

# Fine-Grained Reactivity について ひたすら語ってみる

# 自己紹介

みゅーとん

株式会社 hacomono

Platform Group / Enabling Team / FrontEnd Tech Lead

VRChat で “フロントエンドエンジニア集会” を  
毎月開催

個人的なミッションで、アバターワークを推進



X(twitter) .. @\_mew\_ton

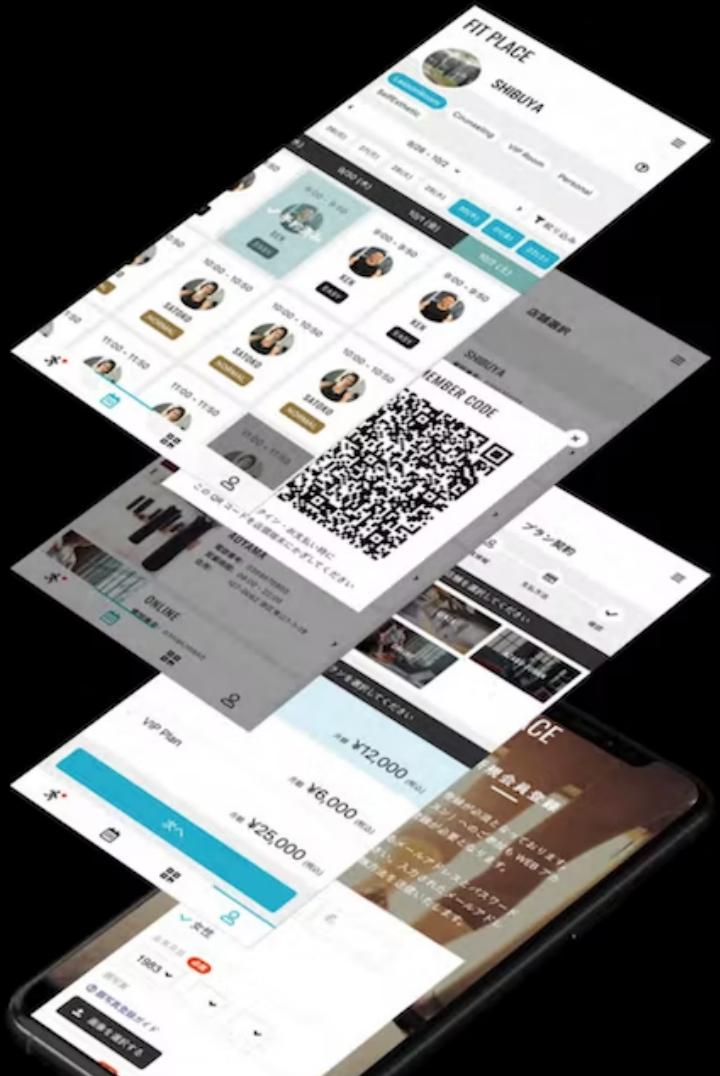
github .. @mew-ton

avatar .. ©まめひなた/もち山金魚



会員管理・予約・決済  
すべてオンライン

ウェルネス産業向けオールインワン基幹システム



# 目的

「Vue の Composition API が出てから、なんか書き方が React っぽくなったよね」

違う！

# Fine-Grained Reactivity のいみ

Fine-Grained .. きめ細かい

Reactivity .. 反応性

## Fine-Grained Reactivity を採用しているフレームワーク



signal  
computed  
effect



ref  
computed  
watchEffect



useSignal  
useMemo  
useEffect



let  
\$:  
\$state



createSignal  
createMemo  
createEffect



signal  
computed  
effect

## Fine-Grained Reactivity を採用しているフレームワーク



signal  
computed  
effect



ref  
computed  
watchEffect



useSignal  
useMemo  
useEffect



let  
\$:  
\$state



createSignal  
createMemo  
createEffect



signal  
computed  
effect

一般的に  
**Signal** と呼ばれる  
状態管理

コミュニティ

# Angular Signals

Angular Signalsは、アプリケーションのどこでどのように状態が使用されているかを細かく追跡するシステムで、フレームワークがレンダリングの更新を最適化できるようにします。

## Signal とは何か？

Signalは、値が変化したときに関心をもつ利用者に対して通知できる、値のラッパーです。

Signalの値は常にgetter関数を通して読み取られるため、AngularはSignalがどこで使用されたかを追跡できます。

Signalは、書き込み可能または読み取り専用のいずれかになります。

## 書き込み可能Signal

書き込み可能Signalは、その値を直接更新するためのAPIを提供します。書き込み可能Signalを作成するには、初期値を指定して `signal` 関数を呼び出します：

```
const count = signal(0);
```

PREACT Tutorial Guide About Blog REPL v10.19.2 GitHub Twitter 翻訳

Version: 10.x (current)

# Signals

Signals are reactive primitives for managing application state.

What makes Signals unique is that state changes automatically update components and UI in the most efficient way possible. Automatic state binding and dependency tracking allows Signals to provide excellent ergonomics and productivity while eliminating the most common state management footguns.

Signals are effective in applications of any size, with ergonomics that speed up the development of small apps, and performance characteristics that ensure apps of any size are fast by default.

**Introduction**

- Getting Started
- What's new?
- Upgrading from 8.x
- Tutorial

**Essentials**

- Components
- Hooks
- Signals**
- Forms
- References
- Context

**Debug & Test**

- Debugging Tools
- Preact Testing Library
- Unit Testing with Enzyme

**React compatibility**

- Differences to React
- Switching to Preact

- Introduction
- Installation
- Usage Example
- Building the UI
- Deriving state via computed signals
- Managing global app state
- Local state with signals
- Advanced signals usage
  - Reacting to signals outside of components
  - Reading signals without subscribing to them
  - Combining multiple updates into one
    - Rendering optimizations
- API
  - `signal(initialValue)`
  - `computed(fn)`
  - `effect(fn)`
  - `batch(fn)`

Options  Composition [?](#)

## 入門

[はじめに](#)[クイックスタート](#)

## 基本

[アプリケーションの作成](#)[テンプレート構文](#)[リアクティビティの基礎](#)[算出プロパティ](#)[クラスとスタイルのバインディング](#)[条件付きレンダリング](#)[リストレンダリング](#)[イベントハンドリング](#)[フォーム入力バインディング](#)[ライフサイクルフック](#)[ウォッチャー](#)

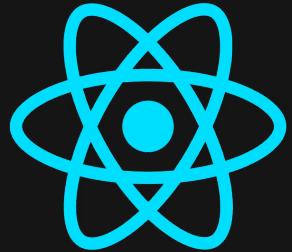
# シグナルとの関連

他のたくさんのフレームワークでも、Vue の Composition API の ref に似たリアクティビティープリミティブを「シグナル」という用語で導入しています：

- [Solid のシグナル](#)
- [Angular のシグナル](#)
- [Preact のシグナル](#)
- [Qwik のシグナル](#)

基本的に、シグナルは Vue の ref と同じ種類のリアクティビティープリミティブです。これは、アクセス時の依存関係の追跡と、変更時の副作用のトリガーを提供する値コンテナーです。このリアクティビティープリミティブベースのパラダイムは、フロントエンドの世界においては特に新しい概念ではなく、10 年以上前の [Knockout observables](#) や [Meteor Tracker](#) のような実装に遡ることができます。Vue の Options API や React の状態管理ライブラリーである [MobX](#) も同じ原理に基づいていますが、オブジェクトプロパティの裏側にあるプリミティブを隠しています。

# React は・・?



✗

useState  
useMemo  
useEffect

○

jotai-signal  
legend-state

# 改めて, Fine-Grained Reactivity とは?

- プリミティブな値を管理する変数
  - アクセス時に依存関係を保持
  - 値の変更時に依存先の変更をトリガーする
- ・・・ ちょっとよくわからんな

コードで理解してみる

# Vue (Options API の場合)

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

export default defineComponent({
  data() {
    // firstName, lastName の状態を持たせておく
    return {
      firstName: 'John',
      lastName: 'Smith'
    }
  },
  computed: {
    fullName(): string {
      return this.firstName + ' ' + this.lastName
    }
  },
  methods: {
    update() {
      this.firstName = 'Jacob'
    }
  }
})
</script>

<template>
  <!-- update 後自動反映される -->
  <p>{{ fullName }}</p>
</template>
```

## Vue (Composition API) の場合

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

// firstName, lastName の getter, setter ができる
const firstName = ref('John')
const lastName = ref('Smith')

const fullName = computed(() => `${firstName.value} ${lastName.value}`)

function update() {
  firstName.value = 'Jacob'
}
</script>

<template>
  <!-- update 後自動反映される -->
  <p> {{ fullName }} </p>
</template>
```

## SolidJS の場合

```
import { createSignal, createMemo } from 'solid'

const FullName = () => {
    const [firstName, setFirstName] = createSignal('John')
    const [lastName, setLastName] = createSignal('Smith')

    const fullName = createMemo(() => `${firstName()} ${lastName()}`)

    // どこから呼ばれる想定
    function update() {
        setFirstName('Jacob')
    }

    return (
        <p>
            {/* update 後、関数が再評価されず、fullName のみ更新される */}
            {fullName()}
        </p>
    )
}
```

# React (Hooks API) の場合

```
import { useState, useMemo } from 'react'

const FullName = () => {
  const [firstName, setFirstName] = useState('John')
  const [lastName, setLastName] = useState('Smith')

  // 第2引数に依存を設定
  // 関数再実行時に firstName, lastName が変化していれば再評価する
  const fullName = useMemo(() => `${firstName()} ${lastName()}`, [firstName, lastName])

  // どこからよばれる想定
  function update() {
    // Component の関数を再実行する
    setFirstName('Jacob')
  }

  return (
    <p>
      {fullName()}
    </p>
  )
}
```

結構似てる...?

# 描画処理の違い

## Fine-Grained Reactivity

※ 全てのフレームワーク共通ではないので注意

コンポーネントの初期化は描画前に1回のみ

DOM の構成内からその値に依存している場合,  
副作用としてその部分のみを更新する

## React Hooks API

Functional Component

描画, 再描画のたびに関数が実行される

useState したコンポーネントを覚えており  
setter を呼ぶと, そのコンポーネントから再描画される

# 書くモチベーションの違い

## Fine-Grained Reactivity

※ 全てのフレームワークではないので注意

どの値を再描画の対象にするか を意識する

再描画させたい .. signal で値を管理する

再描画させたくない .. const にする

依存が迷路にならないように,  
大量にならないように書いていく

## React Hooks API

どの値を再描画させない対象にするか を意識する

依存を明示しない場合,  
副作用は再描画のたびに実行される

依存を明示して、副作用を抑えるように書いていく

## まとめ

- Fine-Grained Reactivity は最近のフロントエンドではトレンドの状態管理
- Vue だと Options API から定着してるやり方
- React とは明確に違う部分

おわり