# WEBAPP勉強会 第3回目 解説



---

## 編集ダイアログの実装

### ダイアログを実装する

以下を参考にフォームを内包したダイアログを作成していきます。

https://element.eleme.io/#/en-US/component/dialog#customizations

```
<el-button type="text" @click="dialogFormVisible = true">open a Form nested
Dialog</el-button>
```

起動(表示状態)するには `dialogFormVisible` の値を `true` に設定します。

### 基本的なダイアログ

```
<el-dialog title="Shipping address" :visible.sync="dialogFormVisible">

  <!-- ダイアログの中身をHTMLを使って書く -->

  <!-- <span slot="footer"...> ~ </span> は ボタンを表示するエリア-->
```

```
    <span slot="footer" class="dialog-footer">
      <el-button @click="dialogFormVisible = false">Cancel</el-button>
      <el-button type="primary" @click="dialogFormVisible = false">Confirm</el-
  button>
    </span>

  </el-dialog>
```

`:visible.sync` にダイアログの表示／非表示を制御する変数 `dialogFormVisible` をバインドします。

`<span slot="footer" ...> ~ </span>` にはダイアログ右下部分に表示されるボタンを記述します。
`@click="..."` にボタンがクリックされたときの動きを記述します。 ダイアログを閉じるためには、`dialogFormVisible` の値を `false` に設定します。

## 基本的な入力フォーム

参考URL：https://element.eleme.io/#/en-US/component/form

```
  <el-form :model="form">
    <el-form-item label="ID" :label-width="formLabelWidth">
      <el-input v-model="form.id"></el-input>
    </el-form-item>

    <!-- その他入力項目 v-model="form.xxx" でバインドしていく -->

  </el-form>
```

`form` というオブジェクトに入力フォームをバインドします。
入力項目は、`<el-form-item> ~ </el-form-item>` 内に記述します。`label` に表示名を、`:label-width` に幅を設定します。
ラベル幅は一括で指定出来るように変数化( `formLabelWidth` )をしています。

## 入力項目に関して

入力項目を必要なだけ記述します。
テキスト入力の場合は、`<el-input v-model="form.xxx"></el-input>` となります。

```
  <el-form-item label="入力項目" :label-width="formLabelWidth">
    <el-input v-model="form.xxx"></el-input>
  </el-form-item>
```

オプションについては次を参照：https://element.eleme.io/#/en-US/component/input

選択入力の場合は、次のような形となります。

```
<el-form-item label="選択項目" :label-width="formLabelWidth">
  <el-select v-model="form.yyy" placeholder="選択してください">
    <el-option label="表示名1" value="one"></el-option>
    <el-option label="表示名2" value="two"></el-option>
    <el-option label="表示名3" value="three"></el-option>
  </el-select>
</el-form-item>
```

詳細なオプション・例は次を参照：https://element.eleme.io/#/en-US/component/select

### SCRIPT部分の説明

ダイアログの制御、入力フォームデータ保持のために、script部分に変数を記述します。

```
formLabelWidth: '150px',    // フォームラベル幅を一括指定する
dialogFormVisible: false,   // ダイアログ表示制御変数：初期値はfalse(非表示)
form: {          // 入力フォームデータ保持のためのオブジェクト
  id: null,      // ID
  title: '',     // タイトル
  content: '',   // 内容
  status: ''     // 状態：TODO | PROGRESS | DONE
}
```

---

## 実装していきます。

### ダイアログ部分

```
<template>
  <div>
    <el-row ...>
      :
    </el-row>

    <!-- この部分に下記の <div> ~ </div> を追加します -->

  </div>
</template>
```

```
<!-- 編集入力 Dialog -->
<div>
  <el-dialog title="Todo 編集" :visible.sync="dialogFormVisible">
    <el-form :model="form">
      <el-form-item label="ID" :label-width="formLabelWidth">
        <el-input v-model="form.id" disabled></el-input>
      </el-form-item>
```

```
        <el-form-item label="タイトル" :label-width="formLabelWidth">
          <el-input v-model="form.title"></el-input>
        </el-form-item>
        <el-form-item label="内容" :label-width="formLabelWidth">
          <el-input v-model="form.content"></el-input>
        </el-form-item>
        <el-form-item label="状態" :label-width="formLabelWidth">
          <el-select v-model="form.status" placeholder="選択してください">
            <el-option label="TODO" value="TODO"></el-option>
            <el-option label="PROGRESS" value="TODO"></el-option>
            <el-option label="DONE" value="DONE"></el-option>
          </el-select>
        </el-form-item>
      </el-form>
      <span slot="footer" class="dialog-footer">
        <el-button @click="dialogFormVisible = false">中止</el-button>
        <el-button type="primary" @click="dialogFormVisible = false">確定</el-
button>
      </span>
    </el-dialog>
  </div>
```

## スクリプト部分

`date() { ~ }` 部分に変数を追加します。

```
  data() {
    return {
      tableData: [
        :
      ],
      search: '',
      formLabelWidth: '150px',
      dialogFormVisible: false,
      form: {
        id: null,
        title: '',
        content: '',
        status: ''
      }
    }
  },
```

編集ボタンクリック時にダイアログが表示されるように`handleEdit` に`this.dialogFormVisible`を`true`にする行を追加します。

```
  handleEdit(index, row) {
    console.log(index, row)
```

```
    this.dialogFormVisible = true
  },
```

### 動作確認

ここまで出来たら、一覧の編集アイコンをクリックしてダイアログが表示されることを確認します。
また、ダイアログの「中止」または「確定」ボタンを押すと、ダイアログが閉じます。(非表示)

動作しましたでしょうか？

開いたダイアログの入力欄には何も表示されていませんので、これから、その中身を表示する実装を行います。

---

## 編集動作の実装

編集ボタンをクリックされた時に、該当する行の内容をダイアログに表示する動きを実装します。
(ただし、バックエンドとつないだ時には、バックエンドにデータ取得しに行くように後から変更します。)

### クリックされた行の内容を取得するには？

行の編集アイコンがクリックされたときに呼ばれる処理が `handleEdit` です。
一覧表示の末尾に編集アイコンを追加した際に、`@click="handleEdit(scope.$index, scope.row)"` と設定しています。

```
handleEdit(index, row) {
  console.log(index, row)
  this.dialogFormVisible = true
},
```

この引数である`index`と`row`にクリックされた行の情報がセットされています。
例えば、ID=1の行がクリックされたとき、`index`と`row`には次のような値が入っています。

```
index: 0
row: {id: '1', title: 'WEBApplication作成', content: 'NUXT.js+ElementUIでフロントエンドのWEBアプリを作成する', status: 'DONE'}
```

### ダイアログのデータに転送する

```
handleEdit(index, row) {
  this.form = row
  this.dialogFormVisible = true
},
```

これで良いような気もしますが、これだと問題があります。
この記述方法だと、`row` のデータの住所がコピーされただけなので、ダイアログ内で値を編集すると一覧表示されている内容まで変更されてしまいます。それでは中止した場合表示が元に戻らないのでまずいです。

このような場合は、項目毎にコピーする必要があります。

```
handleEdit(index, row) {
  this.form.id = row.id
  this.form.title = row.title
  this.form.content = row.content
  this.form.status = row.status
  this.dialogFormVisible = true
},
```

項目が多い場合 **面倒** ですね！
`foreach` とかで項目数分ループしますか？

最近のJavaScriptでは、全く同じ型なら、次のようにすれば1行で中身のコピーが完了します。

```
  this.form = { ...row }
```

これでOK？

通常は検索した時点のデータは古い(他で更新された)可能性があるので、編集時には再度DBへクエリーを発行し、最新データを取得します。
今回のアプリも編集前に一度DBからデータを取得してその内容をダイアログに表示するようにします。
ただし、現時点ではバックエンドは何もありませんので、ダミーの関数を作成し対応します。

次のような流れになるように構成していきます。

1. 編集アイコンをクリックする
2. handleEditがコールされる
    1. 選択行のINDEXを保存(後で使用予定)
    2. 選択行のIDをキーにデータを取得 fetchKey(id)
        1. idをキーにクエリを実行
        2. 結果をitemというオブジェクトに保存
    3. formにitemの内容をコピーする
    4. ダイアログを表示する

## クエリーのダミー処理作成

2.2の選択行のIDをキーにデータ取得する部分のダミーをmethods内に作ります。

```
fetchKey(id) {
  // ToDo: REST APIのkey(row.id)検索し結果をitemにセットする
  // Dummy select
  const items = this.tableData.filter((data) => data.id === id)
```

```
    this.item = { ...items[0] }
  },
```

`this.tableData.filter...` の部分は、tabledataの配列で、idが入力のidと同じものだけに絞り込み、その配列を`items`に入れる　という処理です。
参考URL: filterで配列のデータを抽出する方法

以上を踏まえて、methodsは次のようになります。

```
methods: {
  handleEdit(index, row) {
    this.rowNumber = index
    this.fetchKey(row.id)
    this.form = { ...this.item }
    this.dialogFormVisible = true
  },
  handleDelete(index, row) {
    console.log(index, row)
  },
  fetchKey(id) {
    // ToDo: REST APIのkey(row.id)検索し結果をitemにセットする
    // Dummy select
    const items = this.tableData.filter((data) => data.id === id)
    this.item = { ...items[0] }
  }
}
```

また、data() 部分は以下のようになります。

```
data() {
  return {
    tableData: [
      :
    ],
    search: '',
    formLabelWidth: '150px',
    dialogFormVisible: false,
    rowNumber: '',
    form: {
      id: null,
      title: '',
      content: '',
      status: ''
    },
    item: {
      id: null,
      title: '',
      content: '',
      status: ''
```

```
      }
    },
  },
```

### 動作確認

ここまで出来たら、一覧の編集アイコンをクリックしてダイアログが内に選択行のデータ表示されることを確認します。
ダイアログ内のデータを編集しても、一覧表示は変更されないことを確認します。

動作しましたでしょうか？

次に、「確定」ボタンをクリックして編集結果を反映する処理を実装します。

---

## 確定ボタン動作の実装

確定ボタンをクリックしたときの動作をダミーで実装します。
バックエンドとつないだ時には、更新のAPIをコールするように変更します。

確定ボタンクリック後の流れを次のように構成していきます。

1. 確定ボタンクリック　`@click="doExecute()"`
2. 確定ボタン処理(`doExecute()`)
    1. 更新処理を実行(`updateItem(this.form)`)
        1. DB更新処理のダミー処理
        2. DBから1件検索(`fetchKey(id)`)
        3. 取得内容をテーブルデータに反映
        4. 更新結果メッセージ表示

### 更新処理の本体部分

```
updateItem(param) {
  // ToDo: 本来はREST APIのアップデートを起動する
  this.tableData[this.rowNumber] = { ...param } // これは本来ならばDB側の更新となります

  // 更新後の情報を取得しtableDataにセットする
  this.fetchKey(param.id)
  this.tableData[this.rowNumber] = { ...this.item }

  const target = `${param.id}:${param.title}`
  this.$message({
    type: 'success',
    message: `${target} : 更新が成功しました。`,
    showClose: true,
    duration: 5000
  })
}
```

更新処理実行後、IDによる検索を行い、結果をitemに保存する。
現在編集中の行(INDEXがthis.rowNumber)にitemの内容をコピーする。

処理結果のメッセージについてはメッセージの部品を利用します。
参考URL：https://element.eleme.io/#/en-US/component/message

### methods部分

```
methods: {
  handleEdit(index, row) {
    this.isUpdate = true
    this.rowNumber = index
    this.fetchKey(row.id)
    this.form = { ...this.item }
    this.dialogFormVisible = true
  },
  handleDelete(index, row) {
    console.log(index, row)
  },
  doExecute() {
    this.dialogFormVisible = false
    this.updateItem(this.form)
  },
  fetchKey(id) {
    // ToDo: 本来はREST APIのkey(row.id)検索し結果をitemにセットする
    const items = this.tableData.filter((data) => data.id === id)
    this.item = { ...items[0] }
  },
  updateItem(param) {
    // ToDo: 本来はREST APIのアップデートを起動する
    this.tableData[this.rowNumber] = { ...param }

    // 更新後の情報を取得しtableDataにセットする
    this.fetchKey(param.id)
    this.tableData[this.rowNumber] = { ...this.item }

    const target = `${param.id}:${param.title}`
    this.$message({
      type: 'success',
      message: `${target} : 更新が成功しました。`,
      showClose: true,
      duration: 5000
    })
  }
}
```

### 起動部分

```
<el-button type="primary" @click="doExecute()">
  <i class="el-icon-circle-check" style="font-size: 120%"></i>
```

```
    <span>確定</span>
  </el-button>
```

### 動作確認

編集ダイアログ内を編集し、確定ボタンをクリックした後選択行にデータが反映されることを確認します。

動作しましたでしょうか？

第3回目の実習は以上です。