コンポーネントの作成

コンポーネントとは

- Vue.js アプリケーションの構成要素である再利用可能な UI 部品
- Vue.js のコンポーネントシステムを使用して定義され、テンプレート、JavaScript のロジックス、スタイルシートをまとめてカプセル化される
- 自己完結型の機能を持っており、他のコンポーネントやアプリケーションの他の部分と独立して動作する

コンポーネントシステムとは

コンポーネントを定義するための仕組みのこと。アプリケーションをより小さな部品に分割し、各部品を独立して開発、テスト、保守をすることができる。

> Vue.js のコンポーネントシステムは、再利用性が高く、保守性と拡張性を向上できる。

- コンポーネントの組み合わせ：コンポーネントは Vue.js のコンポーネントシステムを使用して、他のコンポーネントやアプリケーションの他の部分と組み合わせることができる。
- コンポーネントの通信：コンポーネントは親コンポーネントからデータを受け取ったり、子コンポーネントにデータを渡したりすることができる。このようなコンポーネント間の通信を簡単に行うことができる。

クラススタイル Vue コンポーネント

- クラス構文を使用する
- クラスコンポーネントでは`vue-property-decorator`をインポートする
- `@Component`(デコレーター)と書き、その中に`components`を書く

```
import { Component, Vue } from "vue-property-decorator";
@Component({
  components: {
    Directive,
    ComputedWatch,
    OriginalFilter,
  },
})
```

components の中身は以下を省略した書き方

```
@Component({
  components: {
```

```
    Directive:Directive,
    ComputedWatch:ComputedWatch,
    OriginalFilter:OriginalFilter,
  },
})
```

クラスコンポーネント以外のコンポーネント

(Typescript を導入しない書き方とほぼ同じとなるため JavaScript で解説)

```
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

<div id="app">
  <my-component></my-component>
  <my-component></my-component>
  <my-component></my-component>
  <my-component></my-component>
  <my-component></my-component>
</div>
```

```
Vue.component("my-component", {
  data: function () {
    return {
      count: 0,
    };
  },
  template: '<p>カウント指数{{count}}<button @click="count++">+</button></p>',
});
new Vue({}).$mount("#app");
```

> data は関数で表記する。変数とした場合参照によって共有されるため、1 つのメモリを参照していくことになる。 関数にすることでバラバラに増える

補足 1

`@Component`のようなデコレーターを理解する

デコレーターとは何か

- デコレーターとはクラスを受け取ってデコレーションをする関数。
- TypeScript ではデフォルトでは使えないため、`tsconfig.json` で`"experimentalDecorators": true,`にする必要がある。
- クラス全体、クラスの中の一部分など`@xxx`を置く場所によってデコレーションの適用範囲が変わる。
- class はコンストラクタ関数の糖衣構文

デコレーターはいつ実行されるのか？

- class がインスタンス生成する前に実行される
- **デコレーターはインスタンスの生成時ではなく、クラスの定義時に生成される**

デコレーターファクトリとは何か

- デコレーターにパラメーターを持たせたいという場合がある(勝手に追加することができない)
- 解決策としてデコレーターを返す関数を書く

実例を確認する

```
import { Vue, Component, Prop, PropSync, Emit } from "vue-property-decorator";

@Component
export default class DataDelivery extends Vue {
  @Prop() navItem!: string;
  @Prop() navNumber!: number;
  childMessage: string = "子コンポーネントでセットしたメッセージ";

  //computed
  get navItemUpperCase() {
    return this.navItem.toUpperCase();
  }
  //emit
  //@Emit('渡したい名前')
  @Emit("change-msg")
  changeMsg(): string {
    return this.childMessage;
  }
}
```

- @Componentの型定義をindex.d.tsで確認する

```
declare function Component<VC extends VueClass<Vue>>(target: VC): VC;
```

- @Prop()の型定義をvue-property-decorator.d.tsで確認する

```
export declare function Prop(options?: PropOptions | Constructor[] |
Constructor): (target: Vue, key: string) => void;
```

- @Emit("change-msg")の型定義をvue-property-decorator.d.tsで確認する

```
export declare function Emit(event?: string): (_target: Vue, propertyKey:
string, descriptor: any) => void;
```

補足 2

`vue-property-decorator`とは何か

- `Vue`を`TypeScript`特有のクラス構文で書くためのツール
- Vue CLI でプロジェクト作成時に`Use class-style component syntax`を`yes`とするとインストールされる
- ラップしている`vue-class-component`によりクラス構文が書けるようになっている
- `vue-property-decorator`により様々なデコレーターが使えるようになる

補足 3

`shims-vue.ts`ファイルとは何か

- 単一コンポーネント`.vue`ファイルは通常 TypeScript ファイルとして扱われない
- `.vue`ファイルを`import`する際に、記述されているコードを`TypeScript`として認識させる役割を担っている
- 特別な記述は必要なく、`src`ディレクトリ内にあればよい

補足 4

`tsconfig.json`ファイルとは何か

- TypeScript のコンパイル時に使用される設定ファイル
- Vue CLI のプロジェクト生成時に設定される