

Управление приложением

- Введение в условия и циклы
- Взаимодействие с пользователем
- Реактивное создание сложных структур
- Доступ к DOM шаблона
- Использование веб-хранилища

Введение в условия и циклы

Директива v-if

Управлять присутствием элемента в DOM шаблона можно с помощью директивы **v-if**.

Пример `example_1` демонстрирует возможность связывания данных не только с текстом и атрибутами, но и со структурой DOM. Более того, Vue имеет мощную **систему анимаций**, которая может автоматически применять **эффекты переходов** при добавлении, обновлении или удалении элементов.

`example_1`. Тестируем v-if

```
<div id="app">
  <div v-if="show">
    <h2>Личный кабинет бэтмена</h2>
     <br>
    <b>Позиция:</b> Добро <br>
    <b>Альтер-эго:</b> Брюс Уэйн <br>
    <b>Вид:</b> Человек <br>
    <b>Рост:</b> 188 см <br>
    <b>Вес:</b> 95 кг <br>
```

```
<b>Семейное положение:</b> Холост <br>

</div>

</div>

<script>

  // инициализация экземпляра Vue

  var vm = Vue.createApp({

    data() {

      return {

        show: true

      }

    }

  }).mount("#app");

</script>
```

В примере достаточно изменить значение **show** с **true** на **false**, чтобы увидеть эффект скрытия элемента.

Директива v-for

Существует и некоторое количество других директив, каждая из которых обладает своей особой функциональностью. Например, директиву **v-for** можно использовать для отображения **списка элементов**, используя данные из **массива**.

Вывод массива выполним в виде набора **хэш-тегов**.

example_2. Тестируем вывод элементов массива

```
<div id="app">

  <!-- массив в виде набора хэш-тегов -->

  <span v-for="elem in langs">
```

```
      #{{ elem }}

    </span>

  </div>

<script>

  // инициализация экземпляра Vue

  var vm = Vue.createApp({

    data() {

      return {

        langs: ["php", "java", "javascript", "perl",
          "python", "c#", "vb"]

      }

    }

  }).mount("#app");

</script>
```

Так как в реальных приложениях данные представлены, как правило, в более сложных структурах выведем данные **двумерного массива**.

Вывод выполним в виде **маркированного списка**.

example_3. Тестируем вывод элементов двумерного массива

```
<div id="app">

  <ul>

    <li v-for="item in albums">

      {{ item[0] }} ({{ item[1] }})

    </li>

  </ul>

</div>

<script>
```

```
// инициализация экземпляра Vue

var vm = Vue.createApp({
  data() {
    return {
      albums: albums
    }
  }
}).mount("#app");

</script>
```

С помощью этого же атрибута легко выводить **список элементов**, используя данные **объекта**.

Вывод выполним в виде **таблицы**.

example_4. Тестируем вывод элементов объекта

```
<div id="app">

  <table border="1" width="500">

    <tr>

      <th>Альбом</th><th>Дата выхода</th>

    <tr>

      <tr v-for="item in albums">

        <td> {{ item.name }} </td>

        <td> {{ item.date }} </td>

      </tr>

    </table>

  </div>

<script>

  // инициализация экземпляра Vue
```

```
var vm = Vue.createApp({  
  data() {  
  
    return {  
      albums: albums  
    }  
  }  
}).mount("#app");  
</script>
```

Взаимодействие с пользователем

Стандартные диалоговые окна

Так как при написании приложений мы будем использовать возможности взаимодействия пользователя с браузером, нам нужно вспомнить функции его интерфейса, а именно: **alert**, **prompt**, **confirm**.

alert

Функция показывает сообщение и ждёт, пока пользователь нажмёт кнопку **ОК**.

```
alert("Hello");
```

Это небольшое окно с сообщением называется **модальным окном**.

Понятие модальное означает, что пользователь **не может взаимодействовать с интерфейсом остальной части страницы**, нажимать на другие кнопки и т.д. до тех пор, пока взаимодействует с окном.

В данном случае – пока не будет нажата кнопка **ОК**.

confirm

```
let result = confirm(question);
```

Функция **confirm** отображает **модальное окно** с текстом вопроса **question** и двумя кнопками: **ОК** и **Отмена**.

Результат – **true**, если нажата кнопка **ОК**. В других случаях – **false**.

example_5. Диалоговые окна confirm, alert

```
<div id="app">

  {{ quest() }}

</div>

<script>

  // инициализация экземпляра Vue

  var vm = Vue.createApp({

    methods: {

      quest() {

        let res = confirm("Нажмите любую из кнопок");

        //

        if (res) {

          alert("Вы нажали кнопку ОК");

        } else {

          alert("Вы нажали кнопку Отмена");

        }

      }

    }

  })
```

```
}).mount("#app");
```

```
</script>
```

prompt

```
let result = prompt(title, [default]);
```

Этот код отобразит **модальное окно** с текстом, полем для ввода текста и кнопками ОК/Отмена.

- **title** – текст для отображения в окне.
- **default** – необязательный второй параметр, который устанавливает начальное значение в поле для текста в окне.

Вызов **prompt** возвращает текст, указанный в поле для ввода, или **null**, если ввод отменён пользователем. Введённый текст будет присвоен переменной **result**.

example_6. Диалоговое окно prompt

```
<div id="app">

  {{ auth() }}

  <div v-if="isPersonal">

    <h2>{{ name }}</h2>

    <p>Логин: <b>{{ login }}</b></p>

    <p>Пароль: <b>{{ pwd }}</b></p>

    

  </div>

</div>

<script>

  // инициализация экземпляра Vue

  var vm = Vue.createApp({

    data() {

      return {
```

```
        name: "Инкогнито",
        login: undefined,
        pwd: undefined,
        img: "no-admin.png",
        isPersonal: false
      },
    },
    methods: {
      auth() {
        login = prompt("Введите Логин", "admin");
        pwd = prompt("Введите Пароль", "12345");
        this.isPersonal = true;
        this.login = login;
        this.pwd = pwd;
        if (login == "admin" && pwd == "12345") {
          this.name = "Администратор",
          this.img = "admin.png"
        }
      }
    }
  }) .mount("#app");
</script>
```

Директива v-on

Чтобы позволить пользователям взаимодействовать с приложением, можно использовать директиву **v-on** для **обработчиков событий**, которые будут вызывать **методы экземпляра**.

При использовании директивы допускается применение сокращенной формы записи.

example_7. Обработка событий

```
<div id="app">

  <div id="con">

    Имя <input name="name" value="Денис" /><p>

    Логин <input name="login" value="denis" /><p>

    Email <input name="email" value="denis@mail.com" /><p>

  </div>

  <button @click="addData()">Добавить данные</button><p>

  <button @click="outData()">Вывести данные</button>

</div>

<div id="resp"></div>

<script>

  var vm = Vue.createApp({

    data() {

      return {

        user : {

          name : 'Нет данных',

          login : 'Нет данных',

          email : 'Нет данных'

        }

      }

    },

    methods: {

      addData() {

        let con =
```

```
document.querySelector("#con").getElementsByTagName("input");

//

with(this.user) {

    name = con[0].value;

    login = con[1].value;

    email = con[2].value;

};

},

outData() {

    let resp = prompt("Вывести данные в браузер (1) /  
консоль (0)", "1");

    if (parseInt(resp)){

        with(this.user) {

            html = `

            <h3>Вы добавили следующие данные</h3>

            Имя: <b>${name}</b><br>

            Логин: <b>${login}</b><br>

            Email: <b>${email}</b><br>

            `;

            el = document.querySelector("#resp");

            el.innerHTML = html;

        };

    } else { console.log(this.user) }

}

}).mount("#app");

</script>
```

Обратите внимание, в коде примера использованы нативные JavaScript методы манипуляции данными. Такое положение вещей не совсем уместно, фреймворк предоставляет свои способы, манипуляции, и там, где это возможно, желательно использовать возможности фреймворка, делая таким образом код чище и более предсказуемым. Далее познакомимся со специальным атрибутом – **ref** для этих целей.

Директива v-model

До сих пор мы рассматривали реактивность приложения в направлении от экземпляра Vue к шаблону.

Vue также предоставляет директиву `v-model`, которая реализует двустороннюю привязку между элементом формы и состоянием приложения.

example_8. Тестируем директиву `v-model`

```
<div id="app">

  <!-- -->

  <input v-model="message" />

  <!-- -->

  <div style="min-height: 50px;">

    <h2>{{ message }}</h2>

  </div>

</div>

<script>

  var vm = Vue.createApp({

    data() {

      return {

        message: ""

      }

    }

  })
```

```
    }  
  
    }).mount("#app");  
  
</script>
```

Очень удобно использовать привязку так, как это показано в примере example_9.

example_9. Привязка элементов формы к объекту в свойстве data

```
<div id="app">  
  <div id="con">  
    Имя <input v-model="user.name" placeholder="Имя"><p>  
    Логин <input v-model="user.login" placeholder="Логин"><p>  
    Email <input v-model="user.email" placeholder="Email"><p>  
  </div>  
  <button @click="outData()">Вывести данные</button>  
</div>  
<div id="resp"></div>  
<script>  
  var vm = Vue.createApp({  
    data() {  
      return {  
        user : {  
          name : '',  
          login : '',  
          email : ''  
        }  
      }  
    },  
    methods: {
```

```
outData() {  
  
    let resp = prompt("Вывести данные в браузер (1) /  
консоль (0)", "1");  
  
    if (parseInt(resp)){  
  
        with(this.user) {  
  
            html = `  
  
            <h3>Вы добавили следующие данные</h3>  
  
            Имя: <b>${name}</b><br>  
  
            Логин: <b>${login}</b><br>  
  
            Email: <b>${email}</b><br>  
  
            `;  
  
            el = document.querySelector("#resp");  
  
            el.innerHTML = html;  
  
        };  
  
    } else { console.log(this.user) }  
  
    }  
  
    }  
  
    }).mount("#app");  
  
</script>
```

Реактивное создание сложных структур

В следующем примере рассмотрим пример инициализации объекта экземпляра с подготовкой его к отправке в адресной строке в качестве **GET-параметра**.

example_10. Подготовка данных к отправке в другой сценарий

```
<div id="app">  
  
    <!-- -->
```

```
Имя <input v-model="user.name" /><p>

Логин <input v-model="user.login" /><p>

Email <input v-model="user.email" /><p>

<a href="" v-on:click.prevent="sendData()">Отправить
данные</a>

</div>

<script>

var vm = Vue.createApp({
  data() {
    return {
      user : {
        name : '',
        login : '',
        email : ''
      }
    }
  },
  methods: {
    sendData() {
      // упакуем данные в JSON-формат
      userJSON = JSON.stringify(this.user);
      // добавим данные в GET-параметр для отправки на
      другой ресурс
      location.href = event.target.href +
        `?user=${userJSON}`;
    }
  }
}).mount("#app");
```

</script>

Доступ к DOM шаблона

Чтобы получить прямой доступ к DOM элементу шаблона можно воспользоваться специальным атрибутом **ref** (референция). Атрибут **ref** может быть добавлен на любой элемент шаблона

```
<h2 ref="head"> {{ title }} </h2>
```

Применение **ref-атрибута** к определенному элементу шаблона дает возможность получить прямую ссылку на элемент DOM.

Ссылки будут храниться в глобальном объекте \$refs. Прямая ссылка дает возможность получить любую информацию об элементе и производить с ним любые манипуляции.

Это тот же результат, можно получить при использовании стандартных: **getElementById** или **querySelector**.

Доступ через this

Ссылки на элемент DOM шаблона внутри экземпляра реализуются через ключевое слово **this**.

example_11. Референции через привязку this

```
<div id="app">

  <div ref="refDiv">Элемент DIV</div>

  <p ref="refP">Элемент P</p>

  <input ref="refInp" type="text" value="Элемент INPUT"><br>

  <p><button @click="fnOutRefs () ">Вывести информацию по
    референциям</button>

  <p><button @click="fnUpdateRefs () ">Изменить
```

```
    референции</button>

</div>

<script>

    var vm = Vue.createApp({

        methods: {

            fnOutRefs: function() {

                //

                console.log(this.$refs.refDiv.innerText);

                console.log(this.$refs.refP.innerText);

                console.log(this.$refs.refInp.value);

                console.log(this.$refs);

            },

            fnUpdateRefs: function() {

                // изменяем свойство innerText референций

                this.$refs.refDiv.innerText = "Быстрее";

                this.$refs.refP.innerText = "Выше";

                this.$refs.refInp.value = "Сильнее";

            }

        }

    }).mount("#app");

</script>
```

Доступ через экземпляр компонента

Ссылки на элемент DOM шаблона за пределами экземпляра реализуются через имя экземпляра.

example_12. Референции через экзампляр

```
<div id="app">

  <div v-html="con"></div>

  <div ref="con"></div>

  <div id="con"></div>

</div>

<script>

  var vm = Vue.createApp({

    data() {

      return {

        con: ""

      }

    }

  }).mount("#app");

  vm.con = "<h3><a href='#'>Сырые данные на  
экземпляре</a></h3>";

  vm.$refs.con.innerHTML = "<h3><a href='#'>Референция на  
экземпляре</a></h3>";

  document.querySelector("#con").innerHTML = "<h3><a  
href='#'>Классический querySelector</a></h3>";

</script>
```

Использование манипуляций с DOM может быть весьма дорогостоящей операцией, по этой причине для управления приложением рекомендуется использовать существующие **интерфейсы Vue**.

Эмулируем поведение v-model через ref

Попробуем реализовать привязку через референцию.

example_13. V-model через референции

```
<div id="app">

  <h2>Текст: {{ title }}</h2>

  <input type="text" ref="myinput" @input="fInput()">

</div>

<script>

  var vm = Vue.createApp({

    data() {

      return {

        title: undefined

      }

    },

    methods: {

      fInput: function() {

        this.title = this.$refs.myinput.value;

      }

    }

  }).mount("#app");

</script>
```

Использование веб-хранилища

Для выполнения самостоятельных заданий этого и последующих уроков нам понадобятся знания об объекте веб-хранилища **localStorage** и, как следствие, объекте **JSON**. Они не имеют прямого отношения к фреймворку VueJS, но настолько важны в веб-разработке, что вспомним, что это такое. В теории и практике.



Объект JSON

JSON (JavaScript Object Notation) – это общий **строковый формат** для представления значений и объектов. Первоначально был создан для JavaScript, но многие другие языки также имеют библиотеки, для работы с форматом JSON.

Таким образом, JSON легко использовать для обмена данными, когда клиент использует JavaScript, а сервер написан на Ruby/PHP/Java или любом другом языке.

JavaScript предоставляет **методы**:

- **stringify** для преобразования JavaScript объектов в **строку формата JSON**.
- **parse** для преобразования JSON обратно в **JavaScript объект**.

Объект веб-хранилища localStorage

Объект веб-хранилища **localStorage** позволяет хранить пары ключ/значение в браузере. **Что при этом важно – данные, которые в нем записаны, сохраняются после обновления страницы и даже после перезапуска браузера.**

Объект хранилища **localStorage** предоставляет следующие **методы и свойства**:

- **setItem(key, value)** – сохранить пару ключ/значение.
- **getItem(key)** – получить данные по ключу key.
- **removeItem(key)** – удалить данные с ключом key.
- **clear()** – удалить всё.
- **key(index)** – получить ключ на заданной позиции.
- **length** – количество элементов в хранилище.

Хранилище привязано к источнику (домен/протокол/порт). Это значит, что разные протоколы или поддомены определяют **разные объекты хранилища**, и они не могут получить доступ к данным друг друга.

Рассмотрим все сказанное на примере кода **example_14**.

example_14. Используем веб-хранилища при работе с данными

```
<div id="app">

  <a href="#" @click.prevent="fGetUser()">Вывести объект user
  из localStorage</a>

  <div ref="box"></div>

</div>

<script>

  var vm = Vue.createApp({

    data() {

      return {

        user: user

      }

    },

    methods: {

      fGetUser: function() {

        let userJSON = localStorage.getItem("user");

        let user = JSON.parse(userJSON);

        // console.dir(user);

        let html = `

          <h2>${user.surname} ${user.name}
          ${user.patronymic}</h2>

          <p><b>Телефон: </b>${user.phone}</p>

          <p><b>Адрес: </b>${user.address}</p>

          <p><b>Образование: </b>${user.edu}</p>

          <p><b>Курс: </b>${user.courses}</p>

          <p><b>Email: </b>${user.email}</p>

        `;

      }

    }

  });
```

```
        this.$refs.box.innerHTML = html;

    },

    mounted() {

        this.user = JSON.stringify(this.user);

        // console.log(this.user);

        window.localStorage.setItem("user", this.user);

    }

  }).mount("#app");

</script>
```

Задача 1

В файле **task_1.html** раздаточного материала вам предложен шаблон с текстовыми элементами **input**, имеющими привязки к объекту **person** экземпляра Vue. Элементы расположены в блоке **div** с референцией **ref = "con"**. Напишите сценарий, который по клику на кнопку **Добавить данные** будет замещать содержимое блока референции данными созданного объекта **person**.

Задача 2

В файле **task_2.html** раздаточного материала вам предложен набор связанных элементов **input** с типом **type = "radio"** (**радиокнопки**). Напишите сценарий, в котором по клику будет загружаться соответствующее радиокнопке **изображение**. Соответствие определяется **названием** изображения и **идентификатором** радиокнопки.

Задача 3

В файле **task_3.html** раздаточного материала вам предложен файл с данными **user.js**. Сохраните данные в объект веб-хранилища **localStorage**. Перейдите по ссылке в файл **target.html**. В файле используя цикл **v-for** и **референцию** извлеките данные из объекта **localStorage** в окно браузера.

P.S. При выполнении задач допускается сверяться с предлагаемым решением. Важная поправка – сверяться, это не значит переписать. Это значит, посмотреть решение и повторить, или предложить свое.

P.S.

Для отработки и закрепления учебного курса **донам** группы предоставляется следующий раздаточный материал.

К каждому уроку курса:

- Файлы **демонстрационного кода** (example);
- **Задачи** с решениями в контексте рассматриваемых вопросов урока (task).

К каждой теме курса:

- **Практические работы**.