2.1 Vue プロジェクト作成の準備

Vue アプリケーションを作成するには、Vue プロジェクトを作成し、そのプロジェクトに対してさまざまなコードを記述していきます。本節では、まず、このプロジェクトを作成、コーディングするための準備、つまり、環境構築を行います。

2.1.1 Vue プログラミングに必要なツール

Vue アプリケーションの作成、すなわち、Vue プログラミングを行うにあたって必要なツールは、次の 3 つです。

■ **Visual Studio Code**

Vue プログラミングにおいては、テキストエディタとして Visual Studio Code を利用することが推奨されています。

■ **Visual Studio Code の拡張機能**

Visual Studio Code の拡張機能として、Volar と TypeScript Vue Plugin をインストールして利用します。**Volar** は、Vue 独特の記法に合わせて、Visual Studio Code でのコーディング途中でエラー表示などを行ってくれる拡張機能です。さらに、**TypeScript Vue Plugin** を入れることで、Vue プログラミングにおける TypeScript 構文のチェックなどにも対応してくれるようになります。

■ **Node.js**

実際の Vue プロジェクトの作成、実行環境として 1.1.7 項で紹介した Node.js を利用します。

次項から、順にインストールを行っていきます。

2.1.2 Visual Studio Code のインストール

プログラミングを行う際に利用するテキストエディタや IDE にはさまざまなものがありますが、Vue プログラミングの推奨エディタは、Microsoft がリリースしている **Visual Studio Code**（**VS Code**）です。そこで、まずこの VS Code をインストールします。なお、すでにインストール済みの場合は、次項に進んでください。

NOTE　WebStorm

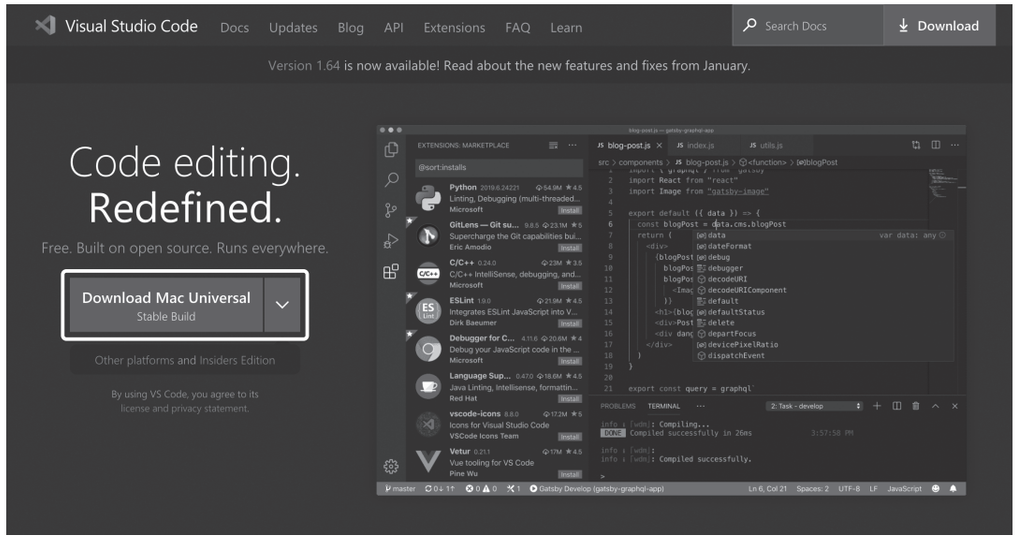
VS Code 以外に、JetBrains 社がリリースしている **WebStorm**[*1] も推奨ツールとされていますが、こちらは学生など一部の対象者以外は有償となっています。

VS Code のサイトにアクセスしてください。URL は次の通りです。

https://code.visualstudio.com/

図 2-1 の画面が表示されます。

▼ **図 2-1　VS Code サイトのトップページ**



このページのダウンロードボタンからファイルをダウンロードしてください。macOS の場合は、zip ファイルがダウンロードされます。これを解凍すると、そのまま VS Code のアプリケーションファイル（app ファイル）となっているので、アプリケーションフォルダに格納してください。Windows の場合は、インストーラとして exe ファイルがダウンロードされます。これを起動して、指示に従ってインストールを行ってください。

インストールが終了したら、VS Code を起動します。初回起動時は、図 2-2 のように英語表示になっています。

---

*1　https://www.jetbrains.com/ja-jp/webstorm/

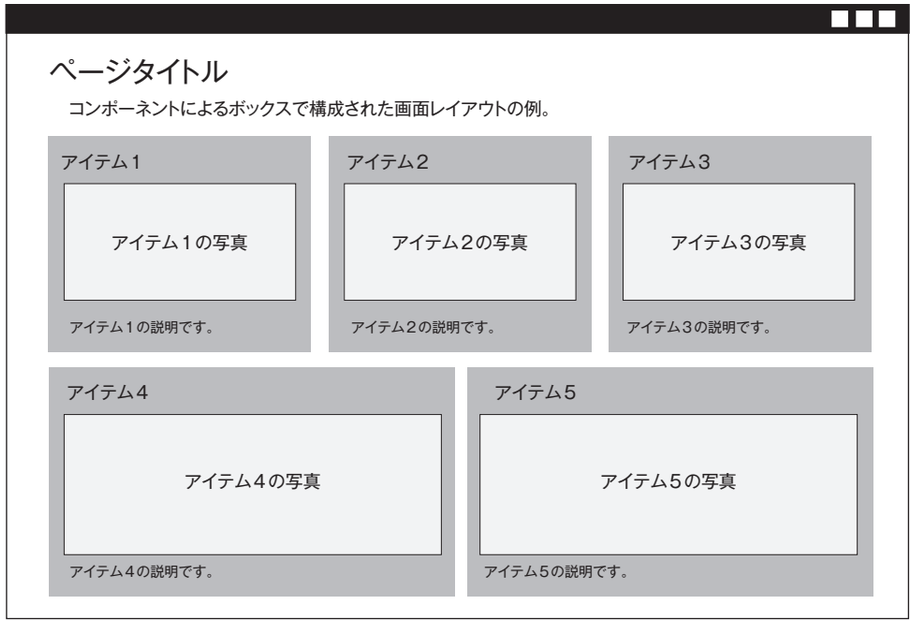
3.1 Vue のコンポーネントと基本構文

Vue プロジェクトでは、コンポーネントというものを使ってアプリケーションを作成していきます。まず、このコンポーネントとは何かから話を始めて、次に、Vue アプリケーションの基本構文を解説していきます。

3.1.1 コンポーネントとは

例えば、図 3-1 のようにタイトルと画像、解説をワンセットのボックスとして繰り返す Web ページを考えてみます。

▼ **図 3-1　ボックスの繰り返しで構成される Web ページ**



このボックスひとつひとつは、例えば、次のような HTML タグで構成されています。

```
<div class="…">
  <h3>…</h3>
  <img src="https://…">
  <p>…</p>
</div>
```

これらのタグは、ワンセットで Web ページのひとつのパーツとなっています。この Web ページの構成パーツのことを、**コンポーネント**と呼びます。このようなコンポーネントは、以下の要素からできています。

- HTML タグ
- CSS スタイルシート
- JavaScript コード

これらの要素は、通常の Web アプリケーションでは、それぞれ別のファイルに記述するのが一般的です。

3.1.2 単一コンポーネントファイルとは

一方、Vue プロジェクトでは、これら 3 個の要素が 1 個のコンポーネント（ファイル）にまとめられています。具体的に見ていきましょう。前章で作成した hello-vue プロジェクトの src/App.vue ファイルを開いて、中のコードを見てください。リスト 3-1 のような構造になっています。

▼ **リスト 3-1　hello-vue/src/App.vue**

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

<template>
  ⋮
</template>

<style>
  ⋮
</style>
```

❶
❷
❸

中身は、大きく次の 3 パートに分かれています。

❶ **script** タグで囲まれた部分。これを**スクリプトブロック**といい、JavaScript（TypeScript）コードを記述します。
❷ **template** タグで囲まれた部分。これを**テンプレートブロック**といい、HTML タグを記述します。
❸ **style** タグで囲まれた部分。これを**スタイルブロック**といい、CSS コードを記述します。

このように、App.vue ファイルでは、本来は別々のファイルに記述していた HTML + CSS + JavaScript（TypeScript）というコンポーネントの 3 個の構成要素をひとつのファイル内に記述できます。このようなファイルを、**単一コンポーネントファイル**といい、拡張子を **.vue** とすることになっています。Vue プロジェクトでのアプリケーション作成は、この .vue ファイルを作成し、そこにさまざまな記述を加えることで進めていきます。

4.1 データバインディングのディレクティブ

第 3 章で登場したマスタッシュ構文は、タグとタグに囲まれた部分（要素のテキスト部分）にしか使えません。ということは、属性にテンプレート変数を利用したい場合、別の方法が必要になります。その際に利用するのが、データバインディングのディレクティブです。本節では、このデータバインディングのディレクティブを紹介します。

4.1.1 ディレクティブとは

ディレクティブとは、テンプレートブロックで HTML タグ内に記述する **v-** で始まる属性のことです。これらディレクティブのうち、主なものを表 4-1 にまとめておきます。

▼ **表 4-1　主なディレクティブ**

ディレクティブ	役割
v-bind	データバインディング
v-on	イベント処理
v-model	双方向データバインディング
v-html	HTML 文字列表示
v-pre	静的コンテンツ表示
v-once	データバインディングを初回のみに制限
v-clock	マスタッシュ構文の非表示
v-if	条件分岐
v-show	表示 / 非表示の制御
v-for	ループ処理

表 4-1 のディレクティブのうち、本章では v-bind と v-on を紹介します。また、v-if、v-show、v-for は、制御のディレクティブとしてまとめて第 6 章で扱います。それ以外は、第 5 章で取り上げます。

4.1.2 属性にデータバインドする v-bind

表 4-1 のディレクティブのうち、最初に紹介するのが **v-bind** です。これはデータバインディングのディレクティブであり、マスタッシュ構文が使えないタグの属性部分に対して、テンプレート変数を利用するためのものです。

簡単な例を見ていきましょう。directive-bind-basic プロジェクトを作成して、src/App.vue をリスト 4-1 の内容に書き換えてください。

▼ **リスト 4-1　directive-bind-basic/src/App.vue**

```
<script setup lang="ts">
import {ref} from "vue";

const url = ref("https://vuejs.org/");
</script>

<template>
  <p><a v-bind:href="url" target="_blank">Vue.jsのサイト</a></p> ―― ❶
  <p><a :href="url" target="_blank">Vue.jsのサイト(省略系)</a></p> ―― ❷
  <p><a v-bind:href="url + 'guide/introduction.html'" target="_blank">Vue.jsガイドのページ</a> ⮐
</p> ―― ❸
</template>
```

表示結果は、図 4-1 の通りです。

▼ **図 4-1　生成されたリンク**

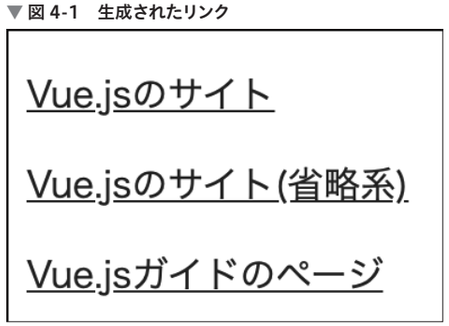


図 4-1 の画面に実際にレンダリングされたタグは、次のようになります。

```
<p><a href="https://vuejs.org/" target="_blank">Vue.jsのサイト</a></p> ―― ①
<p><a href="https://vuejs.org/" target="_blank">Vue.jsのサイト(省略系)</a></p> ―― ②
<p><a href="https://vuejs.org/guide/introduction.html" target="_blank">Vue.jsガイドのページ</a> ―― ③
```

なお、以降のサンプルでは、表示結果とレンダリング結果について、いずれかが説明に不要な場合は省略することがあります。

5.1 双方向データバインディング

第 4 章で紹介したデータバインディングのディレクティブとイベントのディレクティブを組み合わせると、入力フォーム内のコントロールの値とテンプレート変数を双方向にバインディングすることが可能です。この節ではそのような双方向データバインディングを紹介していきます。

5.1.1 双方向データバインディングを実現する v-model

まず、双方向データバインディングとはどのようなものかをイメージしてもらうために、具体例を見てみます。directive-model プロジェクトを作成して、src/App.vue をリスト 5-1 の内容に書き換えてください。

▼ リスト 5-1　directive-model/src/App.vue

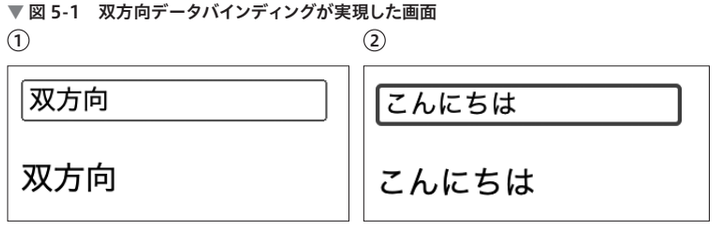
```
<script setup lang="ts">
import {ref} from "vue";

const inputNameModel = ref("双方向"); ❶
</script>

<template>
  <section>
    <input type="text" v-model="inputNameModel"> ❷
    <p>{{inputNameModel}}</p> ❸
  </section>
</template>
```

表示結果は、図 5-1 の通りです。①の状態で入力欄の文字を変更すると、その下の「双方向」というテキスト部分が、②のように入力した内容に変化します。

▼ 図 5-1　双方向データバインディングが実現した画面



まさに、入力したデータがリアルタイムでテンプレート変数と連動することが体感できたと思います。ソースコードとしては非常にシンプルで、入力欄である❷のタグに新たに登場したディレクティブ **v-model** を記述し、その属性値として❶のテンプレート変数を指定しているだけです。これだけで、入力内容とテンプレート変数が自動で連動します。結果は、テンプレート変数 inputNameModel をマスタッシュ構文で表示している❸部分で確認できます。これが双方向データバインディングです。

5.1.2 片方向のデータバインディング

では、なぜこのような双方向データバインディングが成り立つのか、種明かしを行っていくことにします。いきなり双方向バインディングの原理を紹介すると難しく感じやすいので、片方ずつ紹介していきます。

前章の復習も兼ねて、わかりやすいところから始めましょう。まず、具体例を作成します。directive-oneway プロジェクトを作成して、src/App.vue をリスト 5-2 の内容に書き換えてください。

▼ リスト 5-2　directive-oneway/src/App.vue

```
<script setup lang="ts">
import {ref} from "vue";

const inputNameBind = ref("しんちゃん"); ❶
const inputNameOn = ref("ななし"); ❷
const onInputName = (event: Event): void => { ❸
  const element = event.target as HTMLInputElement; ❹
  inputNameOn.value = element.value; ❺
}
</script>

<template>
  <section>
    <input type="text" v-bind:value="inputNameBind"> ❻
  </section>
  <br>
  <section>
    <input type="text" v-on:input="onInputName"> ❼
    <p>{{inputNameOn}}</p> ❽
  </section>
</template>
```

表示結果は、図 5-2 の通りです。①の状態で下側の入力欄に何か文字を入れると、その下の「ななし」というテキスト部分が、②のように入力した内容に変化します。

6.1 条件分岐のディレクティブ

条件分岐のディレクティブは、通常のプログラミングの条件分岐と同じく、if、else if、else をディレクティブにしたものです。順に見ていきましょう。

6.1.1 条件分岐ディレクティブの基本 v-if

条件分岐ディレクティブの基本形は **v-if** です。値として条件を記述します。

具体例を見ていきましょう。directive-conditional-basic プロジェクトを作成して、src/App.vue をリスト 6-1 の内容に書き換えてください。

▼ リスト 6-1　directive-conditional-basic/src/App.vue

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

const number = ref(80);                                          ❶
const showOrNot = computed(                                      ❷
  (): boolean => {
    //戻り値用変数を初期値falseで用意。
    let showOrNot = false;
    //0〜100の乱数を発生。
    const rand = Math.round(Math.random() * 100);
    //乱数が50以上ならば、戻り値をtrueに変更。
    if(rand >= 50) {
      showOrNot = true;
    }
    return showOrNot;
  }
);
</script>

<template>
  <p v-if="number >= 50">                                        ❸
    条件に合致したので表示①
  </p>
  <p v-if="Math.round(Math.random() * 100) >= 50">               ❹
    条件に合致したので表示②
  </p>
  <p v-if="showOrNot">                                           ❺
    条件に合致したので表示③
  </p>
</template>
```


表示結果例は、図 6-1 の通りです。なお、乱数を利用するので、2 行目と 3 行目は表示されない場合があります。

▼ 図 6-1　条件によって表示 / 非表示が変わる画面

条件に合致したので表示①

条件に合致したので表示②

条件に合致したので表示③

図 6-1 のレンダリング結果は、次のようになります。先ほど同様、②と③はレンダリングされない場合もあります。その際、p タグが存在するはずの位置に「<!--v-if-->」というコードがレンダリングされています。

```
<p>条件に合致したので表示①</p>  ①
<p>条件に合致したので表示②</p>  ②
<p>条件に合致したので表示③</p>  ③
```

条件に合致した場合にタグがレンダリングされる

リスト 6-1 の❸〜❺の p タグ内に記述された v-if が、条件分岐のディレクティブの基本です。属性値として条件式を記述しますが、この条件式内ではテンプレート変数が利用できます。実際に、リスト 6-1 の❸では、❶で用意した number を利用しています。

v-if では、条件に合致した場合に、そのタグと配下のタグがレンダリングされます。❶では number の値を 80 としているので、number >= 50 という条件に合致し、レンダリング結果①のように p タグがレンダリングされます。この値を 50 より小さい値、例えば 30 とすると、p タグごとレンダリングされなくなりレンダリング結果①は表示されません。

複雑な式では算出プロパティを活用する

ここで、このテンプレート変数 number の値を乱数にする場合を考えます。v-if の条件部分には JavaScript/TypeScript 式を記述できるため、リスト 6-1 の❹のコードは問題なく動作し、乱数で発生した値が 50 以上の場合はレンダリング結果②がレンダリングされますが、50 より小さい場合はレンダリングされません。

しかし、可読性という観点からは、このようにテンプレートに式を直接記述するのはよいコードとはいえません。何より、3.3.1 項の Note（p.38）で紹介した Vue のスタイルガイドに反します。そこで、このような複

7.1 ウォッチャー

これまでの章でスクリプトブロックに記述してきたのは、ref() 関数や reactive() 関数でリアクティブデータにしたテンプレート変数、算出プロパティ、メソッド、および、それらが利用する元データや関数ぐらいでした。Vue では、これら以外にも、スクリプトブロックで使用できるさまざまな機能が用意されています。まず最初に取り上げるのは、ウォッチャーです。

7.1.1 算出プロパティの役割

ウォッチャーの説明に入る前に、なぜウォッチャーが必要なのかを体感していただくため、これまでの知識で作成可能なアプリケーションをひとつ作成してみましょう。画面表示直後は図 7-1 の①のようにカクテル番号とカクテル情報が表示され、これが図 7-1 の②のように、1 秒ごとにランダムに変化するアプリケーションです。

▼図 7-1　change-cocktail プロジェクトの表示結果

①

現在のカクテル番号: 1

該当するカクテルはホワイトレディで、価格は1200円です。

②

現在のカクテル番号: 2

該当するカクテルはブルーハワイで、価格は1500円です。

change-cocktail プロジェクトを作成して、src/App.vue をリスト 7-1 の内容に書き換えてください。なお、❺のコードは、リスト 6-9 と同じですので省略しています。また、❶のコードは、リスト 6-10 とほぼ同じですが、set() の際に使用するキーの値と各カクテル情報の id の値が違います。リスト 6-10 では ID らしい 4 桁の数値でしたが、リスト 7-1 では乱数と相性のよい 1 ～ 4 の連番にしています。

▼リスト 7-1　change-cocktail/src/App.vue

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

//カクテルリストデータを用意。
const cocktailDataListInit = new Map<number, Cocktail>();
cocktailDataListInit.set(1, {id: 1, name: "ホワイトレディ", price: 1200});
cocktailDataListInit.set(2, {id: 2, name: "ブルーハワイ", price: 1500});
cocktailDataListInit.set(3, {id: 3, name: "ニューヨーク", price: 1100});
cocktailDataListInit.set(4, {id: 4, name: "マティーニ", price: 1500});

//カクテル番号のテンプレート変数を用意。
```

❶


```
const cocktailNo = ref(1);
//カクテル番号に対応するカクテル情報の算出プロパティを用意。
const priceMsg = computed(
  (): string => {
    //カクテル番号に該当するカクテルデータを取得。
    const cocktail = cocktailDataListInit.get(cocktailNo.value);
    //カクテル番号に該当する情報がない場合のメッセージを用意。
    let msg = "該当カクテルはありません。";
    //カクテル番号に該当する情報があるなら…
    if(cocktail != undefined) {
      //カクテル番号に該当するカクテルの名前と金額を表示する文字列を生成。
      msg = `該当するカクテルは${cocktail.name}で、価格は${cocktail.price}円です。`;
    }
    //表示文字列をリターン。
    return msg;
  }
);

//cocktailNoを1秒ごとに1～4の乱数を使って変更。
setInterval(
  ():void => {
    cocktailNo.value = Math.round(Math.random() * 3) + 1;
  },
  1000
);

interface Cocktail {
  ～省略（リスト6-9と同じ）～
}
</script>

<template>
  <p>現在のカクテル番号: {{cocktailNo}}</p>
  <p>{{priceMsg}}</p>
</template>
```

❷

❸

❹

❺

内容的に新しいことはありませんが、少し解説しておきます。❶で用意したカクテルリストデータの ID の値に該当するカクテル番号をリアクティブテンプレート変数として用意しているのが、❷の cocktailNo です。そして、この cocktailNo の値に応じて、該当するカクテル名と金額を用意しているのが❸の算出プロパティです。関数の内部では、cocktailNo の値を元に、❶のカクテルリストから該当データを探し出して表示する文字列を生成しています。

一方、❹では cocktailNo を 1 秒ごとに乱数を使って変更しています。これによって cocktailNo の値が変わると、リアクティブシステムのおかげで、自動的に算出プロパティである priceMsg も変更されます。その結果、表示内容が 1 秒ごとに変わるようになります。

8.5 子から親へのコンポーネント間通信

8.4.2 項末尾で紹介したように、場合によっては子から親へのコンポーネント間通信が必要になることがあります。この通信は **Emit**（エミット）という仕組みで実現できます。Emit を利用しなければ、components-props-indepth の欠陥、すなわち、子コンポーネントで会員ポイントを増やしても会員全員の保有ポイント合計値が変わらない事象は修正できません。本節では、この Emit を紹介します。

8.5.1 子から親への通信はイベント処理

まずは簡単なサンプルを作成して、Emit の仕組みの基本を紹介します。components-emit-basics プロジェクトを用意して src/components フォルダ内に OneSection.vue 作成し、リスト 8-13 の内容を記述してください。

▼ **リスト 8-13　components-emit-basics/src/components/OneSection.vue**

```
<script setup lang="ts">
interface Props {
  rand: number;
}

interface Emits {                                          ❶
  (event: "createNewRand"): void;                         ❷
}

defineProps<Props>();
const emit = defineEmits<Emits>();                         ❸

const onNewRandButtonClick = (): void => {                 ❹
  emit("createNewRand");                                   ❺
}
</script>

<template>
  <section class="box">
    <p>子コンポーネントで乱数を表示: {{rand}}</p>
    <button v-on:click="onNewRandButtonClick">新たな乱数を発生</button>   ❻
  </section>
</template>

<style scoped>
.box {
  border: green 1px solid;
  margin: 10px;
}
</style>
```


次に、src/App.vue をリスト 8-14 の内容に書き換えます。

▼ **リスト 8-14　components-emit-basics/src/App.vue**

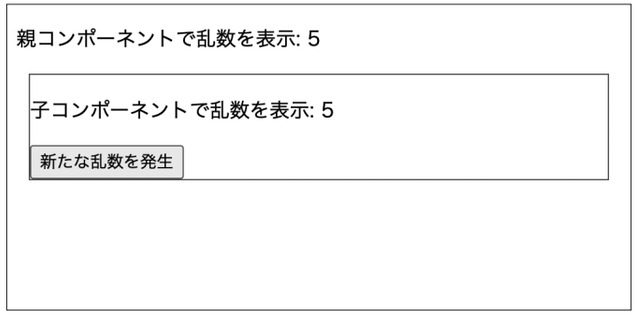
```
<script setup lang="ts">
import {ref} from "vue";
import OneSection from "./components/OneSection.vue";

const randInit = Math.round(Math.random() * 10);
const rand = ref(randInit);
const onCreateNewRand = (): void => {                      ❶
  rand.value = Math.round(Math.random() * 10);
}
</script>

<template>
  <section>
    <p>親コンポーネントで乱数を表示: {{rand}}</p>
    <OneSection
      v-bind:rand="rand"
      v-on:createNewRand="onCreateNewRand"/>               ❷
  </section>
</template>
```

実行結果は、図 8-20 の通りです。この画面中の［新たな乱数を発生］ボタンをクリックすると、表示された乱数の値が親コンポーネント側、子コンポーネント側で同じように変化します。

▼ **図 8-20　components-emit-basics プロジェクトの表示結果**



このサンプルでは、子コンポーネントに配置したボタンをクリックすると、親コンポーネントのリアクティブ変数である rand が更新されます。つまり、子コンポーネントから親コンポーネントへの通信が行われているこ

9.1 子コンポーネントをカスタマイズする Slot

本節で紹介するのは、Slot と呼ばれる機能です。まずは、Props では実現が難しい処理を確認しながら、Slot とは何かという部分から話を始めます。

9.1.1 Slot とは

親コンポーネントから子コンポーネントにデータを渡すには、8.3.1 項で紹介した Props が利用できます。ただし、この Props では HTML 要素そのものを渡すことができません。

例えば、親コンポーネントで次のようなテンプレート変数 tag を用意します。

```
const tag = ref("<p>連絡がつきません。</p>");
```

この tag を、次のように Props 経由で子コンポーネントに渡したとします。

```
<OneSection v-bind:tag="tag"/>
```

子コンポーネントでこれを次のようにマスタッシュ構文で表示しようとした場合、エスケープされてしまうため、HTML 要素として認識されず、あくまでタグ形式の文字列として表示されてしまいます（図 9-1）。

```
<section class="box">
  {{tag}}
</section>
```

▼ 図 9-1　HTML 要素として認識されない

連絡がつきません。

もちろん、5.2.1 項で紹介した v-html ディレクティブを利用し、次のように記述すれば HTML 要素としてレンダリングされます（図 9-2）。

```
<section class="box" v-html="tag">
</section>
```


▼ 図 9-2　v-html ディレクティブだと HTML 要素として認識される

連絡がつきません。

しかし、この方式では 5.2.1 項（p.98）で説明したような XSS 脆弱性が含まれてしまいます。

さらに、v-html ディレクティブを使ったとしても、Props で子コンポーネントに渡すことができるのは静的な HTML 記述が基本です。例えば、次のように v-for によって動的にレンダリングされた HTML 要素を渡すことはできません。

```
<ul>
  <li v-for="problem in problems" v-bind:key="problem">
    {{problem}}
  </li>
</ul>
```

このように、静的であれ動的であれ、HTML 要素を子コンポーネントに渡して、子コンポーネントでそのままレンダリングさせるには、子コンポーネントのレンダリング内容を親コンポーネントがカスタマイズできるような別の仕組みが必要になります。この仕組みこそが **Slot**（スロット）です。

9.1.2 Slot の基本的な記述方法

それでは、具体例を作成しながら、Slot の基本的な記述方法を見ていきましょう。slot-basic プロジェクトを作成して、src/components フォルダ内に OneSection.vue を作成し、リスト 9-1 の内容を記述してください。

▼ リスト 9-1　slot-basic/src/components/OneSection.vue

```
<script setup lang="ts">
interface Props {
  name: string;
}
defineProps<Props>();
</script>

<template>
  <section class="box">
    <h1>{{name}}さんの状況</h1>
    <slot/>  ❶
  </section>
</template>

<style>
.box {
  border: green 1px solid;
```

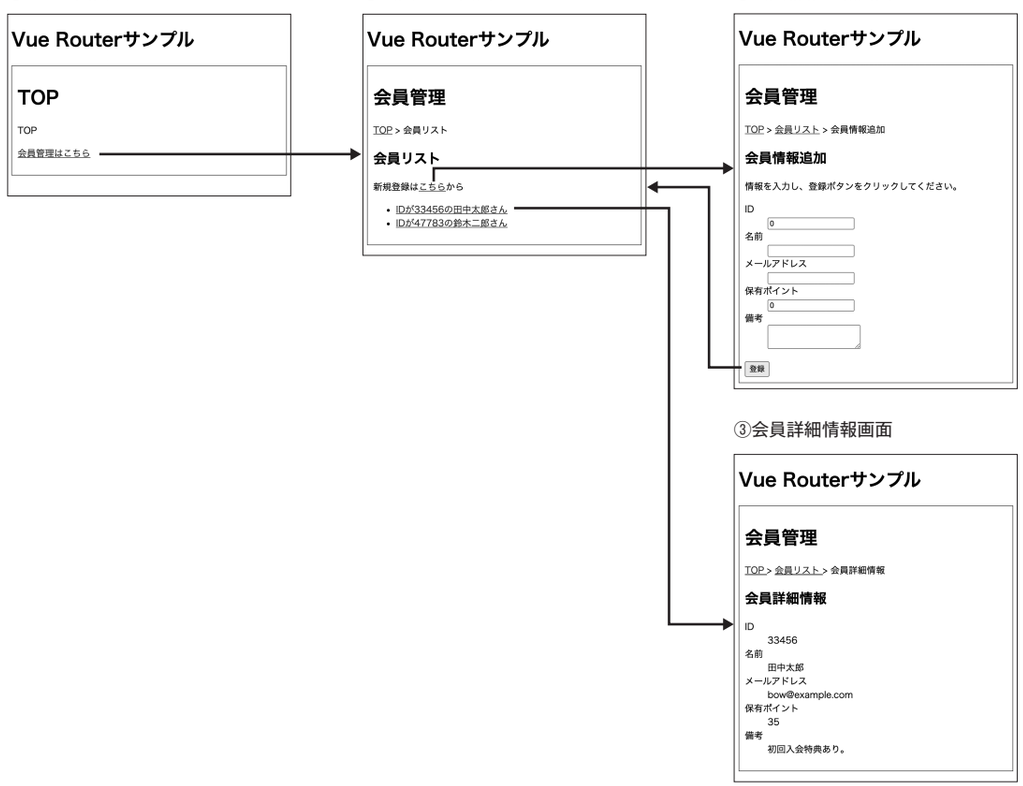
10.1 シングルページアプリケーション

本章で紹介する Vue Router を利用することで、シングルページアプリケーションを簡単に作成できるようになります。まずは、そもそもシングルページアプリケーションとは何かについて、本章で作成するサンプルを題材に紹介していきます。

10.1.1 ルーティングとは

本章で作成するサンプル（router-basic プロジェクト）は 4 画面からできており、それぞれの画面遷移を図にすると、図 10-1 の通りです。表示などに利用するデータは、第 8 章でも使用した会員情報リストを利用します。

▼図 10-1　router-basic プロジェクトの画面遷移



画面番号と、画面名とその画面を表すパス（URL のドメイン以降の部分）の対応関係は、表 10-1 の通りです。

▼表 10-1　router-basic プロジェクトの画面

番号	画面名	パス
①	TOP 画面	/
②	会員リスト画面	/member/memberList
③	会員詳細情報画面	/member/detail/33456
④	会員情報追加画面	/member/add

アプリケーションを表示すると、最初は「① TOP 画面」となります。そのため、この TOP 画面のパスは、/（ルート）となっています。その TOP 画面中の［会員管理はこちら］のリンクをクリックすると、「②会員リスト画面」が表示されます。表 10-1 の通り、その際のパスは /member/memberList です。

「②会員リスト画面」の各会員情報はリンクとなっており、それをクリックすると「③会員詳細情報画面」が表示されます。その際のパスは、表 10-1 では /member/detail/33456 ですが、パス末尾の「33456」はクリックする会員によって変化します。こちらに関しては、10.3 節で紹介します。

一方、「②会員リスト画面」の「新規登録はこちらから」の［こちら］リンクをクリックすると「④会員情報追加画面」が表示されます。その際のパスは、/member/add です。この画面に必要情報を入力して［登録］ボタンをクリックすると、入力した会員情報が保存されて「②会員リスト画面」に戻ります。もちろん、その際、新たに保存された会員情報がリストに追加されています。

このように画面遷移を伴うアプリケーションにおいて、表示させる画面とパスとを結びつけ、特定のパスに対応して画面表示処理を行うことを、**ルーティング**といいます。そして本章で紹介する **Vue Router** は、Vue と組み合わせてルーティング処理を行うライブラリです。次節以降では、ここで紹介した画面遷移の流れを Vue Router で組み込んでいきます。

10.1.2 サーバサイド Web アプリケーションでの処理の流れ

ところで、この画面遷移のアプリケーションを、いわゆるサーバサイド Web アプリケーションで作成した場合、処理の流れは図 10-2 のようになります。

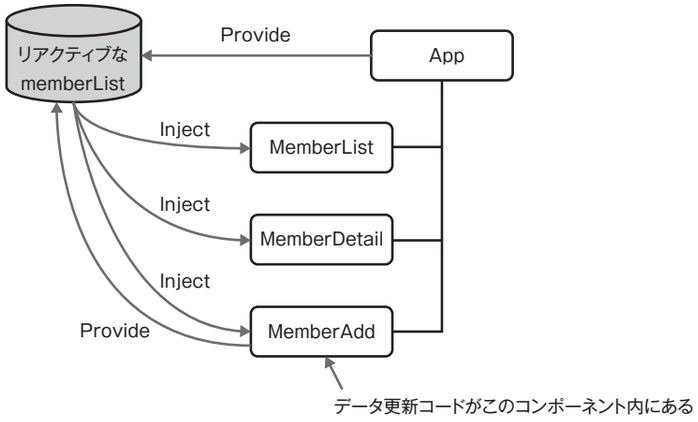
11.1 Pinia の基本

Pinia（ピニア）は、コンポーネント横断でのデータのやり取りに際し、Provide/Inject では力不足な場面で、その問題を解決してくれるものです。まず、Provide/Inject ではどのような問題があるのか、そこから話を始めていきます。

11.1.1 Provide/Inject の問題点と Pinia

前章で作成した router-basic プロジェクトのデータのやり取りを図にすると、図 11-1 のようになります。

▼ **図 11-1　router-basic プロジェクトでのデータのやり取り**



App コンポーネントで memberList を Provide しておきます。Vue Router によって、その App コンポーネントの子コンポーネントとして埋め込まれる MemberList や MemberDetail、MemberAdd コンポーネントでは、Provide された memberList を Inject して利用しています。MemberAdd コンポーネントでは、Inject するだけでなく、Inject した memberList データの変更（要素の追加）も行っています。

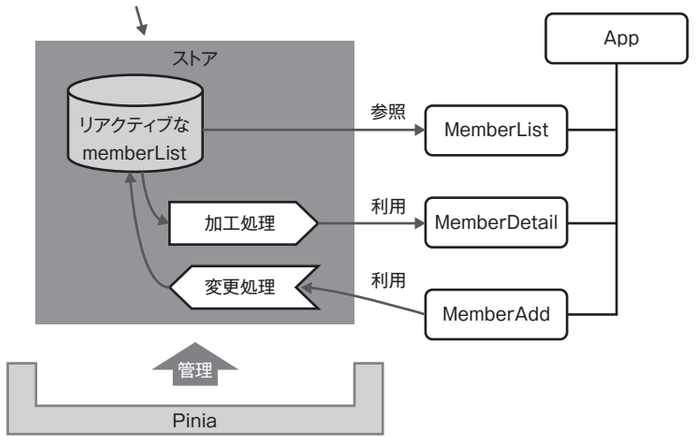
このように、単に Inject したデータを表示するだけでなくデータに変更も加えるとなると、データ変更処理が各コンポーネント内に散在していくことになります。この状態でアプリケーションが大きくなっていくと、どのコンポーネントでどのような変更処理が行われているのか、管理できなくなります。当然、同じような変更処理が数箇所のコンポーネントに記述されている、という状況も発生します。こうなってくると、アプリケーション自体のメンテナンス性が低下してしまいます。

この問題の原因はデータ処理コードの散在ですので、解決する方法は簡単で、1 箇所にまとめることです。

これには例えば、「dataaccess.ts のようなファイルを作成し、その中にデータ処理コードをまとめる」という方法も考えられます。ただし、リアクティブシステムとの関係上、あくまでまとめられるのはデータの変更および加工処理程度であり、データそのものは相変わらず Provide/Inject に頼る必要があります。このような問題を全て解決してくれるのが、**Pinia** です。

Pinia は、**ストア**（**Store**）ライブラリと呼ばれ、コンポーネントやページをまたいで**状態**（**State**）を共有、管理するためのライブラリです。状態を管理するということは、アプリケーションで利用するデータ本体のみならず、そのデータへの変更、加工処理もまとめて管理できることを意味します。もちろん、Pinia で用意したデータはリアクティブシステムの対象です。各コンポーネントでデータを表示したい場合は、ストアで用意されたデータ本体を参照します。また、データの変更や加工したデータの利用を行いたい場合は、同じくストアで用意した専用の関数を呼び出します（図 11-2）。

▼ **図 11-2　ストアを提供する Pinia**



> **NOTE**
> **Vuex**
>
> Vue プロジェクトとして Vue CLI の利用がデフォルトだったころは、Vue と連携するストアライブラリは **Vuex** でした。現在、Vuex はメンテナンスモードとなり、もはや新規機能の追加がされない状態となっています。その代わりとなったのが、Pinia です。

11.1.2 Pinia プロジェクトの作成

概論はこのあたりにして、実際に Pinia を利用していきましょう。まず本節では、簡単なサンプルを作成して、

12.1 非同期処理の基本

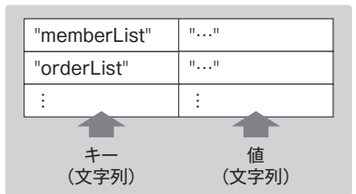
前章の最後に作成した pinia-storage は、外部データサービスとしてセッションストレージを利用しました。ここではその応用として、ブラウザにあらかじめ備わったデータベースである IndexedDB を外部データサービスとして利用するサンプルを作成していきます。

12.1.1 IndexedDB と非同期処理

前章最後の 11.3.5 項の図 11-14 を見ると、セッションストレージの下に［IndexedDB］という選択肢があります。このことからもわかるように、**IndexedDB** はブラウザにあらかじめ備わった機能であり、その名称通りデータベースの一種です。

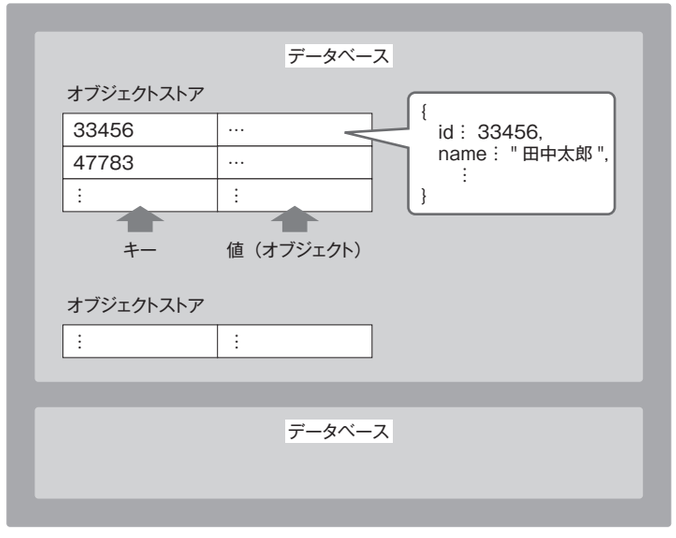
pinia-storage のストレージとのやり取りのコードからもわかる通り、ストレージは図 12-1 のような単純な構造となっており、キーも値も文字列しか格納できません。そのため、オブジェクトを格納したい場合は、pinia-storage サンプルで行ったように、いったん JSON 文字列化する必要がありました。

▼ **図 12-1　ストレージには単純な文字列しか格納できない**
ストレージ



一方 IndexedDB は、キーに対応する値としてオブジェクトをそのまま格納できます（図 12-2）。

▼ **図 12-2　IndexedDB にはオブジェクトをそのまま格納できる**
IndexedDB



このようにデータを格納する入れ物を、**オブジェクトストア**といいます。いわゆるリレーショナルデータベースでいえば、これがテーブルに該当します。格納するデータの種類に応じて、オブジェクトストアはいくつでも作成できます。さらに、これらのオブジェクトストアをまとめて入れておくものがデータベースであり、データベースもまた複数作成できます。もちろん、各オブジェクトストアへのデータの追加や削除も自由にできます。この IndexedDB を利用する場合、大きく次のふたつの手順が必要です。

1. データベースオブジェクトを取得する
2. 1 のデータベースオブジェクトを通じてデータ処理を行う

さらに、それぞれの手順が細かく分かれていきます。これらは、全て非同期処理として行う必要があります。

12.1.2 同期処理と非同期処理の基本

そこで、IndexedDB を利用するサンプルを紹介する前に、非同期処理の基本を紹介します。なお、非同期処理に関して細かく解説しようとすると、それだけでひとつの本が出来上がるぐらいの分量になるので、ここでは簡単な紹介にとどめることをご了承ください。

まず、非同期処理の逆である同期処理に関して、一体何を同期しているのかを確認しておきます。図 12-3 を見てください。

13.1 ユニットテストと Vitest

テストにはさまざまな技法がありますが、本章で紹介するユニットテストはそのひとつです。ここではまず、ユニットテストとは何かということから話を始めていきます。なお、本章で紹介する Vue アプリケーションのユニットテストだけでも、細かい内容も含めると 1 冊の書籍にできるほどの内容があります。ここでは、あくまで導入的な内容の紹介にとどめることをご了承ください。

13.1.1 ユニットテストとは

ユニットテストとは何かを理解するために、ひとつ例を挙げます。utest-basic というプロジェクトで構成されるアプリケーションがあるとします（このプロジェクトは、のちほど実際に作成します）。その中に関数をまとめたファイル functions.ts が含まれており、そこにリスト 13-1 の divideTwoNums() という関数があるとします。

▼ リスト 13-1　utest-basic/src/functions.ts

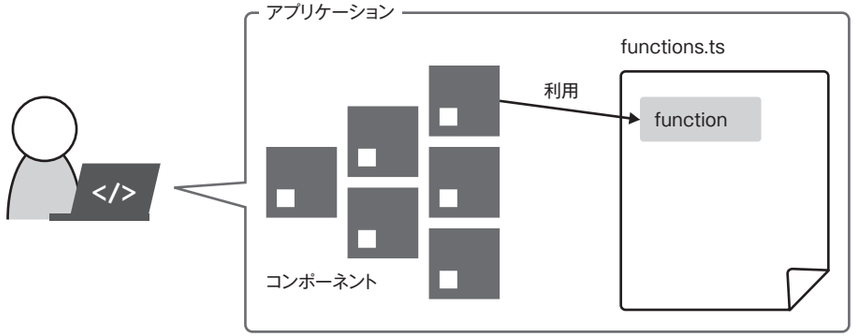
```
export function divideTwoNums(num1: number, num2: number): number {
  return num1 / num2;
}
```

引数として数値を 2 個受け取り、その引数で割り算を行った結果をリターンするだけの単純な関数です。ここではあくまで例として単純な処理としていますが、これは話を簡単にするためです。実際のアプリケーションで利用するような複雑な関数でも、全く同様に考えることができます。

この divideTwoNums() 関数は、アプリケーションの中で利用されます。例えば、Vue アプリケーションでは、ひとつのコンポーネントだけではなく、さまざまなコンポーネントを組み合わせて画面を構成しますが、この関数はそれらのコンポーネントから利用されます。さらに、ひとつの画面表示が出来上がるまでには、コンポーネントのルーティング制御を行う Vue Router やデータ処理を行う Pinia、外部のストレージや Web API など、さまざまなものが組み合わされています。

では、テストはどうかというと、これまではアプリケーションを起動して、画面操作を通して動作確認を行ってきました。その動作確認の中で、そこで使用している関数（例えば、divideTwoNums() 関数）の挙動も、結果として、合わせてテストしていたことになります（図 13-1）。このようなテスト手法を、**結合テスト**といいます。

▼ 図 13-1　画面操作を通した動作確認は結合テスト



本書でサンプルとして紹介してきた規模のアプリケーションならば、結合テストでも問題ありません。しかし、アプリケーションの規模が大きくなると、さまざまな構成要素が複雑に絡まり、ひとつバグが発生しても、どの要素が発生源なのか判断がつきにくいことが出てきます。このような問題を避けるには、構成要素ひとつひとつをテストできた方が望ましいといえます。

例えば、先の divideTwoNums() 関数ならば、この関数だけをテストし、関数そのものにはバグがないことが保証されていると、結合テストを行う際も安心できます。このように、構成要素ひとつひとつをテストすることを、**単体テスト**、あるいは、**ユニットテスト**といいます。

13.1.2 ユニットテスト対応プロジェクトの作成と Vitest

ユニットテストは、専用のツールを利用しなければ簡単には実行できません。Vite 上で動作するツールとしては、**Vitest**[*1] というものがリリースされており、これは create-vue で Vue プロジェクトを作成する際にプロジェクトに組み込むことが可能です。

では実際に、ユニットテストに対応したプロジェクトとして、utest-basic プロジェクトを作成しましょう。このプロジェクトでは、Vue Router と Pinia は使いませんので、Vue プロジェクト作成時の質問 4 と 5 は「No」を選択してください。一方、Vitest を含んだプロジェクトとする必要があるため、質問 6「Add Vitest for Unit Testing?」で「Yes」を選択してください。それ以外は、これまでのプロジェクト作成手順と同様です（図 13-2）。

▼ 図 13-2　Vue プロジェクト作成時の質問 6 で「Yes」を選択

```
Vue.js - The Progressive JavaScript Framework

✔ Project name: … utest-basic
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit Testing? … No / Yes
✔ Add Cypress for End-to-End testing? … No / Yes
✔ Add ESLint for code quality? … No / Yes
```

＊1　https://vitest.dev/