

Vue3 最新構文 <script setup> の紹介

FUTURE CON 2022 - Jul 18th

Press Space for next page →



自己紹介



太田 洋介

- 年齢: アラフォー 📍 : 神奈川県
- GitHub: [@ota-meshi](#) ♡ Sponsor, npm: [ota-meshi](#), Twitter: [@omoteota](#), Qiita: [@ota-meshi](#)
- 所属:
 - [フューチャー株式会社](#) 社員 (2015/06 -)
 - [Vue.js](#) eslint-plugin-vue メンテナー (eslint-plugin-vue 2018/08 - , Vue 2019/07 -)
 - [Stylelint](#) Owners チーム (2020/09 -)
 - [Intlify](#) eslint-plugin-vue-i18n 担当 (2020/07 -)
 - [Stylus](#) チーム (2022/06 -)
- [WEB+DB PRESS Vol.120](#) 「最新Vue.js 3入門」 共同執筆 (2020/12/24)
- [Google Open Source Peer Bonus 2022](#) 受賞

アジェンダ

1. [What is <script setup>?](#)
2. [<script setup>の書き方](#)
 1. [基本](#)
 2. [リアクティブ・コンポーネント](#)
 3. [カスタムディレクティブ・非同期](#)
 4. [Prop・カスタムイベント](#)
 5. [Prop・カスタムイベント with TypeScript](#)
 6. [その他](#)
3. [Reactivity Transform](#)
 1. [What is Reactivity Transform?](#)
4. [まとめ](#)

What is <script setup>? 🤔

What is <script setup>?

- Vue3.2 で正式に導入された SFC 用の新構文
7/1にリリースされた Vue2.7 にも導入されました。
- 冗長な記述を少なくしよりシンプルに記述できる
- 実行時のパフォーマンスの向上
(他の記述方法からのコンパイルでは必要だった中間処理をバイパス)
- TypeScript との親和性向上
(型情報によるコンポーネント定義の生成。
IDE のサポート)

```
<script setup>
import { ref } from "vue";
import MyComponent from "../MyComponent.vue";

const count = ref(0);

function handleClick() {
  count.value++;
}
</script>

<template>
  <button @click="handleClick">
    {{ count }}
  </button>

  <MyComponent />
</template>
```

<script setup>の書き方

<script setup>の書き方（基本）

- <script setup> と記述します
- トップレベルの変数はテンプレートで直接使用できます

```
<script setup>
const msg = "Hello!";

function handleClick() {
  alert(msg);
}
</script>

<template>
  <button
    @click="handleClick">
    {{ msg }}
  </button>
</template>
```

[DEMO](#)

<script setup>の書き方（基本）

別の方法ではこのように書いていました。

Composition API

```
<script>
export default {
  setup() {
    const msg = "Hello!";
    function handleClick() {
      alert(msg);
    }
    return {
      msg,
      handleClick,
    };
  },
};
</script>

<template>
  <button @click="handleClick">
    {{ msg }}
  </button>
</template>
```

Options API

```
<script>
export default {
  data() {
    return {
      msg: "Hello!",
    };
  },
  methods: {
    handleClick() {
      alert(this.msg);
    },
  },
};
</script>

<template>
  <button @click="handleClick">
    {{ msg }}
  </button>
</template>
```


<script setup>の書き方（リアクティブ・コンポーネント）

- リアクティブな状態を管理するには[Reactivity API](#)を使用します
- コンポーネントは`import`するだけで使用できます
(トップレベルの変数と同じ扱いです)

```
<script setup>
import { ref } from "vue";
import MyButton from "./MyButton.vue";

const count = ref(0);

function handleClick() {
  count.value++;
}
</script>

<template>
  <MyButton @click="handleClick">
    {{ count }}
  </MyButton>
</template>
```

[DEMO](#)

<script setup>の書き方（リアクティブ・コンポーネント）

別の方法ではこのように書いていました。

Composition API

```
<script>
import { ref } from "vue";
import MyButton from "./MyButton.vue";
export default {
  components: {
    MyButton,
  },
  setup() {
    const count = ref(0);
    function handleClick() {
      count.value++;
    }
    return {
      count,
      handleClick,
    };
  },
};
</script>

<template>
  <MyButton @click="handleClick">
    {{ count }}
  </MyButton>
</template>
```

Options API

```
<script>
import MyButton from "./MyButton.vue";
export default {
  components: {
    MyButton,
  },
  data() {
    return {
      count: 0,
    };
  },
  methods: {
    handleClick() {
      this.count++;
    },
  },
};
</script>

<template>

  <MyButton @click="handleClick">
    {{ count }}
  </MyButton>
</template>
```

<script setup>の書き方（カスタムディレクティブ・非同期）

- カスタムディレクティブは`v`で始まる変数を使用します
- トップレベル`await`も使用できます
(Vue2では使用できません)

```
<script setup>
import vMyDirective from './my-directive.js';
const post = await fetch(`/api/post/1`).then((r) => r.json)
</script>

<template>
  <p v-my-directive>
    {{ post }}
  </p>
</template>
```

<script setup>の書き方（カスタムディレクティブ・非同期）

別の方法ではこのように書いていました。

Composition API

```
<script>
import vMyDirective from "./my-directive.js";
export default {
  directives: {
    "my-directive": vMyDirective,
  },
  async setup() {
    const post = await fetch(`/api/post/1`).then((r) => r.json());
    return {
      post,
    };
  },
};
</script>

<template>
  <p v-my-directive>
    {{ post }}
  </p>
</template>
```

Options API

```
<script>
import vMyDirective from "./my-directive.js";
export default {
  directives: {
    "my-directive": vMyDirective,
  },
  data() {
    return {
      post: "", // 非同期はスマートな書き方ができない
    };
  },
  async mounted() {
    this.post = await fetch(`/api/post/1`).then((r) => r.json());
  },
};
</script>

<template>
  <p v-my-directive>
    {{ post }}
  </p>
</template>
```

<script setup>の書き方 (Prop・カスタムイベント)

- Props を定義・使用するには `defineProps` というコンパイラマクロを使用します
- カスタムイベントを定義・使用するには `defineEmits` というコンパイラマクロを使用します

```
<script setup>
import { ref } from "vue";
const props = defineProps({
  modelValue: String,
});
const emit = defineEmits(["update:modelValue", "send"]);
const inputRef = ref();

function handleInput() {
  emit("update:modelValue", inputRef.value.value);
}
function handleClick() {
  emit("send", props.modelValue);
}
</script>

<template>
  <input ref="inputRef" :value="modelValue" @input="handle
  <button @click="handleClick">send</button>
</template>
```

[DEMO](#)

<script setup>の書き方 (Prop・カスタムイベント)

別の方法ではこのように書いていました。

Composition API

```
<script>
import { ref } from "vue";

export default {
  props: {
    modelValue: String,
  },
  emits: ["update:modelValue", "send"],
  setup(props, { emit }) {
    const inputRef = ref();

    function handleInput() {
      emit("update:modelValue", inputRef.value.value);
    }
    function handleClick() {
      emit("send", props.modelValue);
    }
    return {
      inputRef,
      handleInput,

      handleClick,
    };
  },
};
</script>

<template>
  <input ref="inputRef" :value="modelValue" @input="handle
  <button @click="handleClick">send</button>
```

Options API

```
<script>
export default {
  props: {
    modelValue: String,
  },
  emits: ["update:modelValue", "send"],
  methods: {
    handleInput() {
      this.$emit("update:modelValue", this.$refs.inputRef.
    },
    handleClick() {
      this.$emit("send", this.modelValue);
    },
  },
};
</script>

<template>
  <input ref="inputRef" :value="modelValue" @input="handle

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

<script setup>の書き方 (with TypeScript)

- `defineProps``と`defineEmits``はTypeScriptを使用する場合、型のみの定義を使用できます
 - `defineProps``ではPropの型を定義します
 - デフォルト値には`withDefaults``を併用します
 - `defineEmits``では関数の型を定義します

```
<script setup lang="ts">
// ...
const props = withDefaults(
  defineProps<{
    modelValue: string;
    message?: "Hello!" | "Goodby!";
 }>(),
  {
    message: "Hello!",
  }
);
const emit = defineEmits<{
  (e: "update:modelValue", newValue: string): void;
  (e: "send", sendValue: string): void;
}>();
// ...
</script>
```

[DEMO](#)

<script setup>の書き方 (with TypeScript)

型のみ定義方法を使用しない場合はこのように書いていました。

```
<script setup lang="ts">
// ...
const props = defineProps({
  modelValue: {
    type: String,
    required: true,
  },
  message: {
    type: String, // ユニオン型にはできません。
    default: "Hello!",
  },
});
const emit = defineEmits({
  "update:modelValue": (newValue: string) => true,
  send: (sendValue: string) => true,
});
// ...
</script>
```


<script setup>の書き方（その他）

- `defineExpose` を使用して外部から参照できるインスタンスプロパティを定義します
- `useSlots` と `useAttrs` を使用して `$slots` や `$attrs` のような情報にアクセスできます
- `<script setup>` とは別に `<script>` ブロックを定義できます
 - `export default {}` を定義することで、`inheritAttrs` などのコンポーネントオプションを定義できます
 - `.vue` が `export` する情報を定義できます
 - `module` が呼び出されたときに一度だけ実行されるスクリプトを書くことができます

```
<script setup>
import { useSlots, useAttrs } from "vue";

const slots = useSlots();
const attrs = useAttrs();
// ...
</script>
```

```
<script>
// module が呼び出されたときに一度だけ実行される
runSideEffectOnce();

// オプションの定義
export default {
  inheritAttrs: false,
};
</script>

<script setup>

// ...
</script>
```

Reactivity Transform

What is Reactivity Transform?

- Vue3.3 で正式に導入**予定**の新構文
(Vue3.2 でも実験的機能として使用可能)
- リアクティブ API 周りの冗長な記述を少なくし
よりシンプルに記述できる
- SFC 以外 (JavaScript・TypeScript) でも使用で
きます
(Vite を使用している場合)

```
<script setup>
import MyComponent from "./MyComponent.vue";

const count = $ref(0);

function handleClick() {
  count++;
}
</script>

<template>
  <button @click="handleClick">
    {{ count }}
  </button>

  <MyComponent />
</template>
```

What is Reactivity Transform?

Ref オブジェクトの`.value`を消し去れます。

以前:

```
<script setup>
import { ref, toRefs, computed } from "vue";
import { useMouse } from "@vueuse/core";

const count = ref(0);

function handleClick() {
  count.value++;
}

const { x, y } = toRefs(useMouse());

const mousePosition = computed(() => `x: ${x.value}, y: ${y.value}`);
</script>
```

Reactivity Transform 使用:

```
<script setup>
import { useMouse } from "@vueuse/core";

let count = $ref(0);

function handleClick() {
  count++;
}

let { x, y } = $(useMouse());

const mousePosition = $computed(() => `x: ${x}, y: ${y}`);
</script>
```

[DEMO](#)

What is Reactivity Transform?

詳細は [Reactivity Transform](https://vuejs.org/guide/extras/reactivity-transform.html) のドキュメントを参照してください。

<https://vuejs.org/guide/extras/reactivity-transform.html>

現在、実験的機能であるため、このスライドでは詳細は割愛します。
また、このスライドに載せた記述方法や機能は変更になる可能性があります。

まとめ


まとめ

- Vue3.2 で <script setup> が導入されました
(Vue2.7 にも導入されました)
- Vue3.3 では Reactivity Transform が導入される予定です

冗長だった記述がかなり少なくなり
シンプルに記述できます。

Vue3.3 が楽しみですね 🌟😊

Thank you for your attention

Support me  or follow me!!

GitHub: <https://github.com/ota-meshi>

Twitter: <https://twitter.com/omoteota>

Qiita: <https://qiita.com/ota-meshi>