# Vue

## Подключение

<https://v3.ru.vuejs.org/ru/guide/installation.html#%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F-%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B8> -3 версия

**Подключение по** CDN

<script src="https://unpkg.com/vue@next"></script>

**Подключение по** CLI

npm install -g @vue/cli

Потом после установки

vue upgrade --next

**Создание объекта Vue**

const App = {

  data() {

    return {

      counter: 0

    }

  }

}

Vue.createApp(App);

Метод App - data - возвращает объект данных с начальным значением. Переменные здесь - модели

В HTML вставляем в нужном месте интерполяцию

{{ counter }}

Необходимо проинициализировать объект. Свяжем Vue с блоком app.

<div id="app"></div>

Vue.createApp(App).mount('#app');

## Начало работы, взаимодействие с HTML

**Значения данных** data в HTML приводятся к **строке**. Пустой объект {} приводится к строке {} с помощью метода JSON.stringify. undefined приводится к пустоте.

Также над переменными можно делать какие-либо операции: +, - и т.д.

Повесим на кнопки события прямо в HTML:

<button class="btn primary" v-on:click="counter++">+</button>

С помощью интерполяции мы не можем напрямую в атрибуты передавать значения. Нужно "забандить" атрибут и название переменной:

<input type="text" v-bind:placeholder="placeholderString"/>

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

<input type="text" v-bind:placeholder="placeholderString" v-bind:value="inputValue" v-on:input="inputChangeHandler" />

const App = {

**data**() {

    return {

      title: 'Список заметок',

      placeholderString: 'Введите название заметки',

**inputValue**: ''

    }

  },

  methods: {

    inputChangeHandler(event) {

*this*.inputValue = event.target.value;

    }

  }

}

Обратим внимание, что обращаемся к переменной из data через this. Аналогично через this можно обращаться и к методам.

## Циклы

Хотим вывести элементы массива notes: ['Заметка 1', 'Заметка 2'] (notes - переменная в data) в цикле. В HTML используем директиву v-for:

      <ul class="list">

        <li class="list-item" v-for="note in notes">{{ note }}</li>

      </ul>

Вместо in можно использовать of.

С использованием **индекса**:

<ul class="list">

<li class="list-item" v-for="(note,idx) of notes"> ({{ idx + 1 }}) {{ note }} </li>

<button class="btn danger" v-on:click="removeNote(idx, $event)">Удалить</button>

</ul>

Здесь можем использовать $event, если нужен event

  methods: {

    inputChangeHandler(event) {

*this*.inputValue = event.target.value;

    },

    addNewNote() {

*this*.notes.push(*this*.inputValue);

*this*.inputValue = '';

    },

**removeNote**(idx, event) {

*this*.notes.splice(idx, 1);

    }

  }

## Условие

Хотим вывести элементы только если в массиве что-то есть.

<ul class="list" v-if="notes.length !== 0">

<li class="list-item" v-for="(note,idx) of notes">

({{ idx + 1 }}) {{ note }}

<button class="btn danger" v-on:click="removeNote(idx, $event)">Удалить</button>

</li>

</ul>

<div v-if="notes.length === 0">Заметок пока нет</div>

Или по-другому:

<div v-else>Заметок пока нет</div>

А также

div v-else-if="notes.length === 0">Заметок пока нет</div>

v-else-if обязательно должен идти после v-if

## Модификаторы

Хотим вывести заметки после нажатия на клавишу Enter в инпуте:

<input

type="text"

v-bind:placeholder="placeholderString"

v-bind:value="inputValue"

v-on:input="inputChangeHandler"

v-on:keypress.**enter**="addNewNote"

/>

В JS:

const App = {

data() {

return {

title: 'Список заметок',

placeholderString: 'Введите название заметки',

inputValue: '',

notes: ['Заметка 1', 'Заметка 2']

}

},

methods: {

inputChangeHandler(event) {

this.inputValue = event.target.value;

},

**addNewNote()** {

this.notes.push(this.inputValue);

this.inputValue = '';

},

}

}

## Методы, возыращающие строку в интерполяции

В интерполяции можем также использовать методы, возвращающие строку. Например, хотим заметки сделать в верхнем регистре.

<li class="list-item" v-for="(note,idx) of notes">

({{ idx + 1 }}) {{ toUpperCase(note) }}

</li>

В JS:

  methods: {

    toUpperCase(item) {

      return item.toUpperCase();

    }

  }

## Альтернативный синтаксис для директив

Вместо **v-bind** и **v-on** можно использовать : и @ Например:

        <input type="text" :placeholder="placeholderString" @input="inputChangeHandler"  />

## computed

В объекте App можем использовать также объект computed:

computed: {

**doubleCountComputed**() {

console.log('doubleCountComputed');

return this.notes.length \* 2;

}

}

Работает с переменными, должен что-то возвращать.

В HTML функции из него не вызываем!:

<strong>Общее количество: {{ notes.length}}</strong> | Удвоено: {{ **doubleCountComputed**}}

Методы из computed используются для оптимизации кода. Они будут вызываться только тогда, когда изменяются изменяемые в них переменные, а не каждый раз, например, когда меняется input

## watch

В объекте App можем использовать также объект watch:. С его помощью можем смотреть за переменными. Для этого нужно в нем создать метод, совпадающий с названием переменных. Также можно при этом сделать валидацию.

watch: {

**inputValue(value)** {

if(value.length > 10) {

this.inputValue = '';

}

console.log('input value :', value);

}

}

## v-model - двустороннее связывание js и html

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

data() {

return { inputValue: '', }

},

<input type="text" v-model="inputValue" />

## Динамическое изменение стилей и классов с условиями

<h1 :style="**{**

**color: inputValue.length < 5 ? 'darkred' : 'darkblue',**

**fontSize: inputValue.length < 6 ? '2rem' : '1.5rem'**

**}**">{{ title }}</h1>

Для класса:

<span :class="**note.length > 5 ? 'primary' : 'bold'**">({{ idx + 1 }}) {{ toUpperCase(note) }}</span>

Другой способ:

<span :class**="{**

**'primary': true,**

**'bold': note.length > 5**

**}**">({{ idx + 1 }}) {{ toUpperCase(note) }}</span>

Еще способ:

<span :class="**['bold', {'primary' : note.length > 5}]**">({{ idx + 1 }}) {{ toUpperCase(note) }}</span>

## II часть

## Директивы

<https://v3.ru.vuejs.org/ru/api/directives.html>

v-text - альтернатива интерполяции // Заголовок

<h1 v-text="title"></h1>

data: () => ({

title: 'Заголовок'

}),

v-once - Отрисовывает один раз элемент или компонент. При последующих обновлениях данных и перерисовках элемент/компонент и все его потомки будут считаться статичными и пропускаться.

<h2 v-text="title"></h2>

<h2 v-once>{{title}}</h2>

<button class="btn" @click="title = 'Теперь заголовок 2'">Изменить название</button>

При нажатии на кнопку название изменится лишь один раз // Теперь заголовок 2 Заголовок

v-pre - вывести как есть //{{ title }}

<h2 v-pre>{{ title }}</h2>

v-html - вывести как HTML // Vue 3 App

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

data: () => ({

myHtml: '<h1>Vue 3 App</h1>',

title: 'Заголовок'

}),

v-cloak - используется, чтобы не рисовать элементы, пока Vue не отработал (чтобы изображение не дергалось)

**<style>**

**[v-cloak] {**

**display: none;**

**}**

</style>

</head>

<body>

<div class="container pt-5" id="app" v-cloak>

## Циклы, условия еще раз - примеры

С индексом i :

<ul class="list">

<li class="list-item" v-for=" (item, i) in 7"> {{ i }} {{ item }}</li> // 01 12 23 34 45 56 67

</ul>

Еще пример

<li class="list-item" v-for="value in person">{{ value }}</li>

data: () => ({

person: {

firstName: 'Vkadilen',

lastName: 'Minin',

age: 27

}

}), // Vladilen Minin 27

Еще пример - удалить по клику элементы из массива. Если элементов нет - вывести сообщение

<ul class="list" **v-if="items.length**">

<li class="list-item" **v-for="(item, i) in items**" **@click="items.splice(i, 1)**"><strong>{{ item }}</strong></li>

<!-- <li class="list-item" v-for="(item, i) in 7">{{ i }}{{ item }}</li> -->

<!-- <li class="list-item" v-for="value in person">{{ value }}</li> -->

</ul>

<h3 **v-else**>Элементов нет</h3>

data: () => ({

items: [1, 2, 3, 4, 5, 6]

}),

В этом случае, если элементы есть, сообщения нет в дереве. Если нам нужно, чтобы оно было (с display: none;), нужно использовать директиву v-show

<h3 v-show="items.length === 0">Элементов нет</h3>

## Модификаторы событий - примеры

<https://v3.ru.vuejs.org/ru/guide/events.html#%D0%BC%D0%BE%D0%B4%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82%D0%BE%D1%80%D1%8B-%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B8>

Модификатор stop - event.stopPropagation() – прекращение всплытия (вертикального). Метод при этом писать не надо.

<input type="text" @click.stop>

Модификатор prevent - event. preventDefault() – отмена действия браузера по умолчанию

<input type="text" @click.prevent>

Можно использовать оба модификатора сразу

<input type="text" @click.prevent.stop>

.capture

можно отслеживать события в режиме capture, т.е. событие, нацеленное

на внутренний элемент, обрабатывается здесь до обработки этим элементом

<div @click.capture="doThis">...</div>

.self

вызов обработчика только в случае наступления события непосредственно

на данном элементе (то есть не на дочернем компоненте)

<div @click.self="doThat">...</div>

.once

обработчик click будет вызван максимум 1 раз

<a @click.once="doThis"></a>

.passive

соответствующий опции passive для addEventListener (opens new window).

По умолчанию событие scroll (при прокрутке) произойдёт незамедлительно, вместо ожидания окончания `onScroll` на случай, если там будет `event.preventDefault()`

<div @scroll.passive="onScroll">...</div>

Модификатор .passive особенно полезен для улучшения производительности на мобильных устройствах.

Вспомним options для обработчика событий:

**Options** - Дополнительный объект со свойствами:

* **once**: если true, тогда обработчик будет автоматически удалён после выполнения.
* **capture**: фаза, на которой должен сработать обработчик. Так исторически сложилось, что options может быть false/true, это то же самое, что {capture: false/true}. Если аргумент **false** (по умолчанию), то событие будет поймано при всплытии
* **passive**: если true, то указывает, что обработчик никогда не вызовет preventDefault

## Баг с циклами

Введем в input какой-либо текст. При удалении соответствующего элемента списка текст не удалится, а будет отображаться в следующем элементе

<ul class="list" v-if="items.length">

<li class="list-item" v-for="(item, i) in items" @click="items.splice(i, 1)">

<strong>{{ item }}</strong>&nbsp;

<input type="text" @click.prevent.stop>

</li>

</ul>

Чтобы это исправить, используемдля цикла специальный атрибут key и в нем уникальное значение item:

<li class="list-item" v-for="(item, i) in items" @click="items.splice(i, 1)" v-bind:key="item">

## Фильтрация элементов в циклах

Хотим вывести только четные элементы цикла.

<li class="list-item" v-for="(item, i) in evenItems" @click="evenItems.splice(i, 1)" v-bind:key="item">

computed: {

evenItems() {

return this.items.filter( i => i % 2 === 0)

}

},

## Модификаторы клавиш и использование ссылки - пример

Хотим при нажатии Enter добавить элемент в массив

      <div class="form-control">

        <input type="text" @keyup.enter="addItem" ref="myInput">

      </div>

data: () => ({

items: [1, 2]

}),

methods: {

addItem() {

this.items.unshift(this.$refs.myInput.value);

}

},