 |
|---------------|---|-----------------|
| beforeCreate  | → | setup           |
| created       | → | setup           |
| beforeMount   | → | onBeforeMount   |
| mounted       | → | onMounted       |
| updated       | → | onUpdated       |
| beforeDestroy | → | onBeforeUnmount |
| destroyed     | → | onUnmounted     |
| errorCaptured | → | onErrorCaptured |

(左: Vue 2.x、右: Vue 3 以降)

# 新しいライフサイクルフックの使用例

```
import { onMounted, onUpdated, onUnmounted } from 'vue'

export default {
  setup () {
    onMounted(() => {
      console.log('mounted!')
    })
    onUpdated(() => {
      console.log('updated!')
    })
    onUnmounted(() => {
      console.log('unmounted!')
    })
  }
}
```

# デバッグ向けに新フック

- レンダリング処理に対して以下のフックが提供される
  - onRenderTracked
  - onRenderTriggered

```
import { onRenderTriggered } from 'vue'

export default {
  setup () {
    onRenderTriggered (e => {
      debugger
    })
  }
}
```

**DI**

**(Dependency Injection)**

# provide / inject

- Vue 2.x でも提供されていた DI を Vue 3.0 でも同じように利用できる

```
import { provide, inject, reactive } from 'vue'

// 依存を提供する側 (provider)
const ThemeSymbol = Symbol()

const Ancestor = {
  setup () {
    provide(ThemeSymbol, 'dark')
  }
}

// 依存を利用する側 (consumer)
const Descendent = {
  setup () {
    const theme = inject(ThemeSymbol)
    return { theme }
  }
}
```

# リアクティブな状態も共有できる

- ref でラップしたリアクティブなオブジェクトを渡すことで共有できる

```
import { provide, inject, ref } from 'vue'

// 依存を提供する側 (provider)
const themeRef = ref('dark')
provide(ThemeSymbol, themeRef)

// 依存を利用する側 (consumer)
const theme = inject(ThemeSymbol, ref('light'))
watch(() => {
  console.log(`theme set to: ${theme.value}`)
})
```

# プラグインは DI で実装を推奨

- Vue 3.0 以降では、プラグイン開発は provide / inject を使ったパターンを推奨
- Vue 2.x までの問題点
  - \$xxx で依存が注入されるため、型サポートが難しい
  - プラグインで実装されたコンポーネントは、Vue.use しないと動かない

# DI の応用例

- 独自 Store

```
// コンポーネント (xxx.vue)
import useStore from 'store'

export default {
  setup () {
    const { foo } = useStore({ foo: '...' })

    // 何らかのゴニョゴニョ...

    // Store の状態を return
    return { foo }
  }
}
```

```
// Store ライブラリ (store.js)
import { provide, inject, reactive, ref } from 'vue'

const StoreSymbol = Symbol()

export function provideStore (state) {
  const store = reactive(state)
  provide(StoreSymbol, toRefs(store))
}

export function useStore (defaultState) {
  const store = inject(StoreSymbol, ref(defaultState))
  if (!store) {
    // 何らかのエラー
  }
  return store
}
```

プロジェクト内でも provide / inject で DI すれば便利になることもある  
(ご利用は計画的に)

既存 API との  
併用について

# コンポジションAPIを併用できる！

- 併用した場合のコンポジション API の挙動
  - Vue 2.x のオプション API の前に解決される
  - なので、`setup` 内で `this.xxx` は不可
- オプション API 側からはコンポジション API で定義されたプロパティなどにアクセス可

コンポジション API

デメリット

# リアクティブデータの扱いが少し面倒に

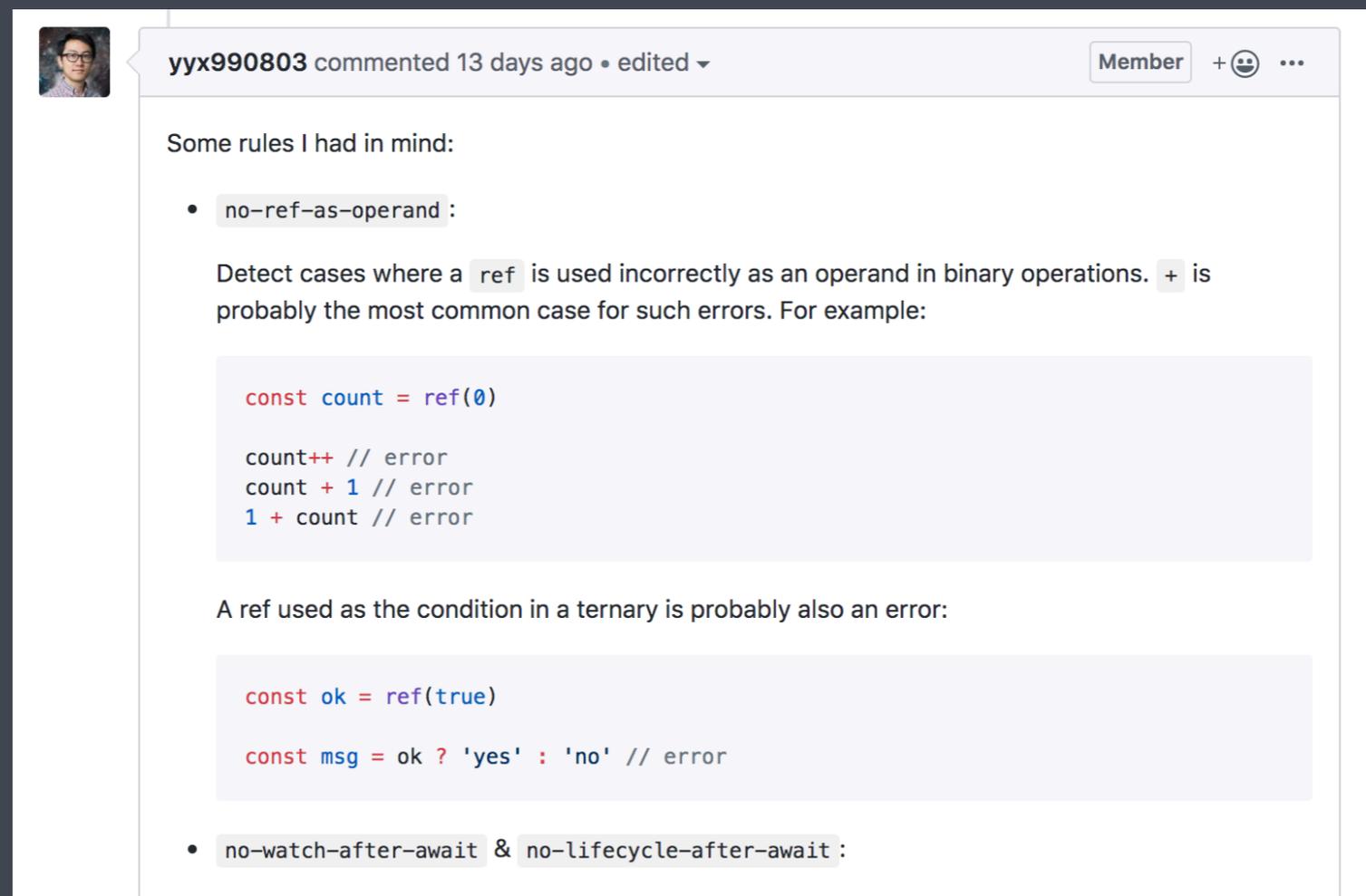
- Refs によるリアクティブな状態をラップするという概念が混乱をきたす
- データが reactive なのものか、それとも ref でラップされてものかどうか意識が必要
- そのために変数の命名規則が必要になる可能性がある

# 注意しないとコードがカオスになりやすい

- JS / TS で、`setup` にロジックを実装&構成してコンポーネント、ライブラリを書いていく

# eslint-plugin-vue でルールを提供する

- ・ カオスにならないためにコンポジション API 向けのルールを提供する予定



yyx990803 commented 13 days ago • edited ▾ Member + 😊 ...

Some rules I had in mind:

- `no-ref-as-operand` :  
Detect cases where a `ref` is used incorrectly as an operand in binary operations. `+` is probably the most common case for such errors. For example:

```
const count = ref(0)  
  
count++ // error  
count + 1 // error  
1 + count // error
```

A ref used as the condition in a ternary is probably also an error:

```
const ok = ref(true)  
  
const msg = ok ? 'yes' : 'no' // error
```

- `no-watch-after-await` & `no-lifecycle-after-await` :

<https://github.com/vuejs/eslint-plugin-vue/issues/1035>

フラグメント

**Fragments**

# フラグメントとは

- コンポーネントが複数の要素を Root 要素でラップしなくても、まさに断片として返すことができる要素群のこと
- Vue 3.0 以降はデフォルトでサポート

```
<template>
```

```
<header>  
  <p>これは header </p>  
</header>      fragment
```

```
<div class="main">  
  <p>こんにちは! </p>  
</div>         fragment
```

```
<footer>  
  <p>これは footer </p>  
</footer>     fragment
```

```
</template>
```

# Root 要素でラップする問題点

- Invalid な HTML になってしまう
  - Semantic な HTML にならない
- a11y対応ができなくなる
- 不必要に div 要素を積んでしまう

# 問題点のコード例

```
<!-- Table.vue -->
<template>
  <table>
    <tr>
      <columns />
    </tr>
  </table>
</template>

<!-- Column.vue -->
<template>
  <div>
    <td>Hello</td>
    <td>World</td>
  </div>
</template>
```

レンダリング



```
<table>
  <tr> 不正な HTML ...
    <div>
      <td>Hello</td>
      <td>World</td>
    </div>
  </tr>
</table>
```

# フラグメントを利用すると...

```
<!-- Table.vue -->  
<template>  
  <table>  
    <tr>  
      <columns />  
    </tr>  
  </table>  
</template>
```

```
<!-- Columns.vue -->  
<template>  
  <td>Hello</td>  
  <td>World</td>  
</template>
```

レンダリング



```
<table>  
  <tr> 正しいHTML!  
    <td>Hello</td>  
    <td>World</td>  
  </tr>  
</table>
```

# 余談: Vue 2.x では関数型コンポーネントで回避できた

```
<!-- Table.vue -->
<template>
  <table>
    <tr>
      <columns />
    </tr>
  </table>
</template>

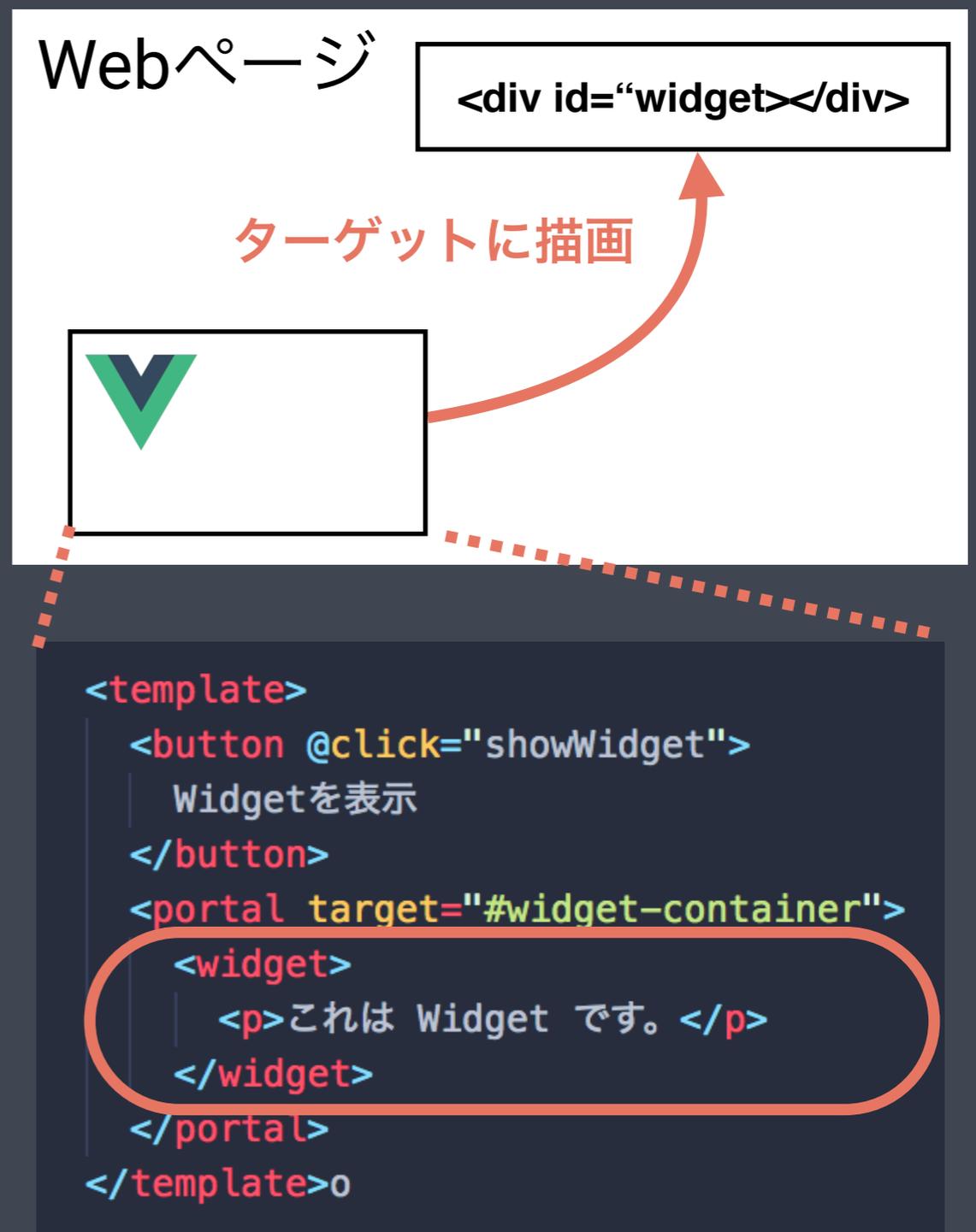
<!-- Columns.vue -->
<script>
export default {
  name: 'Columns',
  functional: true,
  render: h => [h('td', ['hello']), h('td', ['world'])]
}
</script>
```

ポータル

Portals

# ポータルとは

- コンポーネント内のあるコンテンツをVueの描画対象外の要素に描画するための仕組み
- WICG の Portals とは似ているがちょっと違うので注意



# ユースケース

- スタイルレイアウトを Hack しそうな場合  
z-index、position: absolute 駆使したスタイル処理から開放される
- ページ上のウィジェット  
外部の何らかのウィジェットを動的に更新する必要があるもの

# Vue 2.x でポータルを使用するには ...

- プラグイン vue-portal を使う必要があった



<https://github.com/LinusBorg/portal-vue>

- Vue 3.0 以降では、仮想 DOM レベルでポータルが実装されている

# 使い方

- ・ 対象となるコンテンツを Portal コンポーネントでラップ
- ・ target プロパティで対象先の要素を指定するだけ

# 使い方: コード例

```
<template>
  <button @click="showWidget">
    Widgetを表示
  </button>
  <Portal target="#widget-container">
    <Widget>
      <p>これは Widget です。 </p>
    </Widget>
  </Portal>
</template>
```

```
<script>
import { Portal } from 'vue'
import Widget from './widget.vue'

export default {
  components: {
    Widget,
    Portal
  }
}
</script>
```

# target プロパティ

```
<!-- query selector 形式で指定する -->  
<Portal target="#some-id" />  
<Portal target=".some-class" />  
<Portal target="[data-portal]" />  
  
<!-- v-bind で dynamic にできる -->  
<Portal v-bind:target="targetName" />
```

# Portal という名前変わるかも...

- WICG の Portals とコンフリクトしている  
& 詳細仕様詰めるため、現在 RFC で議論

The screenshot shows a GitHub pull request titled "Portal functionality #112". At the top, it indicates that LinusBorg wants to merge 3 commits into the master branch from the rfc-portals branch. The pull request has 33 conversations, 3 commits, 0 checks, and 1 file changed. A green bar shows a net change of +354 lines and -0 lines. A comment from LinusBorg, posted 9 days ago, states: "This RFC introduces a <portal> component, which allows to move its slot content to another part of the document." Below the comment are reaction counts: 38 thumbs up, 9 party poppers, 12 hearts, 7 rockets, and 3 eyes. At the bottom, a commit titled "add portals rfc" is shown with the hash 1eb90ee. On the right side, the "Reviewers" section lists potato4d. The "Assignees" section is empty. The "Labels" section shows "3.x" and "core".

<https://github.com/vuejs/rfcs/pull/112>

サスペンション

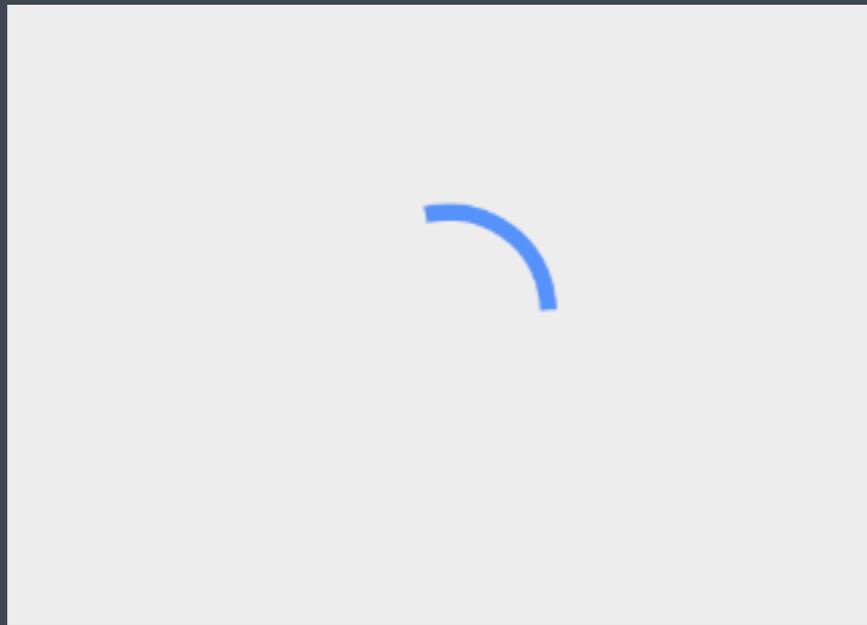
Suspense

# サスペンスとは

- ・ 雑にいうと、コンポーネントの非同期な描画処理をいい感じにやってくれる仕組み

# ユースケース

- 例: Spinner を使ったデータのロード状態の表示



Spinner を回している間に  
非同期にデータを fetch

データを fetch したら  
コンポーネントを描画

- 並列で読み込み箇所が複数でてくるとカオスになりがち

# 使い方

```
// Suspense で使われるコンポーネント
const AsyncComp = {
  setup () {
    const fetchContent = () => {
      return new Promise(resolve => {
        setTimeout(() => {
          resolve('火曜サスペンス劇場！')
        }, 2000)
      })
    }
    // setup では、非同期に render される処理を Promise でラップして
    // そのインスタンスを return
    return new Promise(async (resolve) => {
      const content = await fetchContent()
      resolve(() => h('p', [content]))
    })
  }
}
```

# 使い方

```
export default {
  setup () {
    // Suspense コンポーネントは以下のような感じで使う
    // - default: Suspense されるコンポーネント(VNode)を指定する
    // - fallback: まだ Suspense で解決されていないコンポーネント(VNode)を指定する
    return () => h(Suspense, null, {
      default: h(AsyncComp),
      fallback: h('p', ['21:00です!'])
    })
  }
}
```

ちなみに

素朴に実装すると

```
<template>
  <p v-if="loading">21:00です！</p>
  <p v-else>{{ content }}</p>
</template>

<script>
import { h, reactive, toRefs, onMounted } from 'vue'

export default {
  setup () {
    const state = reactive({
      loading: false,
      content: ''
    })

    const fetchContent = () => {
      return new Promise(resolve => {
        setTimeout(() => {
          resolve('火曜サスペンス劇場！')
        }, 2000)
      })
    }

    onMounted(async () => {
      state.loading = true
      const content = await fetchContent()
      state.content = content
      state.loading = false
    })

    return { ...toRefs(state) }
  }
}
</script>
```

# DEMO

多段非同期コンテンツの読み込み

# サスペンス使える？

- alpha 版では入っているが、ビルドオプションで機能 on/off できるようになっている
- サーバーサイドレンダリングの実装状況次第

# Vue 2.x からの 変更について

単一ファイルコンポーネント

Single File Components

# スコープ付き CSS の仕様変更

- ・ 現在 RFC で最終フィードバック受付中

## Scoped Style Changes #119 Edit

Open yyx990803 wants to merge 3 commits into `master` from `scoped-styles-changes`

Conversation 12 Commits 3 Checks 0 Files changed 1 +116 -0

 yyx990803 commented 9 days ago Member +😊 ⋮

Provide more consistent custom CSS extensions in Single File Component scoped styles.

```
<style scoped>
/* deep selectors */
::v-deep(.foo) {}

/* targeting slot content */
::v-slotted(.foo) {}

/* one-off global rule */
::v-global(.foo) {}
</style>
```

Rendered

 11

**Reviewers** ⚙️  
 posva 🔄 💬

**Assignees** ⚙️  
No one—assign yourself

**Labels** ⚙️  
3.x  
breaking change  
sfc

**Projects** ⚙️  
None yet

**Milestone** ⚙️

<https://github.com/vuejs/rfcs/pull/119>

# Vue 3.0 以降の新仕様

- Vue 独自 CSS 拡張  
疑似クラスを提供
- 既存の記法はコンパイラで警告するが、動作としては互換性をサポート  
予定

```
<style scoped>
/* deep セレクタ */
::v-deep(.foo) {
  /* ... */
}

/* スロットコンテンツ向け */
::v-slotted(.foo) {
  /* ... */
}

/* グローバルなスタイルルール */
::v-global(.foo) {
  /* ... */
}
</style>
```

# ::v-deep() 擬似クラス

- ディープセレクタ (Deep Selector)  
Scoped された子コンポーネントにもスタイルを適用したい場合に使用
- >>>、/deep/、::v-deep 結合子(注:カッコが無い方)は廃止

# ::v-deep() コード例

```
/* foo クラスの要素の bar クラスの子要素にスタイルを当てたい場合 */  
::v-deep(.foo .bar) {  
  color: ■ red;  
}
```

@vue/sfc-compiler + bundler

```
/* 実際のセレクタ */  
[v-data-xxxxxxx] .foo .bar {  
  color: ■ red;  
}
```

# ::v-slotted() 擬似クラス

- 親から子に渡されるスロットコンテンツに子のスタイルが適用されなくなる
- スロットコンテンツには明示的にスタイルの指定が必要

```
<modal>
  <template #header>
    <h3>custom header</h3>
  </template>
</modal>
```

Modal の header スロットに注入されるコンテンツに対してスタイルの適用が必要

```
<!-- Modal.vue -->
<template v-slot="header">
  <header class="header">デフォルトヘッダー</header>
</template>

<div class="modal-body">
  <template v-slot="body">
    <p>デフォルトコンテンツ</p>
  </template>
</div/>

<template v-slot="footer">
  <footer class="footer">デフォルトフッター</footer>
</template>
```

# ::v-slotted() コード例

```
/* スロットコンテンツの foo クラスの要素にスタイルを当てたい場合 */  
::v-slotted(.foo) {  
  color: ■ red;  
}
```

@vue/sfc-compiler + bundler

```
/* 実際のセレクタ */  
[v-data-xxxxxxx-s] .foo {  
  color: ■ red;  
}
```

# ::v-global() 擬似クラス

- グローバルなスタイルを `<style scoped>` 内で定義できる
- 従来は SFC コンポーネント内に `<style scoped>` とは別の `<style>` で定義していた

# ::v-global() コード例

```
/* foo クラスの要素のスタイルをグローバルなスタイルとして追加した場合 */  
::v-global(.foo) {  
  color: ■ red;  
}
```

↓ @vue/sfc-compiler + bundler

```
/* 実際のセレクタ */  
.foo {  
  color: ■ red;  
}
```

スロツット

Slots

# Vue 2.6 から入った v-slot になる

- ・ 従来の以下によるスロット機能は削除される
- ・ slot 属性を使った名前付きスロット
- ・ slot-scope 属性を使ったスコープ付きスロット

フィルタ

Filters

# フィルタは削除

- パイプ演算子 (|) を使ったフィルタは、ついに Vue 3.0 で削除される

```
<!-- Vue 3.0 モダンビルドではサポートされなくなる -->
{{ msg | format }}

<!-- 最小限に変更をしたい場合は、JS の関数で変換しなければならない -->
{{ format(msg) }}
```

- Vue 2.x 互換ビルド版では警告出力し、機能はサポートされる(予定)

# 代替手段

- ESNext stage 1 のパイプライン演算子 (`>`) に変換して使う
- Acorn プラグインで既存のフィルタ構文を Parse & Transform して使う
- `@vue/compiler-core` で生成された render 関数に対して Babel プラグインを Transform して使う

イベント

Events

# Event Emitter 系 API の削除

- コンポーネントインスタンスが提供する以下の API は削除される予定
  - `$on`
  - `$off`
  - `$once`
- Vue 2.x 互換ビルド版では警告出力し、機能はサポートされる(予定)

# 代替手段

- mitt を使う

<https://github.com/developit/mitt>



npm v1.2.0 build passing dependencies none gzip size 212 B

## Mitt

Tiny 200b functional event emitter / pubsub.

- **Microscopic:** weighs less than 200 bytes gzipped
- **Useful:** a wildcard "\*" event type listens to all events
- **Familiar:** same names & ideas as [Node's EventEmitter](#)
- **Functional:** methods don't rely on `this`
- **Great Name:** somehow `mitt` wasn't taken

Mitt was made for the browser, but works in any JavaScript runtime. It has no dependencies and supports IE9+.

グローバル API

Global API

# I/F の再設計により変更

- Vue 2.x のグローバル API の一部で、状態を保持していることにより、以下の問題があった
  - 単体テストしにくい  
状態のリセットもしくはリセットできない  
(Vue.config.errorHandler、Vue.use、Vue.mixin)
  - 同じページ上で複数の Vue インスタンス間で状態を整合させるのが難しい
- また、TreeShaking 最適化の狙いもある

# Vue 2.x と Vue 3.0 の違い

```
import Vue from 'vue'  
import App from './App.vue'
```

**createApp** でインスタンスを生成

```
Vue.config.productionTip = false  
vue.config.ignoredelements = [/^app-/]
```

```
Vue.use(/* ... */)  
Vue.mixin(/* ... */)  
Vue.component(/* ... */)  
Vue.directive(/* ... */)
```

```
const MyComp = Vue.exnted({/* ... */})
```

```
new Vue({  
  render: h => h(App)  
}).$mount('#app')
```

Vue 2.x

```
import { createApp, defineComponent } from 'vue'  
import App from './App.vue'
```

```
const app = createApp(App)
```

```
app.config.ignoredelements = [/^app-/]
```

```
app.use(/* ... */)  
app.mixin(/* ... */)  
app.component(/* ... */)  
app.directive(/* ... */)
```

```
const MyComp = defineComponent({/* ... */})
```

```
app.provide({  
  [ThemeSymbol]: theme  
})
```

```
app.mount('#app')
```

Vue 3.0

# Vue 2.x と Vue 3.0 の違い

```
import Vue from 'vue'  
import App from './App.vue'
```

**productionTip** は Vue 3.0 では削除

```
Vue.config.productionTip = false  
vue.config.ignoredElements = [/^app-/]
```

```
Vue.use(/* ... */)  
Vue.mixin(/* ... */)  
Vue.component(/* ... */)  
Vue.directive(/* ... */)
```

```
const MyComp = Vue.extend({/* ... */})
```

```
new Vue({  
  render: h => h(App)  
}).$mount('#app')
```

Vue 2.x

```
import { createApp, defineComponent } from 'vue'  
import App from './App.vue'
```

```
const app = createApp(App)
```

```
app.config.ignoredElements = [/^app-/]
```

```
app.use(/* ... */)  
app.mixin(/* ... */)  
app.component(/* ... */)  
app.directive(/* ... */)
```

```
const MyComp = defineComponent({/* ... */})
```

```
app.provide({  
  [ThemeSymbol]: theme  
})
```

```
app.mount('#app')
```

Vue 3.0

# Vue 2.x と Vue 3.0 の違い

```
import Vue from 'vue'  
import App from './App.vue'
```

```
Vue.config.productionTip = false  
vue.config.ignoredElements = [/^app-/]
```

```
Vue.use(/* ... */)  
Vue.mixin(/* ... */)  
Vue.component(/* ... */)  
Vue.directive(/* ... */)
```

この辺は  
ほぼそのまま

```
const MyComp = Vue.extend({/* ... */})
```

```
new Vue({  
  render: h => h(App)  
}).$mount('#app')
```

Vue 2.x

```
import { createApp, defineComponent } from 'vue'  
import App from './App.vue'
```

```
const app = createApp(App)
```

```
app.config.ignoredElements = [/^app-/]
```

```
app.use(/* ... */)  
app.mixin(/* ... */)  
app.component(/* ... */)  
app.directive(/* ... */)
```

```
const MyComp = defineComponent({/* ... */})
```

```
app.provide({  
  [ThemeSymbol]: theme  
})
```

```
app.mount('#app')
```

Vue 3.0

# Vue 2.x と Vue 3.0 の違い

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false
vue.config.ignoredelements = [/^app-/]
```

```
Vue.use(/* ... */)
Vue.mixin(/* ... */)
Vue.component(/* ... */)
Vue.directive(/* ... */)
```

```
const MyComp = Vue.exnted({/* ... */})
```

```
new Vue({
  render: h => h(App)
}).$mount('#app')
```

Vue 2.x

```
import { createApp, defineComponent } from 'vue'
import App from './App.vue'
```

```
const app = createApp(App)
```

```
app.config.ignoredelements = [/^app-/]
```

```
app.use(/* ... */)
app.mixin(/* ... */)
app.component(/* ... */)
app.directive(/* ... */)
```

```
const MyComp = defineComponent({/* ... */})
```

```
app.provide({
  [ThemeSymbol]: theme
})
```

```
app.mount('#app')
```

Vue 3.0 では名前を変更  
(主な用途は TS で  
型付けをサポートするため)

Vue 3.0

# Vue 2.x と Vue 3.0 の違い

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false
vue.config.ignoredelements = [/^app-/]

Vue.use(/* ... */)
Vue.mixin(/* ... */)
Vue.component(/* ... */)
Vue.directive(/* ... */)

const MyComp = Vue.exnted({/* ... */})
```

エントリポイントレベルの  
provide も提供する

```
new Vue({
  render: h => h(App)
}).$mount('#app')
```

Vue 2.x

```
import { createApp, defineComponent } from 'vue'
import App from './App.vue'

const app = createApp(App)

app.config.ignoredelements = [/^app-/]

app.use(/* ... */)
app.mixin(/* ... */)
app.component(/* ... */)
app.directive(/* ... */)

const MyComp = defineComponent({/* ... */})
```

```
app.provide({
  [ThemeSymbol]: theme
})
```

```
app.mount('#app')
```

Vue 3.0

# Vue 2.x と Vue 3.0 の違い

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false
vue.config.ignoredelements = [/^app-/]

Vue.use(/* ... */)
Vue.mixin(/* ... */)
Vue.component(/* ... */)
Vue.directive(/* ... */)

const MyComp = Vue.exnted({/* ... */})
```

```
new Vue({
  render: h => h(App)
}).$mount('#app')
```

DOM への  
マウント方法は同じ

Vue 2.x

```
import { createApp, defineComponent } from 'vue'
import App from './App.vue'

const app = createApp(App)

app.config.ignoredelements = [/^app-/]

app.use(/* ... */)
app.mixin(/* ... */)
app.component(/* ... */)
app.directive(/* ... */)

const MyComp = defineComponent({/* ... */})

app.provide({
  [ThemeSymbol]: theme
})
```

```
app.mount('#app')
```

Vue 3.0

まとめ

# まとめ

- コンポジション API により大規模開発がよくなります
- 便利な新機能により、アプリケーションの実装の表現力が高まります
- Vue 2.x からの breaking change 的な変更もあるので、今から押さえて備えておきましょう  
(移行ツール・フェーズ提供予定)

最後に

# Vue Fes Japan 2020

- 今年 2020 年もやります！
- 11月開催に向けて活動開始！



ご清聴

ありがとうございました！