

vueを責務分離して書くための Container/Presentationalパタ ーンの紹介





目次

1. 伝えたいこと
2. 話すこと・話さないこと
3. Vue用語と認識の整理
4. Container/Presentationalパターン
5. Container/Presentationalの利点

伝えたいこと

Vueは「1ファイルに全部書
く」
だけじゃない！

責務分離して書ける



2. 話すこと・話さないこと

話すこと

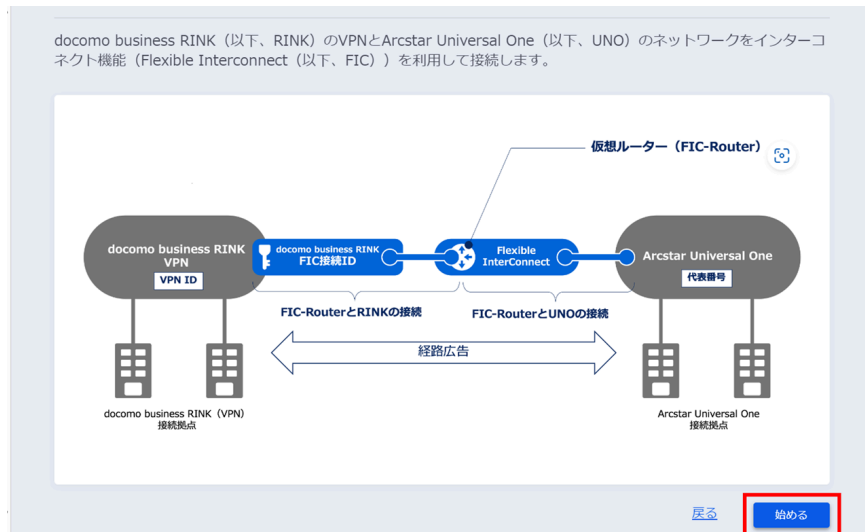
- 案件の簡単な紹介と使用技術
- Vue用語の整理（Reactでいうと」の補足あり）
- **Container/Presentational**パターンでの**Vue**責務分離方法

話さないこと

- 案件の詳細（プロダクト、開発体制など）
- Container/Presentationalパターンの是非（他パターンとの比較はしません）

3. 案件の簡単紹介 & 使用技術


- NTTコミュニケーションズの案件です (7月より docomo business に社名変更)
- 超簡単にいうと、**GUIでインフラを構築するサービス**を作っています



- 技術スタック: **Nuxt, Vue, TypeScript**, JavaScript, vistest, jest, historie, playwright etc...



3. 用語の整理

Vue.jsとは？ 



- Evan You氏によるJavaScriptフレームワーク。
- HTML, CSS, JSを1ファイルにまとめ、**コンポーネント指向**で直感的に記述
- **リアクティビティ**: JavaScriptの状態変化を自動追跡し、効率的にDOMを更新

3. 用語の整理

SFC (Single File Component) とは

コンポーネントのロジック (JavaScript) 、テンプレート (HTML) 、スタイル (CSS) を単一ファイルに収めたもの (`*.vue`)

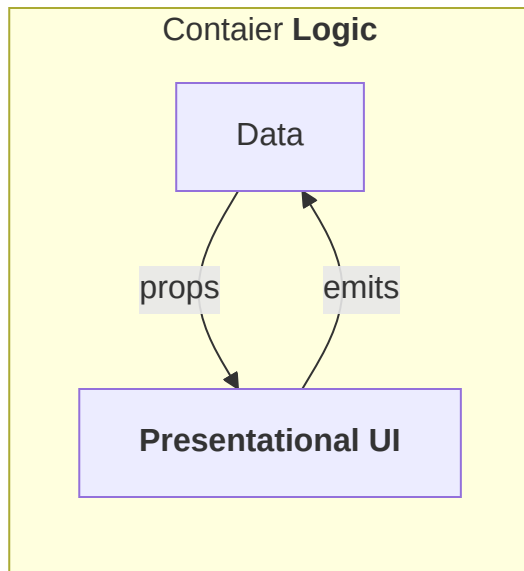
```
<script setup>
import { ref } from 'vue'
const count = ref(0) // リアクティブな変数
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

3. 責務分離 (関心の分離)

- 各コンポーネントが持つ責任を分離すること。
- 今回は**UI (見た目)** と **ロジック (処理)** の分離に絞って話します。



4. Container/Presentationalパターン

処理(logic)と見た目(UI)分ける設計パターン

Container

- ロジックを責務とする
- データフェッチ、イベント処理、状態管理 (LocalStorageなど) に集中
- 例: データを取得してPresentationalに渡す

Presentational

- UI (見た目) を責務とする
- 親から受け取った値の描画に集中
- UIイベント (クリックなど) を親に通知
- 例: 受け取ったデータを表示するボタン

4. 親子でのコンポーネント間のやり取り

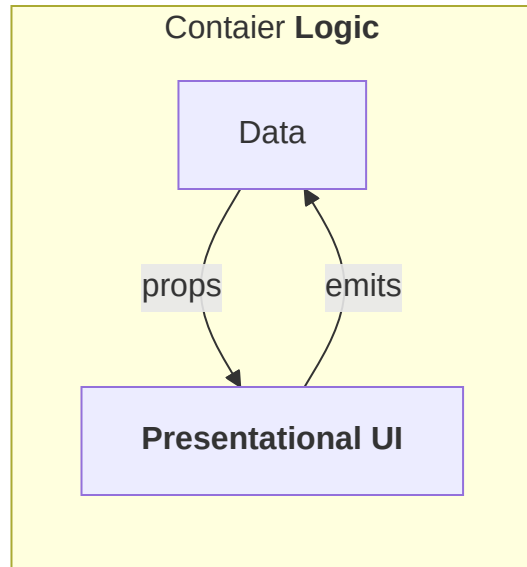
UIとロジックを分離すると、異なるコンポーネント間でデータやイベントをやり取りする方法が必要になります 🚀

1. **Props**: 親から子へデータを渡す

- Presentationalは親から渡されたデータを描画。

2. **Emit**: 子から親へイベントを通知

- PresentationalのUIイベント（ボタンクリックなど）を親に伝える。



4. Container/Presentationalパターン (Propsの例)

ChildComp.vue (Presentational側)

```
<script setup>
const props = defineProps({ msg: String })
</script>

<template>
  <h2>{{ msg || 'No props passed yet' }}</h2>
</template>
```

App.vue (Container側)

```
<script setup>
import { ref } from 'vue'
import ChildComp from './ChildComp.vue'

const greeting = ref('Hello from parent')
</script>

<template>
  <ChildComp :msg="greeting" />
</template>
```

"Hello from parent" が ChildComp に渡されます。

4. Container/Presentationalパターン(Emitの例)

App.vue (Container側)

```
<script setup lang="ts">
import { ref } from 'vue'
import ChildComp from './ChildComp.vue'

const childMsg = ref('No child msg yet')
</script>

<template>
  <ChildComp
    @click:button="(message) => childMsg = message"
  />
  <p>{{ childMsg }}</p>
</template>
```

ChildComp.vue (Presentational側)

```
<script setup lang="ts">
const emit = defineEmits<{
  (event: "click:button", message: string): void
}>()

const handleClick = () => {
  emit('click:button', "Hello from Child"); // イベントを発火
}
</script>

<template>
  <button @click="handleClick">click me</button>
</template>
```

ボタンを押すと「Hello from Child」が表示されます。

4. 補足: Composables (Reactでいう Hooks)

- Vueのリアクティブな状態を持つロジックをカプセル化し、再利用するための関数。
- データフェッチやマウス位置取得など、時間やユーザー操作で変化する要素を管理。
- Containerコンポーネント内で利用し、ロジックをさらに分離・整理できる。

```
// mouse.js
import { ref, onMounted, onUnmounted } from 'vue'

// 慣習として、コンポーザブル関数の名前は "use" で始めます
export function useMouse() {
  // コンポーザブルによってカプセル化および管理される状態
  const x = ref(0)
  const y = ref(0)

  // コンポーザブルは管理している状態を時間の経過とともに更新できます。
  function update(event) {
    x.value = event.pageX
    y.value = event.pageY
  }

  // コンポーザブルは所有コンポーネントのライフサイクルにフックして
  // 副作用のセットアップや破棄することもできます。
  onMounted(() => {
    window.addEventListener('mousemove', update)
  })
  onUnmounted(() => {
    window.removeEventListener('mousemove', update)
  })

  // 管理された状態を戻り値として公開
  return { x, y }
}
```

5. Container/Presentationalの利点

- **責務の明確化:** UIとロジックが分離され、コードの見通しが良くなる。
- **再利用性:** Presentationalコンポーネントは見た目だけなので、様々なロジックを持つContainerコンポーネントから利用可能。
- **テストのしやすさ:**
 - Presentational: PropsとEmitの入出力のみでUIのテストが容易。
 - Container: UIに依存せずロジックの単体テストが可能。
- **開発効率:** チーム開発において、UI担当とロジック担当で分担しやすい。

ここまでのまとめ

🤖一つに全部書いたときの問題点

- Vueは**SFC**(Single File Component) で書ける。
- JavaScriptの状態変化を自動追跡するリアクティブな仕組みを持つ。
- SFCにすべて書くと責務が混在する🤖

🌟解決策

- **Container/Presentational**パターンで**UI**と**ロジック**の責務を分離できる。
 - **Props**: 親から子へ値を渡す。
 - **Emit**: 子から親へ「イベント名」と「値」を通知する。
- **Composables**で状態にかかわるロジックをさらに整理・再利用可能。