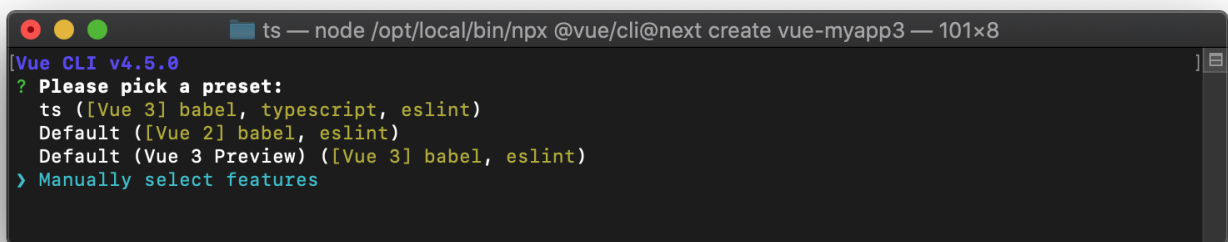


Vue.jsの環境構築

Vue.jsは日本で人気のあるウェブフロントエンドのフレームワークです。柔軟な設定のできるCLIツールが特徴です。本書では3系についてとりあげます。

```
$ npx @vue/cli@next create myapp
```

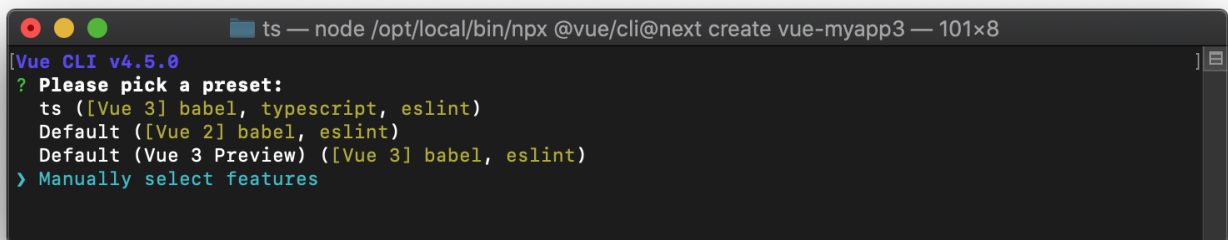
作成時に最初に聞かれる質問でdefaultのpreset（babelとeslint）ではなく、Manually select featuresを選択します。



```
[Vue CLI v4.5.0]
? Please pick a preset:
  ts ([Vue 3] babel, typescript, eslint)
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
> Manually select features
```

TypeScriptを選択する場合はManually select featuresを選択

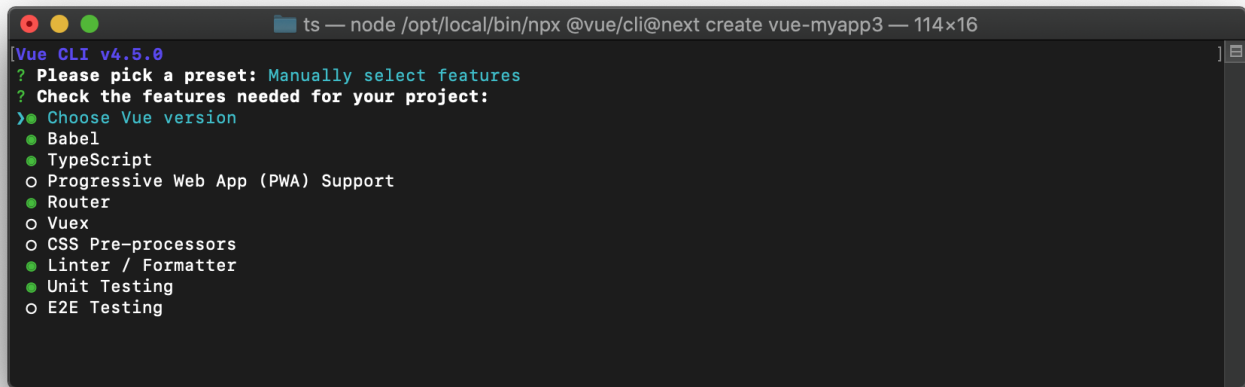
次のオプションで必要なものをスペースキーで選択して、エンターで次に進みます。選んだ項目によって追加の質問が行われます。Routerやステート管理などのアプリケーション側の機能に関する項目以外にも、LinterやユニットテストフレームワークやE2Eテストの補助ツールなど、さまざまなものを選択できます。



```
[Vue CLI v4.5.0]
? Please pick a preset:
  ts ([Vue 3] babel, typescript, eslint)
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
> Manually select features
```

TypeScriptを選択する場合はManually select featuresを選択

途中で、クラスベースかそうではないか、という質問が出てきます。以前ではクラスベースのAPIの方がTypeScriptとの相性がよかったのですが、Vue.js3からの新しいAPIで、クラスベースでない時も型チェックなどに優しいComposition APIが追加されました。そこはチームで好きな方を選択すれば良いですし、あとから別のスタイルにすることもできます。



必要な機能を選択する

現在のVue.jsのプロジェクトのほとんどは、`.vue` ファイルに記述するシングルファイルコンポーネント（SFC）を使っていると思いますが、TypeScriptを使う場合、スクリプトタグの `lang` 属性を `ts` になっています。

```
<template>
  HTML テンプレート
</template>
<script lang="ts">
  コンポーネントのソースコード (TypeScript)
</script>
<style>
  CSS
</style>
```

クラスベースのコンポーネント

クラスベースのコンポーネントは `vue-class-component` パッケージを使い、デコレータをつけたクラスとして実装します。クラスのインスタンスのフィールドがデータ、ゲッターが算出プロパティになっているなど、TypeScriptのプレーンな文法とVueの機能がリンクしており、ウェブフロントエンドを最初に学んだのではなく、言語としてのTypeScriptやJavaScript、他の言語の経験が豊富な人には親しみやすいでしょう。環境構築のCLIのオプションではデフォルトでこちらになるような選択肢になっています。

以下のコードはVue.js 3系のクラスベースのコンポーネント実装です。

```
<script lang="ts">
import { Options, Vue } from "vue-class-component";
import HelloWorld from "../components/HelloWorld.vue";

@Options({
  components: { // テンプレート中で利用したい外部のコンポーネント
    HelloWorld
  }
})
export default class Counter extends Vue {
  // フィールドがデータ
  count = 0;

  // 算出プロパティはゲッターとして実装
  get message() {
    return `カウントは${this.count}です`;
  }

  // メソッドを作成するとテンプレートから呼び出せる
  increment() {
    this.count++;
  }

  decrement() {
    this.count--;
  }

  // ライフサイクルメソッドもメソッド定義でOK
  beforeMount() {
  }
}
</script>
```

これをラップしてより多くのデコレータを追加したvue-property-decoratorというパッケージもあります。こちらの方が、`@Prop` や `@Emit` でプロパティやイベント送信も宣言できて便利でしょう。

- <https://www.npmjs.com/package/vue-property-decorator>

警告

ただし、現時点で3.0系で変わったvue-class-componentの変更にはまだ追従していないように見えます。

関数ベースのコンポーネント作成

Vue本体で提供されている `defineComponent()` 関数を使いコンポーネントを定義します。今までのオブジェクトをそのまま公開する方法と違い、この関数の引数のオブジェクトの型は定まっているため、以前よりもTypeScriptとの相性が改善されています。このオブジェクトの属性で名前や他の依存コンポーネント、Propsなどを定義するとともに、`setup()` メソッドでコンポーネント内部で利用される属性などを定義します。

```
<script lang="ts">
import { defineComponent, SetupContext, reactive } from "vue";
import HelloWorld from "../components/HelloWorld.vue";

type Props = {
  name: string;
}

export default defineComponent({
  name: "App",
  components: {
    HelloWorld
  },
  props: {
    name: {
      type: String,
      default: "hello world"
    }
  },
  setup(props: Props, context: SetupContext) {
    const state = reactive({
      counter: 0,
    });
    const greeting = () => {
      context.emit("greeting", `Hello ${props.name}`);
    };

    return {
      state,
      greeting
    }
  }
});
</script>
```

注釈

Nuxt.jsを使ったプロジェクト作成

Vue.jsにも、Vue.jsをベースにしてサーバーサイドレンダリングなどの自分で設定すると大変な機能がプリセットになっているNuxt.jsがあります。Nuxt.jsの場合は、通常の設定の後に、いくつか追加のパッケージのインストールや設定が必要です。日本語によるガイドもあります。

- <https://typescript.nuxtjs.org/ja/guide/setup.html>

ただし、現時点ではVue.js 3対応はまだ計画中でリリースはまだ行われていません。

Vue.jsを使ったjQueryのリブレース

jQueryは歴史があるライブラリで、使い勝手の良さから、非フロントエンド開発者にも広く普及しました。一方で、開発が大規模化する場合に整合性をとるのが難しくなってくることが多く、フロントエンドの比重が高まるにつれて、ReactやVue.jsを使う人が増えています。

jQueryからVue.jsへはパラダイムがかなり違うので、多少コーディングが必要となります。jQueryは、セレクトでマッチしたHTMLのタグを直接変更していきます。一方、最近のウェブフロントエンドのフレームワークはTypeScript内部に状態を持ち、それを画面に反映させる、という形をとります。反映するときはテンプレートエンジンのような記法を用いて表現します。VueやReactは仮想DOMという仕組みを使っており、ビュー関数を頻繁に実行し、その結果を画面に反映します。

jQueryの方が、極めて簡単なことをする場合は短いコードで済むことがあります。一方、変更が多くなると更新が複雑になります。

- 同じ値を何度も表示する場合、VueやReactの場合、大元の変数を変更するとすべての箇所が変わります。jQueryでは利用箇所をすべて自分で見つけて更新しなければなりません。
- テーブル表示など、表示先の階層が深くて場所の特定も大変な場合にはロジックが複雑になります。
- 確認ダイアログを出してOK/Cancel時に別のダイアログを出してという場合には、次のダイアログを表示にする、前のダイアログを非表示にする、といったように、すべての変更を1つずつ適用していかなければなりません。状態遷移が複雑になってくると、一箇所の修正漏れで画面の遷移がおかしくなります。VueやReactであれば、現在の遷移はどこか、という情報を1つもち、それをみるようにすると、扱う状態が少ない分、ミスが減ります。

次のようなシンプルなjQueryのコードをVue.jsにしてみましょう。

```
<div>jQuery test page</div>
<button class="pushbutton">button</button>
<div class="panel" style="display: none; background-color: lightblue;"
  hidden
</div>

<style scoped>
  .clickedButton {
    background-color: "yellow";
  }
</style>

<script>
$(function() {
  $(".pushbutton").click(function() {
    $(".pushbutton").addClass("clickedButton");
    $(".panel").fadeIn();
  });
});
</script>
```

ここで使っているjQueryの機能は3つです。

- `click()` でボタンのクリックのイベントハンドラの設定
- クリック時に `addClass()` でCSSのスタイルの変更
- クリック時に、`fadeIn()` を使って隠された要素の表示

まずはそのまま.vueファイル化

まずはプロジェクトを作り、既存のHTMLファイルを取り込みます。この説明では次の想定で進めます。

- プロジェクトはvue-cliで作成
- 対象バージョンはVue.js 3で、クラス形式のコンポーネントではない関数ベースのコンポーネントを利用
- ルーターを利用

jQueryと比較した場合の、Vue.jsの大きく違うポイントは次の通りです。

- Vueは特定のノード以下のみをプログラムで変更可能にする
- 1つのHTMLを元にページ切り替えを実現できる（シングルページアプリケーション）

元のjQueryでシングルページアプリケーションを実現しているケースはほとんどないと思うので、いったん、元のプロジェクトは複数ページから構成されているものとして話を進めます。

まず、複数のページであっても、1つのHTMLとTypeScriptコードで実現します。今まで共通のヘッダーなどを個別に実装していた場合はベースのHTML側に書いておけば共通で利用されます。複数のページで違いの発生する部分のみを.vueファイルにします。

まずはjQueryをライブラリに追加します。

```
$ npm install jquery @types/jquery
```

次に、.vueファイルを作成します。これはHTML、CSS、スクリプトが1ファイルにまとまった、シングルファイルコンポーネントと呼ばれるものです。 `<template>` にHTMLを書き、 `<script lang="ts">` にはTypeScriptのコードを書きます。

DOM読み込み後に呼ばれるイベントハンドラでjQueryのイベント定義などを行っていました。jQueryの読み込み後のハンドラーは次のどちらかを設定していました。

- `$(document).ready(function () {})`
- `$(function() { })`

Vue.jsでは、ページごとにコンポーネントと呼ばれるオブジェクトを作成します。それが作成されるタイミングで `setup()` メソッドが呼ばれますが、その中で `onMounted()` 関数に登録したコードが実行時に呼ばれます。まずはそこにjQueryのコードを移植してしまいましょう。完成形は次の通りです。

/src/views/mypage.vue

```
<template>
  <div>jQuery test page</div>
  <button class="pushbutton">button</button>
  <div class="panel" style="display: none; background-color: lightblue;"
    hidden
  </div>
</template>

<style scoped>
  .clickedButton {
    background-color: "yellow";
  }
</style>

<script lang="ts">
import $ from "jquery";

import { defineComponent, onMounted } from "vue";

export default defineComponent({
  name: "MyPage",
  setup() {
    onMounted(() => {
      $(".pushbutton").click(() => {
        $(".pushbutton").css("background-color", "yellow");
        $(".panel").fadeIn();
      });
    });
  }
});
</script>
```

なお、この `onMounted` は作成時に一度だけ呼ばれます。その後、属性の変更などがあると、タグの再作成プロセスが走ります。差分のみの更新のはずですが、元の変更によってはイベントハンドラをつけたタグが再作成されたりすることもあり、イベントハンドラがなくなる可能性もあります。このやり方はあくまでも移行のための暫定なので、本番環境にデプロイなどをしてはいけません。

ページを作ったら、そのファイルが表示可能になるように、ルーターに登録します。そのページのURLと、その時に使われるコンポーネントのペアを関連づけるものです。これで表示できるようになりました。

/src/router/router.ts

```
import { createRouter, createWebHistory, RouteRecordRaw } from "vue-router";
import MyPage from "../views/mypage.vue";

const routes: Array<RouteRecordRaw> = [
  {
    path: "/",
    name: "MyPage",
    component: MyPage
  },
];

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
});

export default router;
```

まずはここまでで動作することを確認しましょう。

クリックイベントの定義

Vueの作法でクリックイベントを設定しましょう。jQueryはHTMLの外でタグとイベントの関連付けを `$.click()` を使って行っていました。Vue.jsの場合は、テンプレートの中で関数を設定します。呼び出す関数は `setup` の中で作成します。テンプレートで `clicked` という名前呼びたい場合には次のように作成します。

/src/views/mypage.vue

```
import { defineComponent } from "vue";

export default defineComponent({
  name: "MyPage",
  setup() {
    // イベントハンドラを作成
    const clicked = () => {
      $(".pushbutton").css("background-color", "yellow");
      $(".panel").fadeIn();
    }
    // テンプレートで使う要素はsetupのレスポンスのオブジェクトとして返す
    return {
      clicked
    }
  }
});
```

HTMLは次のようになります。素のHTMLのイベントハンドラの方に近くなります。ここまで動作することを確認しましょう。

/src/views/mypage.vue


```
<button class="pushbutton" @click="clicked">button</button>
```

jQueryとの大きな違いとしては、変更対象のHTMLタグの発見方法があります。jQueryはすでにあるHTMLのタグの中から変更対象を見つける必要があります。そのため、セレクトーという機能に磨きをかけて、変更対象をすばやく確実に見つける機能を備えました。一方のVue.jsは、生成時に差し込みます。そのため、タグを探してくる「セレクトー」という概念がありません。

クリックによるインタラクションをVue.js化

Vueでは一元管理されたデータを元にビューを更新するとすでに説明しました。その本丸に攻め入ります。その状態管理で利用するのが `reactive` 関数です。引数にオブジェクトを渡すと、それがコンポーネントのデータ源として利用できます。これもテンプレートから利用するため、`setup()` の返り値で返します。

今回のサンプルは小さいので問題ありませんが、実用的なコンポーネントの場合、この `reactive()` の各要素にコメントを書いておいても良いでしょう。

`clicked()` の中は、単にこの状態を更新するだけになりました。

`/src/views/mypage.vue`

```
import { defineComponent, reactive } from "vue";

export default defineComponent({
  name: "jQuery",
  setup() {
    const state = reactive({
      show: false // パネルの表示制御に使う
    });

    const clicked = () => {
      state.show = true;
    };
    return {
      clicked,
      state
    };
  }
});
```

テンプレート側はつぎのようになりました。この状態をみて、ボタンの方はクラスのON/OFFの切り替えを行い、パネルの方はCSSのスタイルの内容の変更を行っています。

`:class` は、オリジナルのVueのルールでは `v-bind:class` でしたが、省略記法として `:class` も提供されています。実は、`@click` も `v-on:click` の省略形になります。

/src/views/mypage.vue

```
<button
  @click="clicked"
  :class="{ clickedButton: state.show }"
>
  button
</button>
<div class="panel" :style="{visibility: state.show ? 'visible' : 'hidden', backgroundColor:
'lightblue'}">
  hidden
</div>
```

単に表示のON/OFFを切り替えるだけであれば、`v-if` や `v-show` といったディレクティブもあります¹。

¹ <https://v1-jp.vuejs.org/guide/conditional.html>

フェードイン効果を実現する

クリックしたタイミングでスタイルシートや表示のON/OFFの切り替え方法は学びました。本節のラストとして、フェードインを実現します。

jQueryのフェードインはふわっと表示させるときに利用します。デフォルト値は0.4秒での表示になります。また、透明度の変化のカーブの選択もできます。このあたりの複雑なパラメータの時間変化は、最新のCSSを使えば造作ありません。まずはフェードインのCSSクラスです。`opacity` と `visibility` が非表示状態で開始します。その後、クリックで追加のクラス設定があると最終的には完全表示になりますが、その途中経過で0.4秒かけて表示するための `transition` が含まれています。

/src/views/mypage.vue

```
.hiddenPanel {
  opacity: 0;
  visibility: hidden;
}

.hiddenPanel.fadeIn {
  opacity: 1;
  visibility: visible;
  transition: opacity 0.4s, visibility 0.4s;
}
```

すでに実装されているロジックで問題ありません。残る修正箇所はテンプレートのみです。

`/src/views/mypage.vue`

```
<div
  class="panel" :class="{hiddenPanel: true, fadeIn: state.show}"
  :style="{ backgroundColor: 'lightblue' }">
  hidden
</div>
```

これで、クリックされた瞬間に `fadeIn` クラスが付与されるようになりました。このCSSクラスは0.4秒かけて切り替えると実装されているため、`jQuery` の `fadeIn` 同等の処理が実現できました。

Vue.jsの方が行数が長くなる？

jQueryの移植の基本について説明してきました。イベントのハンドリングとスタイルの切り替えはこれで問題がなくなったでしょう。

これらの処理はjQueryがもっとも得意とするものであり、Vue.jsの方がかなり行数が増えてしまっています。ここだけみるとVue.jsのメリットが少なく感じます。

しかし、Ajax的なロジックの実装を開始しはじめると複雑度は逆転します。サーバー通信して、取得してきたJSONを元にリストを表示するようなケースでは、jQueryの場合はプログラムの中でHTMLのDOM要素を作って挿入という処理になります。適切なHTML片を作って挿入するのは大変ですし、雑に実装するとXSSというセキュリティの穴があく恐れもでてくるため、jQueryでも `mustache` や `underscore.js` の `template`、`handlebars` といったテンプレートエンジンの導入を検討し始めるところです。ロジックと表示の元データと表示のテンプレートが分離しにくいといった問題も出てきますし、コードの行数も爆発し始めます。表示以外に、編集や削除をしたいといった要件が出てきたらもうお手上げでしょう。

Vue.jsの場合はスクリプトはデータの保持のみになりますし、テンプレートによる結果のレンダリングは最初から行っていますので、複雑度は変わりません。`v-for` ディレクティブでリストの表示も簡単です。

それ以外にjQueryの方が短く、コードが長くなってしまいうケースとしては、jQueryプラグインを活用している場合があります。jQueryにはjQuery UIも含めた高度なコンポーネントの部品が数多くあります。これらはとても便利ですが、やはりデータと表示の分離がしにくく、うまくレールに乗られるような最速のパターンでは短かったとしても、例えばカレンダー部品で取得した情報を元にバリデーションを行って結果を画面に反映させたり、インタラクションが増えてくると周辺のコードが複雑化しがちです。

まだVue.js 3には対応していませんが、`bootstrap-vue` という人気のあるBootstrapをVue.js用にコンポーネント化したものもあります。複雑なフィルタリング機能などを実現するテーブルコンポーネントも `cheetahgrid` などのよりよい代替部品がすぐに見つかるでしょう。自分でコンポーネントを実装しなければならない場合を除けば、そこまで行数も変わらず実現できるでしょう。

まとめ

Vue.jsについて紹介してきました。まずはVueの使い勝手の良さの源泉であるvue-cliによるプロジェクト作成の方法について紹介しました。TypeScriptフレンドリーな書き方については2系で相性の良いクラス形式の実装方法と、3系のcomposition APIの2種類紹介しました。

せめて、2系にバックポートされた3系 APIを実現するプラグインの[@vue/composition-api](#)のバージョンからベータが外れると、プロダクション開発もすべてcomposition APIを利用する方針で決定できると思いますが、現時点ではどちらを使うかはプロジェクトの寿命やリリース時期を勘案してどちらか決める必要があるかもしれません。

そのあとで、よくありそうなユースケースとして、jQueryで書かれたウェブフロントエンドをVue.jsで置き換える方法について紹介しました。一見すると行数が増えますが、ロジックが増えても複雑度が増えていかないのが、のちのちにありがたみを感じやすくなるでしょう。