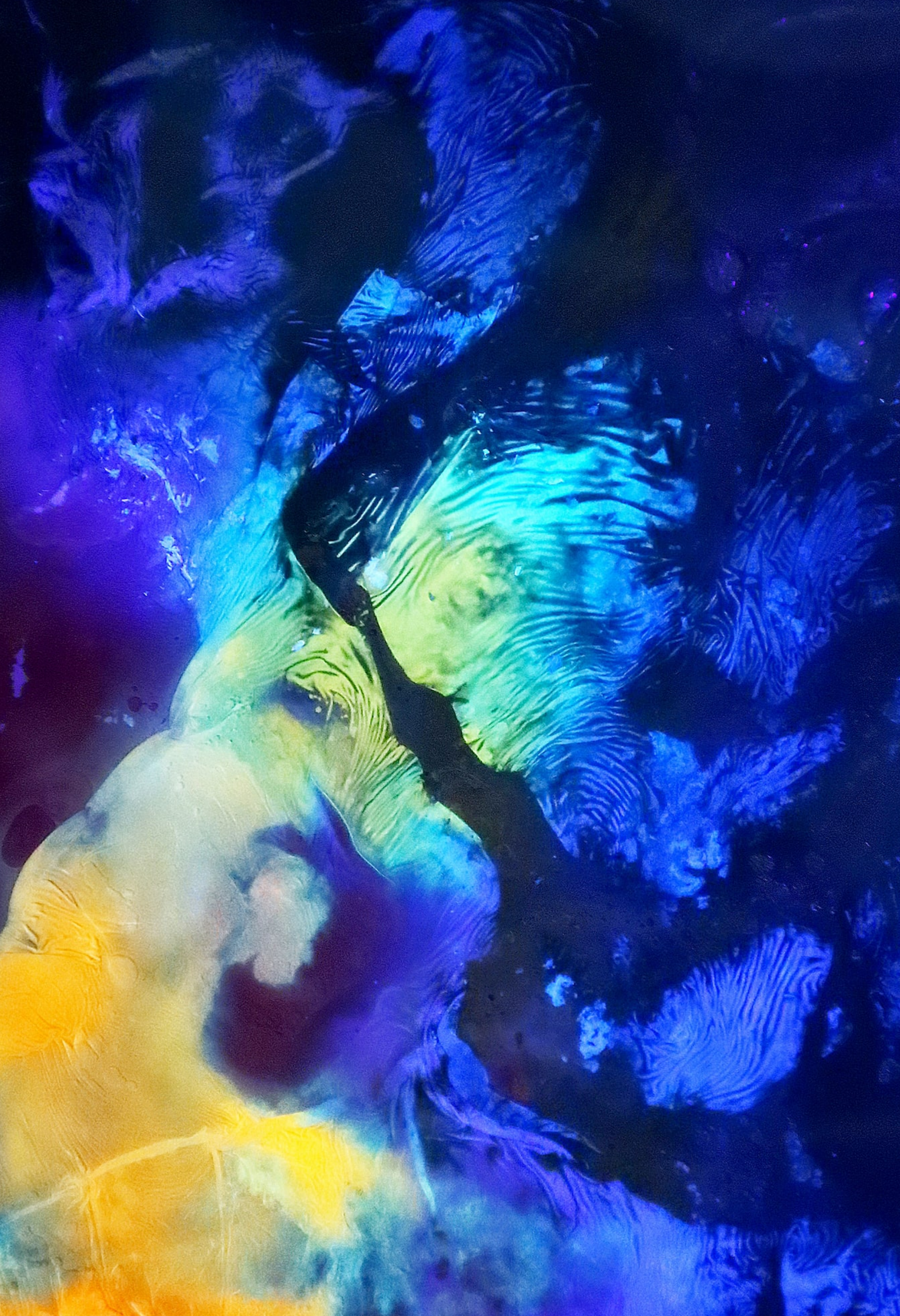




TypeScript 補足資料 Vue.js

An abstract, textured background on the left side of the slide. It features a mix of vibrant colors including deep blues, purples, greens, and yellows, with visible brushstrokes or fiber-like patterns. The right side of the slide is a solid white background.

Vue3 with TypeScript

create-vue



Vue3 ・ ・ TypeScriptで書かれている

TypeScriptでVueを使用する

<https://ja.vuejs.org/guide/typescript/overview.html>

create-vue ・ ・ 公式のセットアップツール(Vite)

VueCLI (webpack)はTypeScriptもサポートしているが
推奨されなくなった

<https://github.com/vuejs/create-vue>

Vue3の環境構築 (create-vue)

```
npm create vue@3  
Project name: vue3-typescript  
Add TypeScript Yes  
Add JSX Support No  
Add Vue Router Yes  
Add Pinia No  
Add Vitest No  
Add and End-to-End No  
Add ESLint No
```

VSCode拡張機能



Vue Language Features(Volar)

TypeScript Vue Plugin (Volar)

(VeturはVue2まで)

フォルダファイル構成

.vscode

dist ・ ・ npm run build 実施してトランスパイル + バンドル + コンパイル

src

assets ・ ・ CSSや画像

components ・ ・ 単一コンポーネント

router ・ ・ ルーティングファイル

views ・ ・ ページコンポーネント

App.vue ・ ・ トップのコンポーネント

main.ts ・ ・ エントリーポイント

env.d.ts ・ ・ 環境定義ファイル

index.html ・ ・ HTMLファイル

vite.config.ts ・ ・ Viteの設定ファイル

tsconfig.jsonは継承関係

extendで継承

referencesでファイル分割

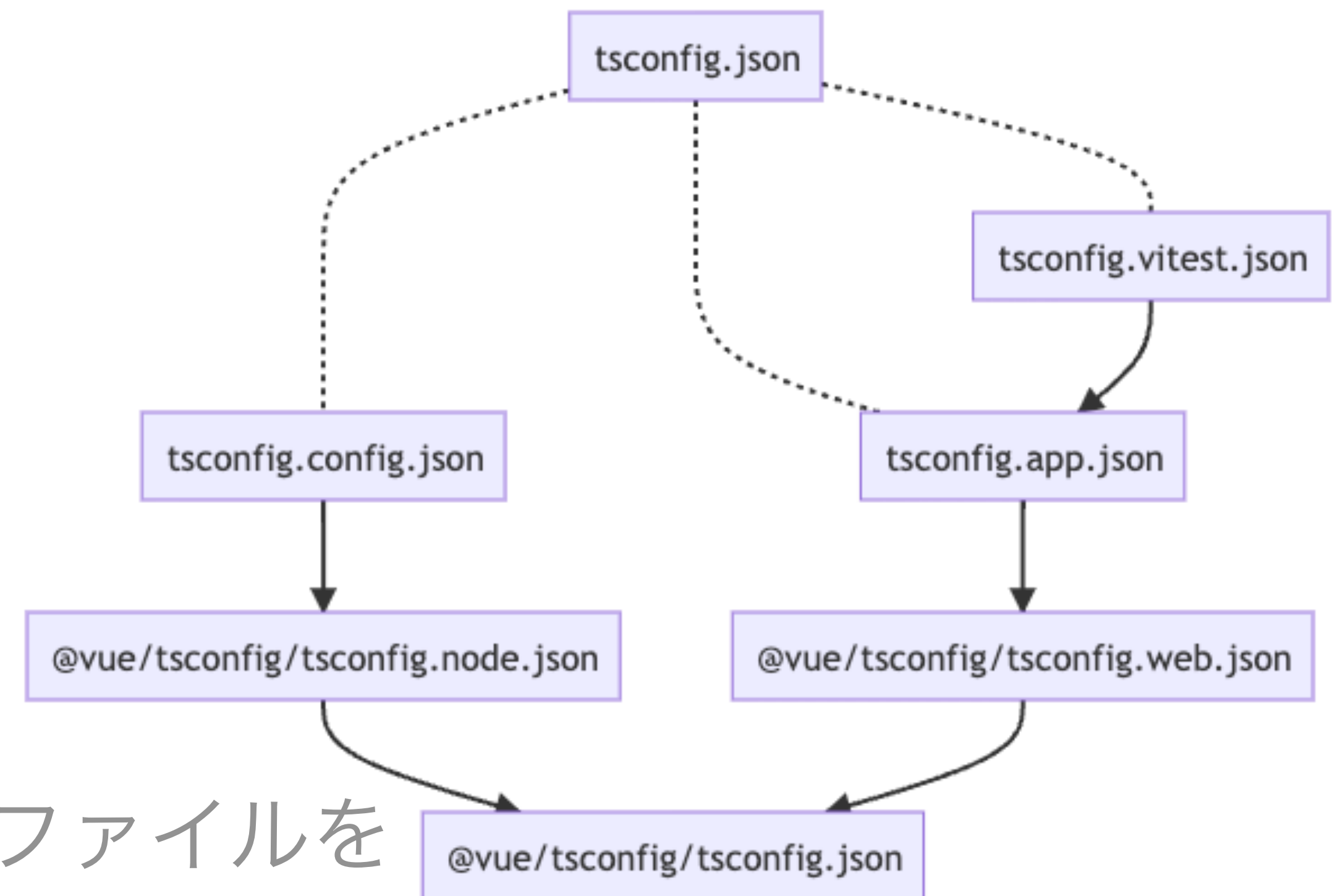
(破線は参照

矢印は継承)

tsconfig.config.json

->Viteなどのツールに関する設定ファイルを

TypeScriptで書くための設定



package.json(抜粋)



scripts

npm run dev 簡易サーバー

npm run build トランスパイル+コンパイル

npm run type-check 型チェック

devDependencies

vue-tsc: tscをラップ .vueファイルの型チェック

@types/node Node.js モジュールの型パッケージ

@vue/tsconfig Vueプロジェクトのtsconfigのベース

src フォルダの整理



main.ts 起動ファイル

App.vue エントリーポイント

router/index.ts ルーティングファイル

views/HomeView.vue ページファイル

components コンポーネントファイル

An abstract painting on the left side of the slide, featuring a dark, swirling composition of deep blues, purples, and blacks. A bright, glowing yellow and orange shape is visible in the lower-left corner, contrasting with the darker tones. The texture of the paint is visible, with some areas appearing more saturated and others more translucent.

CompositionAPI + script setup

CompositionAPI + script setup

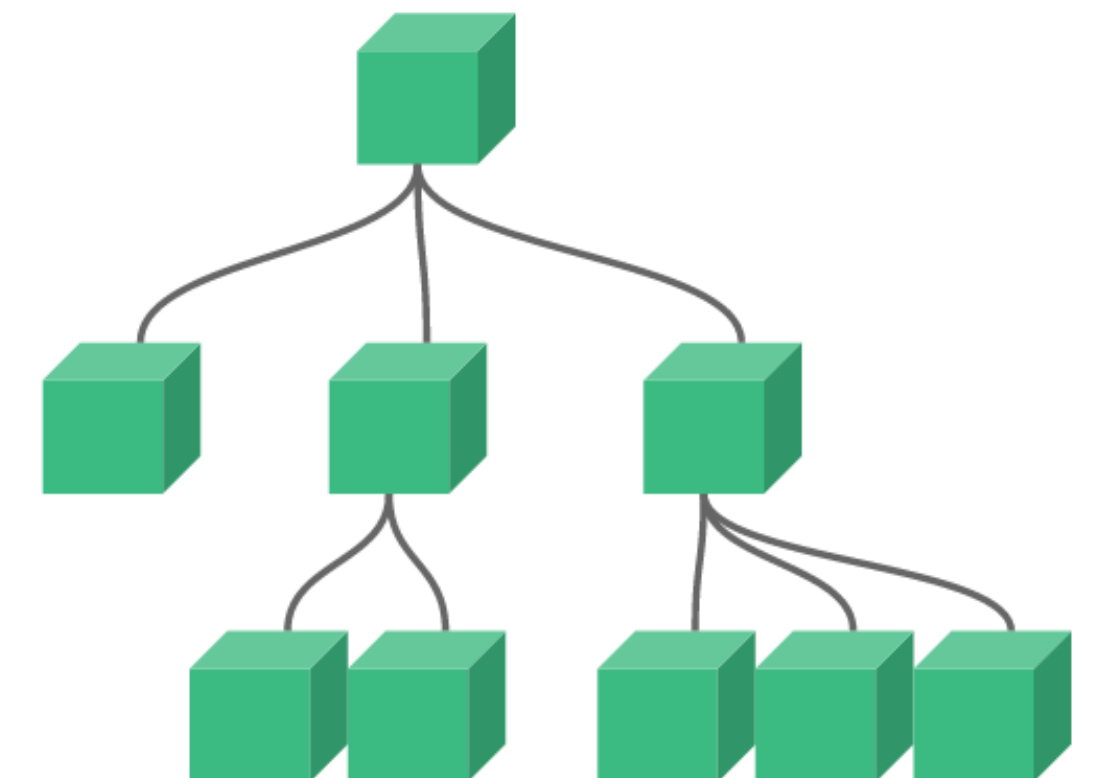
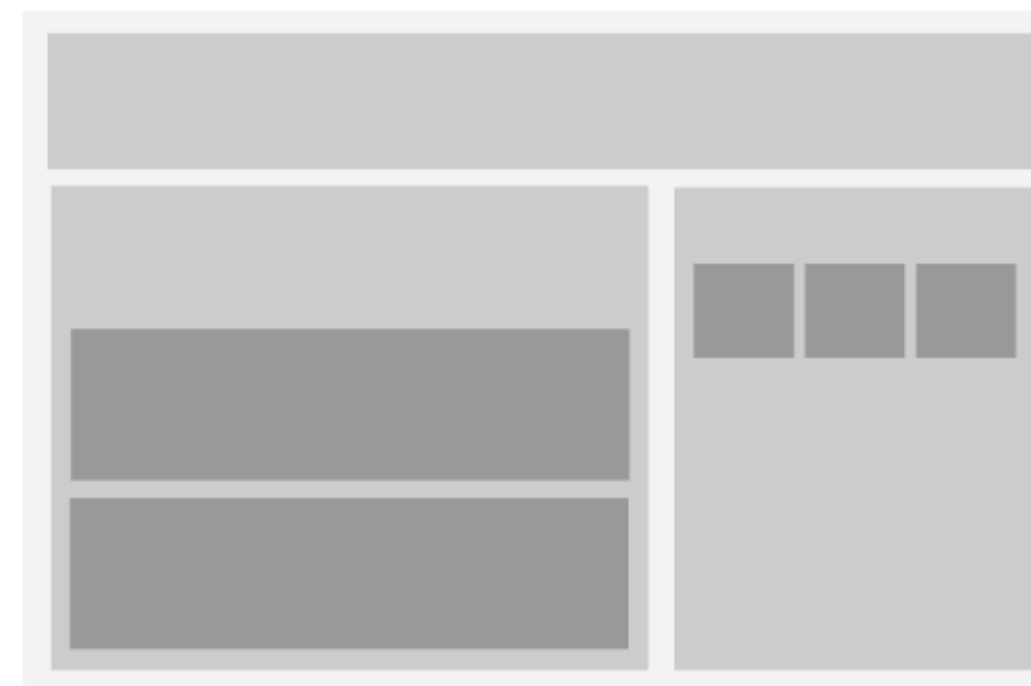
OptionAPI ・ ・ 以前から

CompositionAPI ・ ・ Vue 3.0から追加 (TypeScript対応)

script setup ・ ・ Vue ver3.2から追加

(CompositionAPIを簡潔に書ける)

`<script setup lang="ts">`



ref リアクティブな値

components/RefComponent.vue

型推論してくれるけれど

しっかり明示するなら2パターン

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

```
// ジェネリクス
```

```
const count1 = ref<number>(0)
```

```
// Ref型をimport
```

```
import type { Ref } from 'vue'
```

```
const count2: Ref<number> = ref(0)
```


reactive リアクティブオブジェクト

components/ReactiveComponent.vue

typeやinterfaceで型指定

```
<script setup lang="ts">
```

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

```
type User = { id: number, name: string}
```

```
const user1: User = reactive({ id: 1, name: '堂安'})
```

```
</script>
```

```
<template>
```

```
  <h2>reactive</h2>
```

```
  {{ user1.name }}
```

```
</template>
```


reactive その2

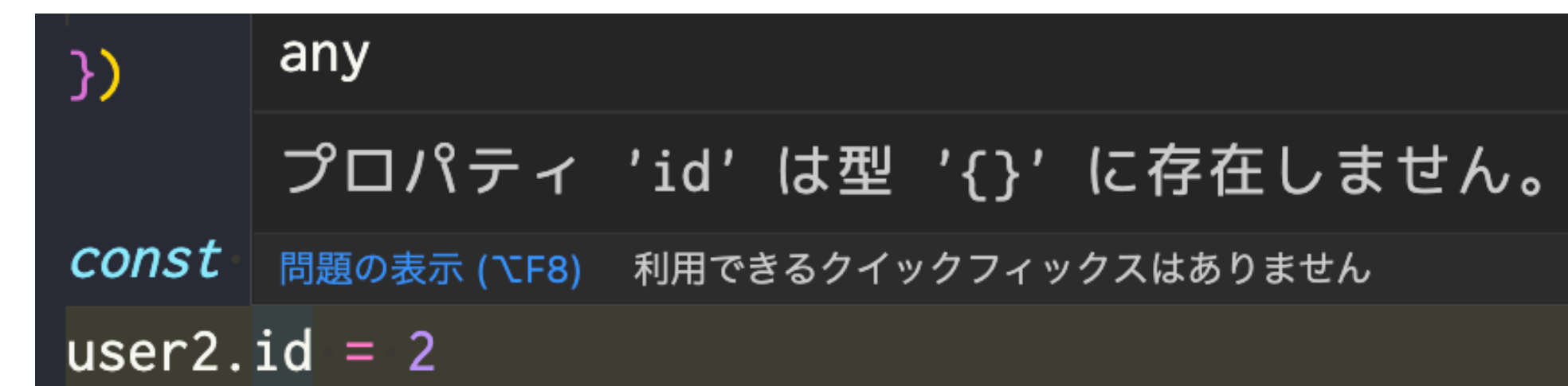
初期値 空 の場合は型アサーションを使う

```
<script setup lang="ts">
```

```
type User = { id: number, name: string}
```

```
// const user2 = reactive({})
```

```
// user2.id = 2 // これだとエラー
```



```
const user 2 = reactive({} as User)
```

```
user2.id = 2 // エラー発生しない
```


イベント その1

components/EventComponent.vue

```
<script setup lang="ts">
```

```
const handleChange = (e) => {
```

```
  console.log(e.target.value)
```

```
}
```

```
</script>
```

```
<template>
```

```
  <input @change="handleChange">
```

```
</template>
```

```
const handleChange = (e) => {  
  (parameter) e: any  
  パラメーター 'e' の型は暗黙的に 'any' になります。  
  ts(7006)
```


イベント その2

tsconfig "strict" : true なので nullチェックがかかる

(Eventの他に MouseEventやKeyboardEventなどもある)

components/EventComponent.vue

```
<script setup lang="ts">
```

```
const handleChange = (e: Event) => {
```

```
  console.log(e.target.value)
```

```
}
```

```
</script>
```

```
<template>
```

```
  <input @change="handleChange">
```

```
</template>
```

```
<script setup lang="ts">
const handleChange = (e: Event) => {
  console.log(e.target.value)
}
</script>
<template>
  <input @chan
</template>
```

(property) Event.target: EventTarget | null

Returns the object to which event is dispatched (its target).

'e.target' は 'null' の可能性があります。 ts(18047)

イベント その3

型アサーションがinstanceof

components/EventComponent.vue

```
<script setup lang="ts">
```

```
const handleChange = (e: Event) => {
```

```
  console.log((e.target as HTMLInputElement).value)
}
```

```
</script>
```

```
<template>
```

```
  <input @change="handleChange">
```

```
</template>
```


Computed 算出プロパティ

computed<number> か ComputedRef<number>

```
<script setup lang="ts">
```

```
import { ref, computed } from 'vue'
```

```
import type { ComputedRef } from 'vue'
```

```
const amount = ref<number>(0)
```

```
const subTotal : ComputedRef<number> = computed(() => amount.value * 3000)
```

```
</script>
```

```
<template>
```

```
  <h2>Computed</h2>
```

```
  単価: 3000 <br>
```

```
  商品数: <input type="number" v-model="amount"><br>
```

```
  小計: {{ subTotal }}
```

```
</template>
```


Props 親->子 に渡す

script setupの場合 defineProps() で宣言
元々propsに型宣言の記法がある typeやinterfaceでも指定できる

親 **HomeView.vue**

```
<PropsComponent :id=1 name="商品1" />
```

子 **PropsComponent.vue**

```
<script setup lang="ts">
```

```
type Props = { id: number, name: string }
```

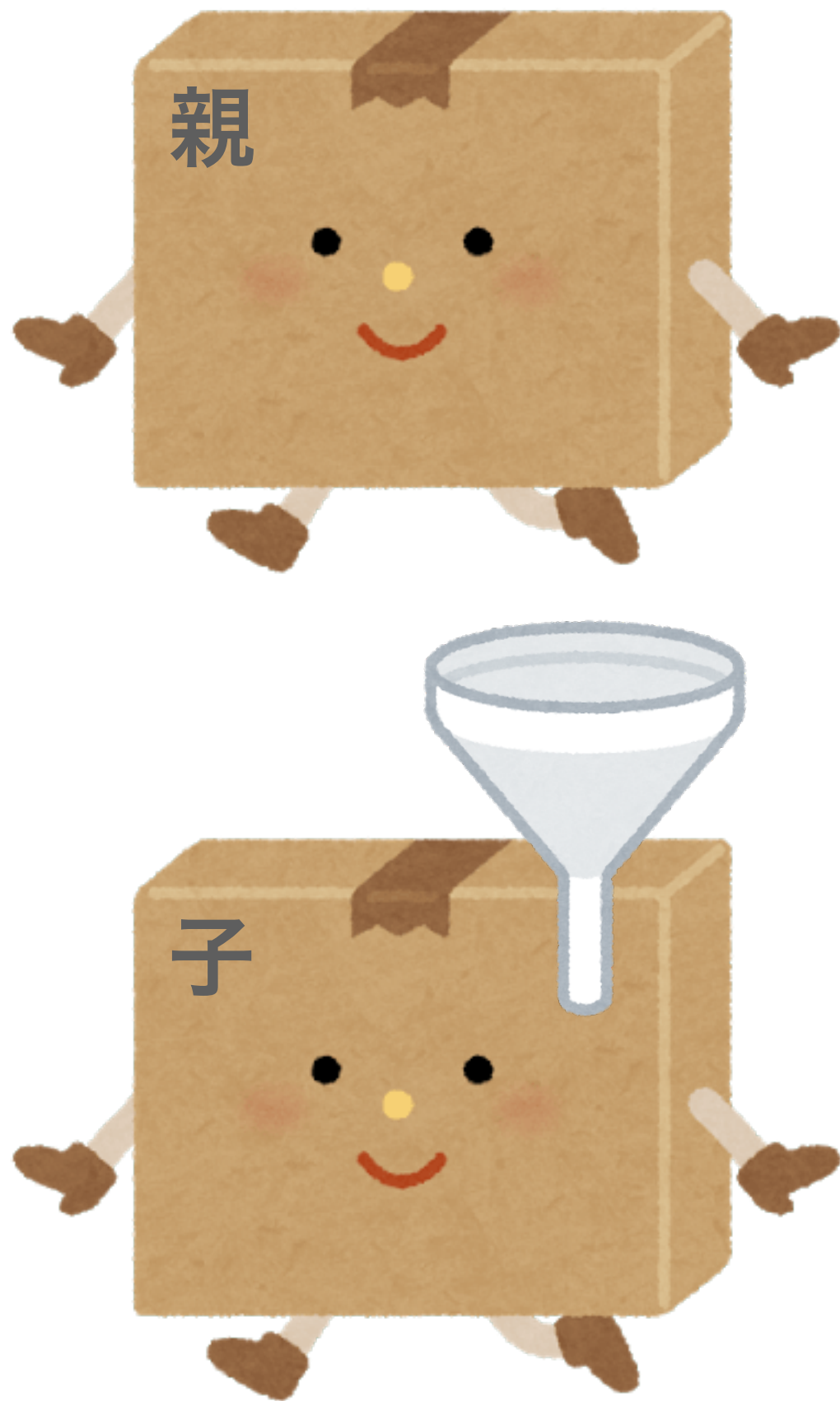
```
const props = defineProps<Props>()
```

```
</script>
```

```
<template>
```

```
  <h2>Props</h2> id: {{ props.id }}<br> name: {{ props.name }}
```

```
</template>
```



PropType

オブジェクトや配列に型をつける

親 **HomeView.vue**

```
const propTypeTest = [{ id: 1, name: '商品1'}, { id: 2, name: '商品2'}]
```

```
<PropTypeComponent :items=propTypeTest />
```

子 **PropTypeComponent.vue**

```
import type { PropType } from 'vue'
```

```
type Items = { id: number, name: string }[]
```

```
const props = defineProps({ items: Array as PropType<Items>})
```

```
<h2>PropType</h2>
```

```
<ul><li v-for="item in props.items" :key="item.id">
```

```
  id: {{ item.id }} name: {{ item.name }} </li></ul>
```


Emit 親<-子 に打ち上げる

カスタムイベント script setupの場合 defineEmits() で宣言

親 HomeView.vue

```
const emitTest = (message:string) : void => { console.log(message) }  
<EmitComponent @btn-click="emitTest" />
```

子 PropsComponent.vue

```
const emit = defineEmits<{ (e: 'btnClick', message: string) : void}>()  
const buttonClick = (message: string) : void =>  
{emit('btnClick', message)}
```

```
<button @click="buttonClick('子でクリックされた')">クリック</button>
```

