# Тема 13. JS-фреймворк Vue.js

# 1. Введение в Vue.js.

# 2. Объект Vue и привязка данных.

# 3. Обработка событий.

# 4. Вычисляемые свойства.

# 5. Работа с классами и стилями.

# 6. Работа с объектом Vue и управление html-элементами.

# 7. Жизненный цикл Vue.

# 8. Условный рендеринг.

# 9. Работа с массивами.

# 10. Работа с формами.

Содержание данной темы включает материалы, доступные по адресу https://metanit.com.

* 1. 1. Введение в Vue.js

Vue.js представляет современный прогрессивный фреймворк, написанный на языке JavaScript и предназначенный для создания веб-приложений клиентского уровня. Основная сфера применения данного фреймворка – это создание и организация пользовательского интерфейса.

Первый релиз фреймворка увидел свет в феврале 2014 года. Его создателем является Эван Ю (Evan You), который до этого работал в Google над AngularJS. С тех пор фреймфорк динамично развивается, на момент составления методического комплекса его текущей версией является версия 2.6.11.

Официальный сайт фреймворка - <https://vuejs.org/>, где можно найти непосредственно файлы фреймворка, а также сопроводительные материалы и документацию. Кроме того, сам проект доступен на github по адресу <https://github.com/vuejs/vue>.

Vue.js имеет довольно небольшой размер – не более 20 кБ, и при этом обладает хорошей производительностью по сравнению с такими фреймворками как Angular или React. Поэтому данный фреймворк в последнее время набирает обороты и становится все более популярным.

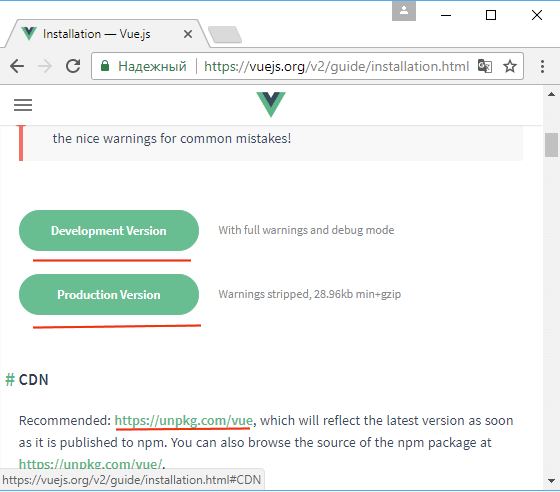
Одним из ключевых моментов в работе Vue.js является виртуальный DOM. Структура веб-страницы, как правило, описывается с помощью DOM (Document Object Model), которая представляет организацию элементов html на странице. Для взаимодействия с DOM (добавления, изменения, удаления html-элементов) применяется JavaScript. Но если манипулировать html-элементами с помощью JavaScript, то можно столкнуться со снижением производительности, особенно при изменении большого количества элементов. А операции над элементами могут занять некоторое время, что неизбежно скажется на пользовательском опыте. Однако если работать из кода js с объектами JavaScript, то операции будут производиться быстрее.

Для этого Vue.js использует виртуальный DOM. Виртуальный DOM представляет легковесную копию обычного DOM. Если приложению нужно узнать информацию о состоянии элементов, то происходит обращение к виртуальному DOM. Если данные, которые используются в приложении Vue.js, изменяются, то изменения вначале вносятся в виртуальный DOM. Потом Vue выбирает минимальный набор компонентов, для которых надо выполнить изменения на веб-странице, чтобы реальный DOM соответствовал виртуальному. Благодаря виртуальному DOM повышается производительность приложения.

Vue.js поддерживается всеми браузерами, которые совместимы с ECMAScript 5. На данный момент это все современные браузеры, в том числе IE11.

**Первое приложение**

Создадим первое приложение на Vue.js. Прежде всего надо подключить файлы фреймворка на веб-страницу. Все необходимые материалы для загрузки можно найти по адресу <https://vuejs.org/v2/guide/installation.html>. С этой страницы можно загрузить файл фреймворка локально (доступен в двух вариантах: Production и Development).



Вместо использования локального файла можно загружать фреймворк из CDN по ссылке: <https://unpkg.com/vue>. Либо если используется Node.js, то можно установить Vue.js через пакетный менеджер npm с помощью команды:

- npm install vue

В данном случае будем использовать подключение файла из CDN. Для этого определим следующую веб-страницу:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <input type="text" v-on:input="setMessage" />

    <p>{{message}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            message: 'Hello Vue!'

        },

        methods: {

            setMessage: function(event){

                this.message = event.target.value;

            }

        }

    })

</script>

</body>

</html>

Для создания объекта приложения в Vue.js применяется объект Vue. Этот объект, во-первых, определяет корневой элемент приложения на веб-странице с помощью параметра el:

- el: '#app'

То есть корневым элементом приложения будет элемент с id равным app.

Также объект определяет используемые данные через параметр data:

- data: {

    message: 'Hello Vue!'

}

В данном случае определено только свойство message, которая хранит строку. В элементе с id=app на веб-странице, используя двойные фигурные скобки можyj вывести значение свойства message и связать участок веб-станицы с этим элементом.

- <p>{{message}}</p>

Последний параметр объекта Vue – methods определяет действия, которые выполняются в приложении:

methods: {

    setMessage: function(event){

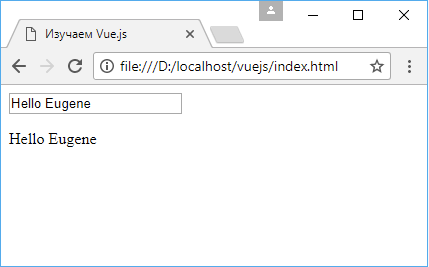
        this.message = event.target.value;

    }

}

Здесь определен метод setMessage, который является обработчиком события ввода для элемента input на странице. Через параметр event в этот метод передается информация о событии. В частности, используя значение event.target.value, можно получить введенное пользователем значение и присвоить его переменной message. Для получения переменной message, которая была определена выше в параметре data, применяется ключевое слово this. А для связи элемента input с этим методом определяется атрибут v-on:input="setMessage".

Откроем веб-страницу в веб-браузере:



При вводе в текстовое поле будет автоматически изменяться значение, которое выводится на веб-страницу.

* 1. 2. Объект Vue и привязка данных

Для создания приложения в Vue.js используется объект Vue. При создании этого объекта в его конструктор передается параметр, с помощью которых производится инициализация приложения:

let vm = new Vue({

  // параметры

})

Касательно кода html приложение Vue.js представляет некоторую область на веб-странице. Для определения этой области на веб-станице можно определить какой-либо элемент. А для связи с объектом Vue в нем определяется параметр el:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

</div>

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

<script>

    let app = new Vue({

        el: '#app'

    })

</script>

</body>

</html>

С помощью параметра data можно определить данные, которые должно хранить приложение, то есть его состояние:

let app = new Vue({

    el: '#app',

    data: {

        name: 'Tom',

        age: 25

    }

})

Здесь через параметр data определяются свойства name и age. Данные можно определять в виде внешнего объекта и затем передавать параметру data:

let userData = {name: 'Tom', age:25};

let app = new Vue({

    el: '#app',

    data: userData

});

app.age=30;

app.name='Bob';

Кроме того, используя объект Vue, можно обращаться к его свойствам по имени, получать и присваивать им значение: app.age=30.

**Методы**

Кроме хранения состояния Vue может определять поведение в виде параметра methods. Этот параметр указывает на набор методов. Каждый метод представляет стандартную функцию JavaScript. Например, определим ряд методов:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <p>{{name}}</p>

    <p>{{welcome()}}</p>

    <p>Name: {{displayName()}}</p>

    <p>Factorial of 6 is equal to {{factorial(6)}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data:{name: 'Tom', age:25},

        methods:{

            welcome: function(){

                return "Welcome";

            },

            displayName: function() {

                return this.name;

            },

            factorial:function(n){

                let result=1;

                for(let i=1; i<=n;i++){

                    result = result \* i;

                }

                return result;

            }

        }

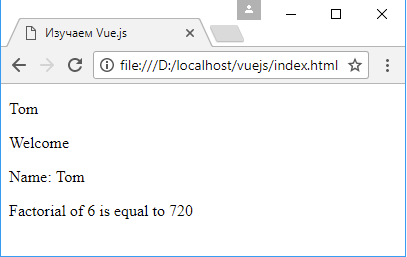
    })

</script>

</body>

</html>

Здесь определено три метода. Первый метод welcome просто возвращает некоторое значение. Второй метод возвращает значение свойства name. Для обращения к свойствам объекта Vue в методах используется ключевое слово this. Третий метод принимает параметр – число и вычисляет для него факториал.



**Привязка данных**

Vue.js позволяет декларативным образом устанавливать привязку между элементами веб-страницы и данными объекта Vue. Есть различные формы привязки.

*Интерполяция.* Простейшую форму привязки представляет интерполяция строк. В этом случае значение, к которому выполняется привязка, заключается в двойные фигурные скобки:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <p>{{name}} - {{age}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {name: 'Tom', age:25}

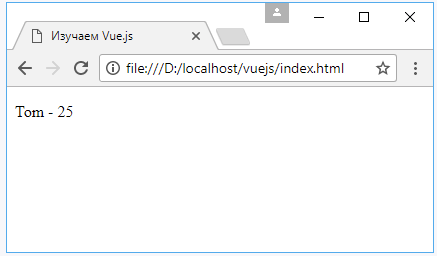
    });

</script>

</body>

</html>

При рендеринге выражения со скобками {{}} будут заменяться соответствующими значениями:



При этом в интерполяции могут использоваться любые валидные выражения JavaScript.

<div id="app">

    <p>{{age > 25?'Больше 25 лет':'25 лет или меньше' }}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {name: 'Tom', age:99}

    });

</script>

*Привязка к атрибутам.* Для привязки к атрибутам html-элементов предназначена директива v-bind:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <a v-bind:href="link">{{text}}</a>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {text: 'Google', link:'https://google.com'}

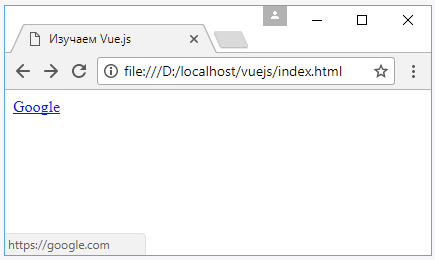
    });

</script>

</body>

</html>

Для привязки к атрибуту после v-bind через двоеточие указывается непосредственно сам атрибут, к которому надо выполнять привязку.



Для привязки атрибутов также можно использовать сокращенную форму:

<a :href="link">{{text}}</a>

*Однократная привязка.* Если необходимо, чтобы значение было привязано к элементу html только один раз и впоследствии не изменялось, то применяется директива v-once:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <input type="text" v-on:input="setMessage" />

    <p v-once>{{message}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {message:'Hello'},

        methods: {

            setMessage: function(event){

                this.message = event.target.value;

            }

        }

    });

</script>

</body>

</html>

*Привязка к html.* Для привязки элемента к коду html применяется директива v-html:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <div v-html="message"></div>

    <div>{{message}}</div>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {message:'<h2>Hello</h2>'}

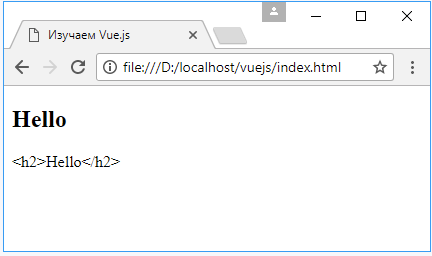
    });

</script>

</body>

</html>

С помощью директивы v-html привязываемое значение будет рассматриваться как код html. В то же время при простой интерполяции код html будет интерпретироваться как обычная строка:



**Шаблоны**

Для работы с веб-страницей vue.js использует шаблоны. Шаблоны в Vue.js представляют валидную разметку с кодом html, которая компилируется в функции рендеринга виртуального DOM. Внутри шаблона может использоваться состояние приложения, то есть свойства объекта Vue, определенные через параметр data. Возьмем следующую страницу:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <p>Имя: {{name}}   Возраст {{age}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {name:'Tom', age:25}

    });

</script>

</body>

</html>

Здесь все, что внутри элемента <div id="app">, представляет шаблон Vue. То есть в данном случае шаблоном является код:

<p>Имя: {{name}}   Возраст {{age}}</p>

Vue.js управляет именно этой частью веб-страницы. Вне шаблона Vue не работает. Также шаблон можно задать через параметр template в объекте Vue:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {name:'Tom', age:25},

        template: '<p>Имя: {{name}}   Возраст {{age}}</p>'

    });

</script>

</body>

</html>

В данном случае результат будет тот же самый.

* 1. 3. Обработка событий

Для обработки событий элементов html в Vue.js используется директива v-on, после которой через двоеточие указывается тип события:

v-on:click="действия"

В качестве типа события используется любое стандартное событие элементов на веб-странице. Затем этой директиве в кавычках передаются те действия, которые должны выполниться при возникновении события. Например:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

    <button v-on:click="if(counter>0) counter--;">-</button>

    <div>{{counter}}</div>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {counter:0},

    });

</script>

</body>

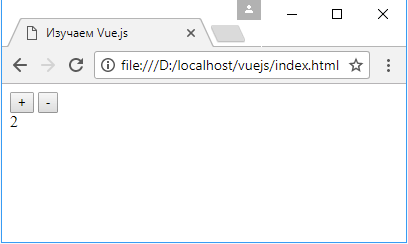
</html>

В качестве действия директиве v-on можно передать любые валидные операции JavaScript. В данном случае, так как определено во Vue свойство counter, то можно манипулировать значением этого свойства:

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

<button v-on:click="if(counter>0) counter--;">-</button>

Фактически директивам v-on передается кусок кода JavaScript, где свойства из Vue доступны как обычные переменные.



Также можно использовать сокращенный синтаксис для определения обработчиков событий:

<button @click="counter++">+</button>

<button @click="if(counter>0) counter--;">-</button>

Однако в случае более сложных операций такой подход не является оптимальным. И в этом случае, как правило, все операции, которые надо вызывать при возникновении события, выносятся в отдельные методы объекта Vue. Например, перепишем предыдущий пример:

<div id="app">

    <button v-on:click="increase">+</button>

    <button v-on:click="decrease">-</button>

    <div>{{counter}}</div>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {counter:0},

        methods:{

            increase: function(){

                this.counter++;

            },

            decrease: function(){

                if(this.counter>0)

                    this.counter--;

            }

        }

    });

</script>

При нажатии на одну кнопку срабатывает метод increase, который увеличивает значение переменной counter. При нажатии на вторую кнопку вызывается метод decrease, который уменьшает это значение. При генерации события в его обработчике в качестве параметра можно получить объект, который инкапсулирует всю информацию о событии:

<div id="app">

    <button v-on:click="increase">+</button>

    <button v-on:click="decrease">-</button>

    <div>{{counter}}</div>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {counter:0},

        methods:{

            increase: function(event){

                console.log(event);

                this.counter++;

            },

            decrease: function(event){

                console.log(event);

                this.counter--;

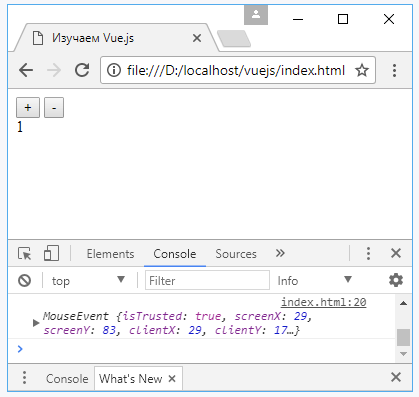
            }

        }

    });

</script>

В качестве параметра в методы-обработчики события передается объект события. Это стандартный объект, который используется при обработке событий в обычном JavaScript. В зависимости от типа события этот объект может нести разную информацию.



Кроме того, можно передать в методы для обработки событий какие-то дополнительные значения:

<div id="app">

    <button v-on:click="increase(3)">+</button>

    <button v-on:click="decrease(2)">-</button>

    <div>{{counter}}</div>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {counter:0},

        methods:{

            increase: function(n){

                this.counter = this.counter + n;

            },

            decrease: function(n){

                this.counter = this.counter - n;

            }

        }

    });

</script>

В данном случае увеличение переменной counter будет идти на 3, а уменьшение – на 2.

Также можно совместить передачу значений и объекта события:

<div id="app">

    <button v-on:click="increase(3, $event)">+</button>

    <button v-on:click="decrease(2, $event)">-</button>

    <div>{{counter}}</div>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {counter:0},

        methods:{

            increase: function(n, event){

                console.log(event)

                this.counter = this.counter + n;

            },

            decrease: function(n, event){

                this.counter = this.counter - n;

            }

        }

    });

</script>

Для передачи объекта события в обработчик передается специальный объект $event.

**Двусторонняя привязка**

Ранее были рассмотрены примеры использования односторонней привязки с помощью интерполяции. Но кроме того, Vue.js поддерживает и двустороннюю привязку. Для создания подобной привязки используется директива v-model. Например:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <input type="text" v-model="name" /><br><br>

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

    <p>Name: {{name}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {name: 'Tom', age:99}

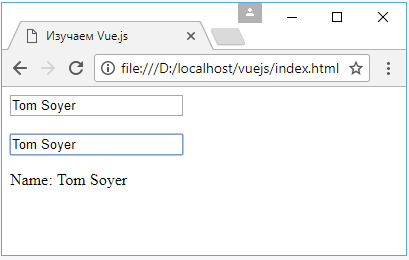
    });

</script>

</body>

</html>

Здесь на странице определено два текстовых поля, и оба они привязаны к свойству name из объекта Vue. В этом случае не надо добавлять к текстовому полю обработчик события ввода, и в этом обработчике менять вручную значение свойства name. При изменении текста в одном из полей автоматически изменится значение в другом. То есть сработает двусторонняя привязка.



Другой пример – определим программу вычисления факториала:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

    <p>Факториал числа {{number}} равен {{factorial(number)}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {number:1},

        methods:{

            factorial: function(n){

                let result = 1;

                for(let i=1;i<=n;i++)

                    result \*=i;

                return result;

            }

        }

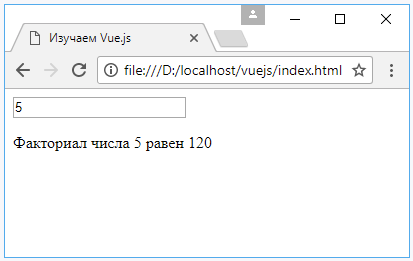
    });

</script>

</body>

</html>

При вводе числа в текстовое поле функция factorial автоматически будет вычислять значение заново.



* 1. 4. Вычисляемые свойства

Кроме обычных свойств объект Vue может содержать вычисляемые свойства, который во многом аналогичны функциям, но в то же время отличаются от них. Рассмотрим небольшой пример:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

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

    <p>Имя: {{name}}   Возраст {{age}}</p>

    <p>{{checkAge()}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {name:'Tom', age:25},

        methods:{

            checkAge: function(){

                console.log("method");

                if(this.age > 17)

                    return "доступ разрешен";

                else

                    return "доступ запрещен";

            }

        }

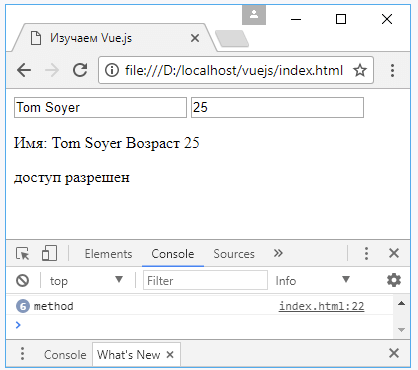
    });

</script>

</body>

</html>

Здесь в зависимости от возраста пользователя функция checkAge возвращает некоторый результат. И при каждом изменении значения свойства age, функция checkAge будет пересчитывать свой результат. Однако минусом подобного подхода является то, что метод checkAge будет выполняться при изменении любого свойства во Vue, в том числе свойства name, которое с методом checkAge никак не связано. Это можно увидеть по консольному выводу из метода checkAge при изменении свойства name:



Более оптимальным в данном случае будет использование вычисляемых свойств, которые определяются с помощью параметра computed:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

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

    <p>Имя: {{name}}   Возраст {{age}}</p>

    <p>{{checkAge()}}</p>

    <p>{{enabled}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {name:'Tom', age:25},

        computed:{

            enabled: function(){

                console.log("computed");

                if(this.age > 17)

                    return "доступ разрешен";

                else

                    return "доступ запрещен";

            }

        },

        methods:{

            checkAge: function(){

                console.log("method");

                if(this.age > 17)

                    return "доступ разрешен";

                else

                    return "доступ запрещен";

            }

        }

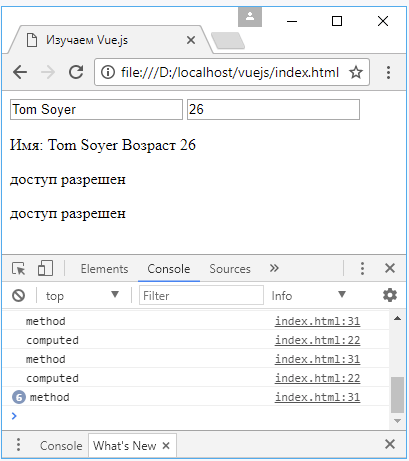
    });

</script>

</body>

</html>

Здесь enabled представляет вычисляемое свойство. Его определение во многом аналогично методу checkAge. Но теперь при изменении состояния Vue будут анализироваться сделанные изменения, и если потребуется, то свойство enabled будет повторно вычисляться. Поэтому в данном случае, если изменить свойство age, то изменится свойство enabled и повторно выполнится функция checkAge. Но если изменится свойство name, то свойство enabled не будет изменяться:



**Сеттеры**

Вычисляемое свойство можно разделить на сеттер и геттер. Геттер возвращает значение, а сеттер устанавливает. По умолчанию свойство имеет только геттер – во всех примерах выше вычисляемое свойство представляет функцию, которая возвращает некоторое значение. Теперь определим и геттер, и сеттер:

<div id="app">

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

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

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

    <p>Имя: {{fullname}}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {firstname:'Tom', lastname:'Smith'},

        computed:{

            fullname: {

                get: function () {

                    return this.firstname + ' ' + this.lastname;

                },

                set: function (newValue) {

                    let names = newValue.split(' ')

                    this.firstname = names[0]

                    this.lastname = names[names.length - 1]

                }

            }

        }

    });

</script>

На уровне кода геттер задается через параметр get, который представляет функцию, возвращающую значение. А сеттер представляет функцию, задаваемую через параметр set. Причем каждое новое значение передается через параметр этой функции (в примере выше параметр newValue). И так как в данном случае вычисляемое свойство представляет объединение простых свойств firstname и lastname, то в сеттере можно получить по отдельности значения этих свойств после изменения свойства fullname.

* 1. 5. Работа с классами и стилями

С помощью привязки атрибута class во Vue.js можно динамически управлять классами элементов. Для условной привязки классов определим следующую страницу:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

<style>

.region{

    background-color: #999;

    width: 100px;

    height: 100px;

    display: inline-block;

    margin: 10px;

}

.active{

    background-color:red;

}

</style>

</head>

<body>

<div id="app">

    <div class="region" v-bind:class="{active: isActive}" v-on:click="isActive=!isActive"></div>

    <div class="region"></div>

</div>

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

<script>

    let app = new Vue({

        el: "#app",

        data: {isActive : false}

    });

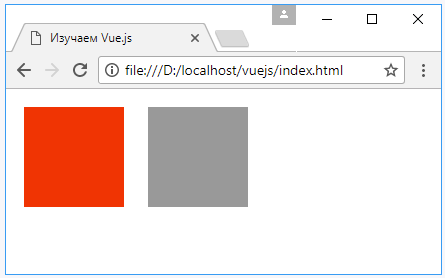
</script>

</body>

</html>

Здесь определены два блока с классом region. Кроме того, у первого блока применяется условная привязка классов. Она имеет форму: {класс\_1: true|false, класс\_2: true|false, класс\_N: true|false}

Если для класса определено значение true, то класс применяется. В данном случае применение класса зависит от свойства isActive, которое определено в объекте Vue. А с помощью события click у первого блока можно переключить значение свойства isActive, и, следовательно, включить или отключить класс active. При этом не столь важно, определены у элемента какие-либо еще классы, все они вместе будут применяться к элементу.



Подобным образом можно привязать несколько классов:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

<style>

.region{

    background-color: #999;

    width: 100px;

    height: 100px;

    display: inline-block;

    margin: 10px;

}

.active{

    background-color:red;

}

.bounded{

    border: 2px solid green;

}

</style>

</head>

<body>

<div id="app">

    <div class="region" v-bind:class="{active: isActive, bounded: isBounded}"></div>

    <div class="region"></div>

    <button v-on:click="isActive=!isActive">Toggle color</button>

    <button v-on:click="isBounded=!isBounded">Toggle border</button>

</div>

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

<script>

    let app = new Vue({

        el: "#app",

        data: {isActive : false, isBounded: false}

    });

</script>

</body>

</html>

Если название класса имеет какие-либо неалфавитно-цифровые символы, то оно помещается в кавычки:

v-bind:class="{'active-color': isActive, bounded: isBounded}"

Если необходимо устанавливать много классов, то для них можно определить один вычисляемый объект:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

<style>

.region{

    background-color: #999;

    width: 100px;

    height: 100px;

    display: inline-block;

    margin: 10px;

}

.active{

    background-color:red;

}

.bounded{

    border: 2px solid green;

}

</style>

</head>

<body>

<div id="app">

    <div class="region" v-bind:class="classObj"></div>

    <div class="region"></div>

    <button v-on:click="isActive=!isActive">Toggle color</button>

    <button v-on:click="isBounded=!isBounded">Toggle border</button>

</div>

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

<script>

    let app = new Vue({

        el: "#app",

        data: {

            isActive : false,

            isBounded: false

        },

        computed: {

            classObj: function () {

                return {

                    active: this.isActive, bounded: this.isBounded

                }

            }

        }

    });

</script>

</body>

</html>

Также можно выполнить простую привязку к свойству, которое хранит название класса:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

<style>

.region{

    background-color: #999;

    width: 100px;

    height: 100px;

    display: inline-block;

    margin: 10px;

}

.yellow{

    background-color: yellow;

}

.red{

    background-color: red;

}

.blue{

    background-color: blue;

}

.green{

    background-color: green;

}

</style>

</head>

<body>

<div id="app">

    <div class="region" v-bind:class="color"></div>

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

</div>

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

<script>

    let app = new Vue({

        el: "#app",

        data: {

            color: ""

        }

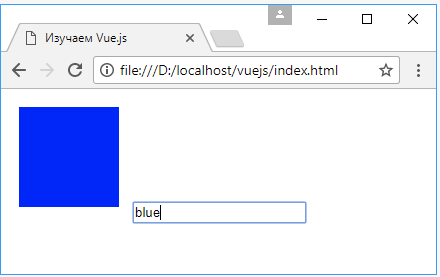
    });

</script>

</body>

</html>

В данном случае привязка атрибута class будет идти к свойству color. С помощью ввода в текстовое поле можно изменить через двустороннюю привязку значение этого свойства и соответственно поменять класс:



Если надо установить привязку сразу к нескольким классом, то их можно перечислить в квадратных скобках:

<style>

.region{

    background-color: #999;

    margin: 10px;

    width: 75px;

    height: 75px;

}

.bounded{

    border: 2px solid black;

}

.yellow{

    background-color: yellow;

}

.red{

    background-color: red;

}

.blue{

    background-color: blue;

}

.green{

    background-color: green;

}

.big{

    width: 100px;

    height: 100px;

}

.small{

    width: 50px;

    height: 50px;

}

</style>

<div id="app">

    <div class="region" v-bind:class="[color, {bounded: isBounded}, size]" v-on:click="isBounded=!isBounded"></div>

    <br>

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

    <br>

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

</div>

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

<script>

    let app = new Vue({

        el: "#app",

        data: {

            color: "red",

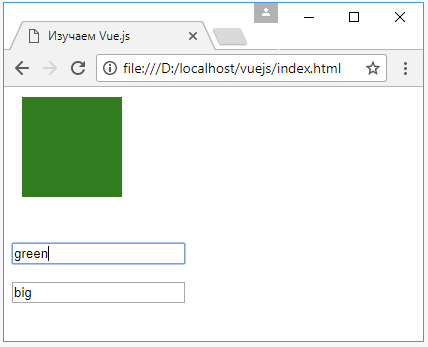
            size: "small",

            isBounded: false

        }

    });

</script>



**Привязка к стилям**

Кроме классов Vue.js позволяет с помощью привязки атрибутов управлять стилями html-элемента. Для привязки к стилям атрибуту style можно передать объект, который содержит стилевые свойства и их значения:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <div class="region" v-bind:style="{'background-color': color, height: height + 'px', width: width+'px'}"></div>

    <br><br>

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

    <br><br>

    <input type="number" v-model="width" />

    <br><br>

    <input type="number" v-model="height" />

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            color: 'red',

            width: 75,

            height: 75

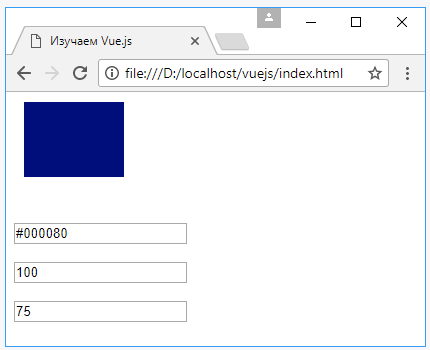
        }

    });

</script>

</body>

</html>

Для установки стилей атрибуту style предаются названия свойств css и их значения. При этом в качестве значения могут выступать как обычные значения стилевых свойств, так и свойства из объекта Vue, через которые можно манипулировать стилями. Стоит учитывать, что если свойство css в своем названии имеет дефисы, то название свойства заключается в кавычки, как в случае со свойством background-color.

Если необходимо устанавливать много различных стилевых свойств, то их можно вынести в отдельное вычисляемое свойство:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <div class="region" v-bind:style="styleObj"></div>

    <br><br>

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

    <br><br>

    <input type="number" v-model="width" />

    <br><br>

    <input type="number" v-model="height" />

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            color: 'red',

            width: 75,

            height: 75

        },

        computed:{

            styleObj: function(){

                return {

                    'background-color': this.color,

                    height: this.height + 'px',

                    width: this.width+'px'

                }

            }

        }

    });

</script>

</body>

</html>

Фактически каждый такой объект представляет отдельный стиль.

Если необходимо установить для элемента сразу несколько подобных стилей, то они передаются в квадратные скобки:

<div id="app">

    <div class="region" v-bind:style="[colorStyle, sizeStyle]"></div>

    <br><br>

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

    <br><br>

    <input type="number" v-model="width" />

    <br><br>

    <input type="number" v-model="height" />

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            color: 'red',

            width: 75,

            height: 75

        },

        computed:{

            colorStyle: function(){

                return {

                    'background-color': this.color

                }

            },

            sizeStyle: function(){

                return {

                    height: this.height + 'px',

                    width: this.width+'px'

                }

            }

        }

    });

</script>

**Наблюдаемые свойства**

Кроме вычисляемых свойств Vue.js позволяет определять наблюдаемые свойства или watchers. Наблюдаемые свойства, как правило, применяются для выполнения асинхронных действий, особенно таких, которые могут занять продолжительное время, например, отправка запроса на сервер.

Например, определим следующую веб-страницу:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <p>Введите число:<input v-model="number"></p>

    <p>{{ result }}</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data:{

            number: '',

            result: ''

        },

        watch: {

            number: function (newNumber) {

                if(newNumber>0){

                    this.factorial(newNumber);

                }

            }

        },

        methods:{

            factorial: function(newNumber){

                this.result = 'Идет вычисление факториала...';

                let vm = this;

                setTimeout(function(){

                    let res = 1;

                        for(let i = 1; i<=newNumber; i++){

                            res = res \* i;

                        }

                        vm.result = 'Факториал числа ' + newNumber + ' равен ' + res;

                }, 2000);

            }

        }

    });

</script>

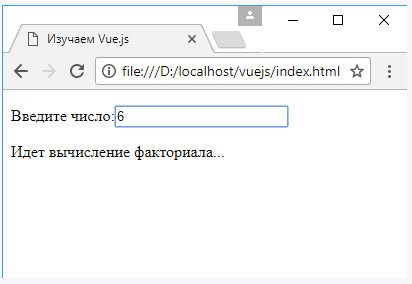
</body>

</html>

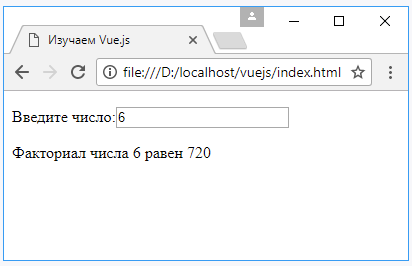
Для создания наблюдаемого свойства используется параметр watch. В данном случае определяется наблюдаемое свойство number, которое указывает на функцию. Причем в параметре data также определяется свойство number. При изменении свойства number, которое определено в секции data, будет вызываться функция number, определенная в параметре watch. Измененное значение будет передаваться в функцию через параметр newNumber.

Функция number в зависимости от нового значения свойства number вызывает метод factorial. В этом методе устанавливается значение свойства result. Причем вначале устанавливается некоторое промежуточное значение. Затем для имитации продолжительной операции применяется функция setTimeout(), которая выполняет задержку на 2 секунды, после которой вычисляет факториал числа. И в конце вычисленный результат передается свойству result.

Поэтому при изменении числа в текстовом поле вначале увидим временную надпись:



И затем и собственно результат - факториал числа:



* 1. 6. Работа с объектом Vue и управление html-элементами.

Объект Vue является стандартным объектом JavaScript, с которым можно работать из остального кода на js, в том числе и в других объектах Vue. Например, определим следующую страницу:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app1">

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

    <input type="number" v-model="age" />

    <p>{{name}} - {{age}}</p>

</div>

<div id="app2">

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

    <button v-on:click="onClick">Change</button>

</div>

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

<script>

let vm1 = new Vue({

    el: '#app1',

    data: {name:'Tom', age: 22},

    methods: {

        setName: function(value){

            this.name = value;

        }

    }

});

// установка свойств объекта Vue в обычном коде JS

vm1.name = "Sam";

vm1.setName("Mike");

// установка свойств объекта Vue в другом объекте Vue

let vm2 = new Vue({

    el: '#app2',

    data: {title:''},

    methods: {

        onClick: function(){

            vm1.setName(this.title);

            //vm1.name = this.title;  // или так

        }

    }

});

</script>

</body>

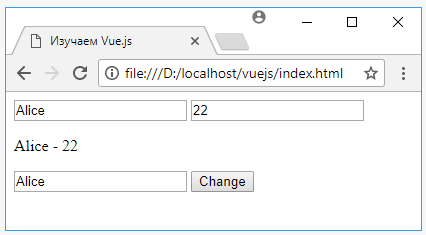
</html>

Здесь определены два объекта Vue, каждый из которых управляет своим шаблоном. Определяя для каждого объекта переменные, можно через эти переменные манипулировать объектами:

vm1.name = "Sam";

vm1.setName("Mike");

В данном случае по нажатию на кнопку во втором объекте Vue можно изменить значения в первом объекте Vue.



Также можно определять дополнительные свойства и методы у объекта извне, как и для любого объекта JavaScript:

<div id="app1">

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

    <input type="number" v-model="age" />

    <p>{{name}} - {{age}} - {{position}}</p>

</div>

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

<script>

let vm1 = new Vue({

    el: '#app1',

    data: {name:'Tom', age: 34}

});

// определение нового метода

vm1.setAge = function(value){

    vm1.age = value;

}

// определение нового свойства

vm1.position = "admin";

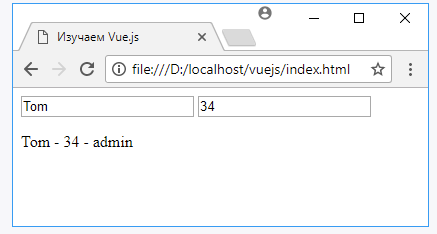
setTimeout(function(){

    vm1.position = "moderator";

}, 3000);

</script>

Однако все свойства определяются вне объекта Vue, как в данном случае свойство position, не будут реактивными и не будут автоматически отслеживаться. Соответственно после выполнения функции setTimeout в данном случае мы не увидим новое значение для этого свойства, пока не будет изменено значение отслеживаемого свойства name или age.



**refs и управление html-элементами**

С помощью параметра refs можно ссылаться на определенный html-элемент из шаблона и управлять им. Например, определим следующую страницу:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2 ref="header">Hello world!</h2>

    <button v-on:click="change">Change</button>

</div>

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

<script>

let vm1 = new Vue({

    el: "#app",

    methods:{

        change: function(){

            this.$refs.header.innerText = "Welcome to Vue.js";

        }

    }

});

</script>

</body>

</html>

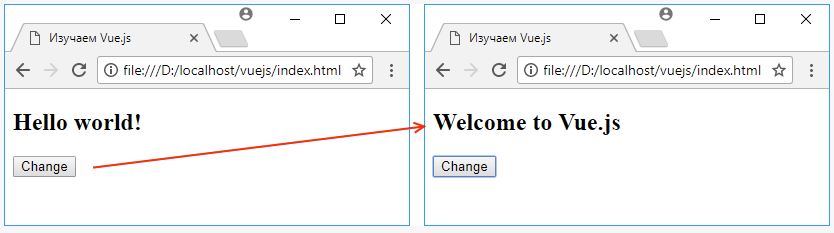
С помощью атрибута ref для html-элемента устанавливается ключ, через который потом можно ссылаться на этот элемент.

<h2 ref="header">Hello world!</h2>

В методах объекта Vue можно обратиться к элементу по установленному ключу через объект $refs:

this.$refs.header.innerText = "Welcome to Vue.js";

С объектом, который представляет html-элемент, (в данном случае this.$refs.header) фактически можно также работать, как и со стандартными объектами JavaScript, которые представляют html-элементы. То есть можно обратиться к его свойствам innerText или innerHTML. Хотя в реальности объект this.$refs.header будет представлять надстройку над объектом js, который собственно и представляет html-элемент. Но фактически через this.$refs.header мы сможем уравлять элементом h2 на веб-странице.



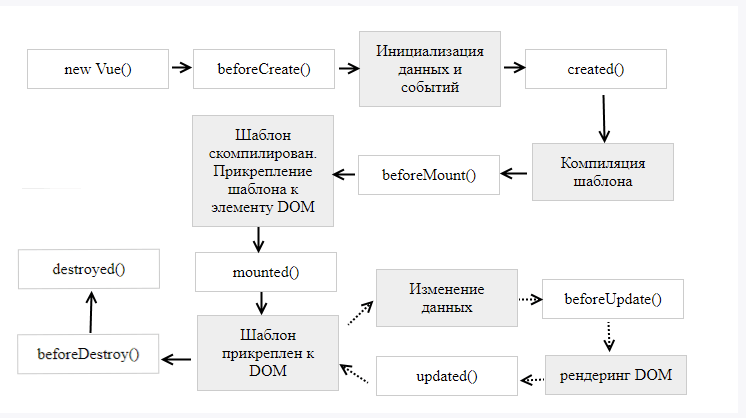
* 1. 7. Жизненный цикл Vue

В процессе своей работы приложение Vue.js проходит через ряд этапов жизненного цикла. И с помощью специальных методов можно вызвать некоторые действия на этих различных этапах жизненного цикла.

Весь жизненный цикл приложения Vue.js можно описать следующим образом:

1. Вызывается конструктор new Vue().
2. Перед созданием объекта Vue вызывается метод beforeCreate().
3. Далее происходит инициализация объекта Vue, установка его данных и методов.
4. После создания объекта Vue вызывается метод created().
5. Далее выполняется компиляция шаблона.
6. Вызывается метод beforeMount().
7. Элемент html, к которому прикреплен объект Vue, заменяется скомпилированным шаблоном.
8. Вызывается метод mounted(), и после этого шаблон прикреплен к DOM, и можно с ним работать.
9. Если в процессе работы обновляются данные объекта Vue, то происходит еще ряд событий:
10. Данные изменяются.
11. Вызывается метод beforeUpdate().
12. Производится повторный рендеринг DOM для его соответствия виртуальному DOM.
13. Вызывается метод updated(). DOM на веб-странице обновлен, и продолжается работа с приложением Vue.js.
14. При завершении работы приложения вызывается метод beforeDestroy().
15. И в конце вызывается метод destroyed(). Объект Vue удалено из памяти, и больше с ним работать нельзя.

Схематично жизненный цикл можно представить так:



Для примера используем методы управления жизненным циклом:

<!DOCTYPE html>

<html>

<head>

<title>Жизненный цикл Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

<button v-on:click="message='Updated'">Update</button>

<button v-on:click="destroy">Destroy</button>

</div>

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

<script>

new Vue({

    el: "#app",

    data:{

        message: 'Welcome to Vue.js'

    },

    methods:{

        destroy: function(){

            this.$destroy();

        }

    },

    beforeCreate: function(){

        console.log('beforeCreate()');

    },

    created: function(){

        console.log('created()');

    },

    beforeMount: function(){

        console.log('beforeMount()');

    },

    mounted: function(){

        console.log('mounted()');

    },

    beforeUpdate: function(){

        console.log('beforeUpdate()');

    },

    updated: function(){

        console.log('updated()');

    },

    beforeDestroy: function(){

        console.log('beforeDestroy()');

    },

    destroyed: function(){

        console.log('destroyed()');

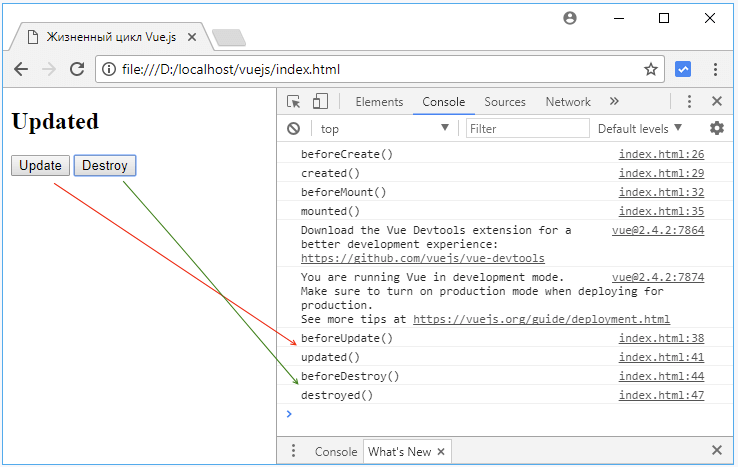
    }

});

</script>

</body>

</html>



При нажатии на кнопку Update выполняется обновление DOM, и вызываются методы beforeUpdate() и updated(). Причем при повторном нажатии эти методы не будут в данном случае вызываться, так как виртуальный DOM никак не изменится, то есть заголовок получит тот же самый текст Update. Соответственно реальный DOM также не будет обновляться.

При нажатии на кнопку Destroy будет вызван метод destroy, в котором будет выполняться метод this.$destroy(). То есть явным образом вызывается у объекта Vue его уничтожение. Затем вызываются методы beforeDestroy() и destroyed(). И после этого объект Vue удален из памяти, и работать с ним больше нельзя, даже при том, что на веб-странице сохраняется его html-разметка.

* 1. 8. Условный рендеринг

**Директива v-if**

Ряд директив позволяет по условию изменять структуру DOM, и одной из таких является директива v-if. Она позволяет отобразить или скрыть элемент html по условию. Например:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <p v-if="visible">Первый параграф</p>

    <p>Второй параграф</p>

    <button v-on:click="visible=!visible">{{visible?'Скрыть':'Показать'}}</button>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            visible: true

        }

    });

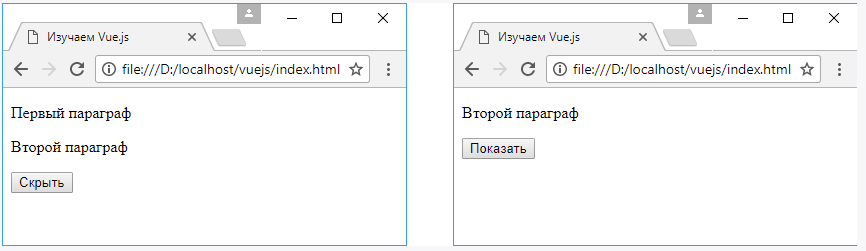
</script>

</body>

</html>

Директива v-if в качестве параметра принимает условие, которое возвращает значение true или false. Если true, то элемент, к которому применяется директива v-if, отображается. Если false, то, наоборот, скрывается. В данном примере это значение определено в свойстве visible.

С помощью кнопки в примере выше можно изменить значение свойства visible и соответственно отобразить или скрыть элемент.



В паре с директивой v-if может использоваться директива v-else, которая позволяет отобразить другой элемент, если условие в директиве v-if равно false:

<div id="app">

    <p v-if="visible">Первый параграф</p>

    <p v-else>Второй параграф</p>

    <button v-on:click="visible=!visible">{{visible?'К параграфу 2':'К параграфу 1'}}</button>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            visible: true

        }

    });

</script>

В данном случае, если условие, которое представляет свойство visible, равно true, то отображается первый параграф. Если условие возвращает false, то отображается второй параграф.

**template**

Так как выражение v-if является директивой, то оно может применяться только к одному элементу. Если надо применить ее к группе элементов, то можно применить v-if для элемента <template>, который выступает в качестве обертки для группы элементов:

<div id="app">

    <template v-if="visible">

        <h1>Заголовок 1</h1>

        <p>Параграф 1</p>

    </template>

    <template v-else>

        <h1>Заголовок 2</h1>

        <p>Параграф 2</p>

    </template>

    <button v-on:click="visible=!visible">{{visible?'К параграфу 2':'К параграфу 1'}}</button>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

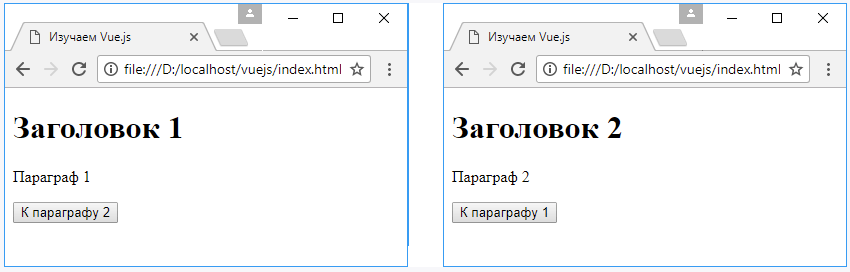
        data: {

            visible: true

        }

    });

</script>



**v-else-if**

С помощью директивы v-else-if к v-if можно добавить обработку дополнительных условий. Например, в зависимости от введенного числа необходимо отображать тот или иной элемент:

<div id="app">

    <input type="number" v-model="number" />

    <p v-if="number==1">Один</p>

    <p v-else-if="number==2">Два</p>

    <p v-else-if="number>2 && number<7">Несколько</p>

    <p v-else>Много</p>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

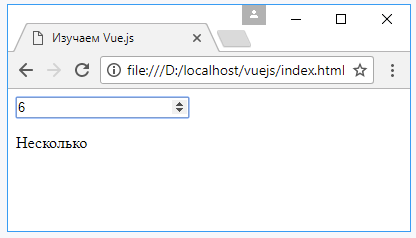
        data: {

            number: 1

        }

    });

</script>



Аналогично можно использовать v-else-if вместе с template:

<template v-if="number==1">

    <p>Один</p>

</template>

<template v-else-if="number==2">

    <p>Два</p>

</template>

<template v-else-if="number>2 && number<7">

    <p>Несколько</p>

</template>

<template v-else>

    <p>Много</p>

</template>

**Директива v-show**

Директива v-show аналогично v-if позволяет скрывать или отображать элементы по определенному условию:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <div v-show="visible">

        <h2>Заголовок</h2>

        <p>Текст</p>

    </div>

    <button v-on:click="visible=!visible">{{visible?'Скрыть':'Отобразить'}}</button>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            visible: true

        }

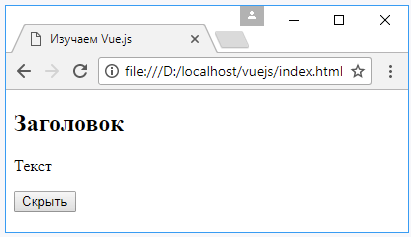
    });

</script>

</body>

</html>

Здесь в зависимости от значения условия, которое представляет свойство visible, будет скрываться или отображаться блок div. А с помощью кнопки можно переключить значение этого свойства с true на false и наоборот.



Но в отличие от v-if директива v-show не изменяет структуру DOM, а манипулирует значением стилевого свойства display. То есть если условие в v-show возвращает false, то для элемента устанавливается стиль display:none, и тем самым данный элемент скрывается на веб-странице.

В то же время манипуляции с DOM через v-if увеличивают накладные расходы и снижают производительность. Поэтому в тех ситуациях, когда возможно частое переключение видимости элемента, следует предпочитать v-show.

**v-for**

Для рендеринга коллекций предназначена директива v-for. Она имеет следующий синтаксис:

v-for="item in items",

где items представляет массив, а item псевдоним для текущего перебираемого элемента из массива items.

Например, выведем массив элементов на страницу:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <ul>

    <li v-for="phone in phones">{{phone}}</li>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            phones: ['iPhone 7', 'Galaxy S8', 'Nokia N8', 'Xiaomi Mi6']

        }

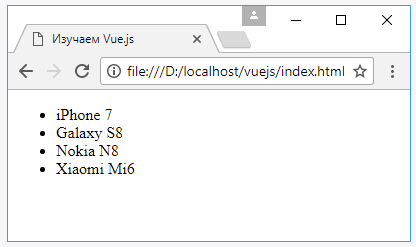
    });

</script>

</body>

</html>

Таким образом, для каждого элемента в массиве phones будет создаваться html-элемент <li>.



Подобным образом можно выводить и более сложные объекты из массива:

<div id="app">

    <ul>

    <li v-for="user in users"><p>Имя: {{user.name}}</p>

                            <p>Возраст: {{user.age}}</p></li>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            users: [

                    {name:'Tom', age:22},

                    {name:'Bob', age:31},

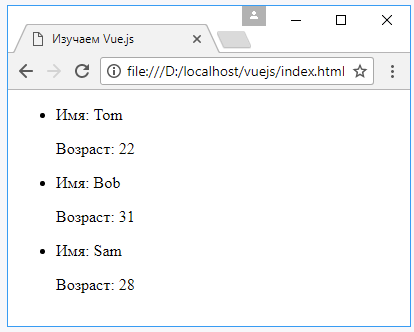
                    {name: 'Sam', age:28}

            ]

        }

    });

</script>



**Индексы**

При переборе элементов с помощью дополнительного параметра в v-for можно получить индекс элемента в массиве, используя следующий синтаксис:

v-for="(element, index) in array",

где element – это текущий перебираемый элемент в массиве array, а index - индекс этого элемента в массиве. Например:

<div id="app">

    <ul>

        <li v-for="(phone, index) in phones">{{index + 1}}. {{phone}}</li>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

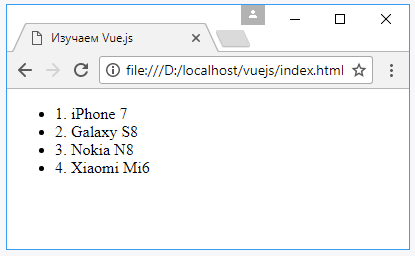
        data: {

            phones: ['iPhone 7', 'Galaxy S8', 'Nokia N8', 'Xiaomi Mi6']

        }

    });

</script>



**Перебор объектов**

Подобно тому, как перебирается массив, можно перебирать и все свойства одного объекта с помощью синтаксиса:

v-for="(value, property) in obj",

где property – название свойства объекта, а value - его значение.

<div id="app">

    <ul>

        <li v-for="user in users">

            <p v-for="(value,key) in user">{{key}} : {{value}}</p>

        </li>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            users: [

                    {name:'Tom', age:22},

                    {name:'Bob', age:31},

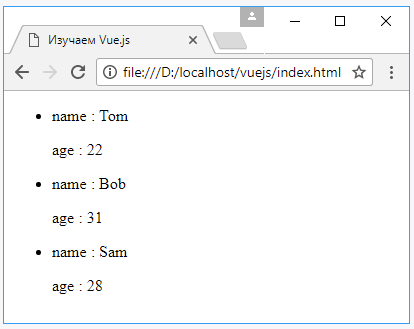
                    {name: 'Sam', age:28}

            ]

        }

    });

</script>



Директиву v-for можно применить только к одному html-элементу. Если необходимо, чтобы для каждого объекта из массива создавалось несколько html-элементов, то блок этих элементов следует обертывать элементом template:

<div id="app">

    <ul>

        <template v-for="user in users">

            <li>Name: {{user.name}}</li>

            <li>Age: {{user.age}}</li>

        </template>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            users: [

                    {name:'Tom', age:22},

                    {name:'Bob', age:31},

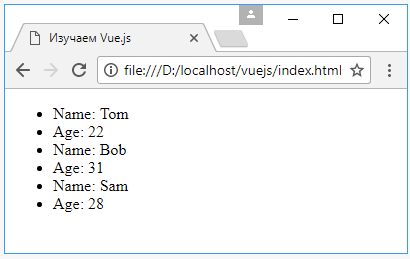
                    {name: 'Sam', age:28}

            ]

        }

    });

</script>



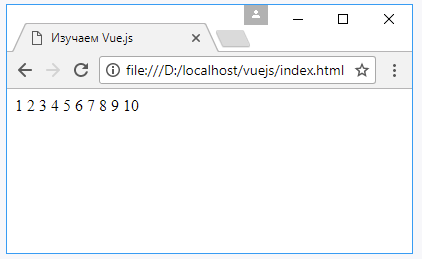
**Перебор чисел**

С помощью v-for можно перебрать все числа от 1 до определенного значения. Например, перебор всех чисел от 1 до 10:

<div>

  <span v-for="n in 10">{{ n }} </span>

</div>



* 1. 9. Работа с массивами

Во Vue.js для каждого массива определено ряд методов, с помощью которых можно управлять элементами массива:

push()

pop()

shift()

unshift()

splice()

sort()

reverse()

Эти методы являются обертками над одноименными [стандартными методами](https://metanit.com/web/javascript/5.3.php) javascript для управления массивами и работают точно также. Единственное отличие их от стандартных методов состоит в том, что эти методы-обертки информируют систему Vue.js о том, что с массивом были произведены некоторые действия, и соответственно для этого массива может быть произведен повторный рендеринг на веб-странице.

Но кроме выше описанных методов, которые изменяют отдельные элементы массив, есть ряд методов в javascript, которые возвращают новый массив, типа filter(), concat(), slice(). Результат таких функций лучше привязывать к вычисляемому свойству, которое позволит произвести повторный рендеринг элементов веб-страницы.

**Добавление и удаление**

Определим код для добавления нового элемента в массив и удаления из массива:

<!DOCTYPE html>

<html>

<head>

<title>Изучаем Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <p>

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

        <button v-on:click="phones.push(newPhone)">Добавить</button>

    </p>

    <ul>

      <li v-for="(phone, index) in phones">

      <p>{{ phone }} <button v-on:click="phones.splice(index, 1)">Удалить</button></p>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            newPhone: '',

            phones: ['iPhone 7', 'Galaxy S8', 'Nokia N8', 'Xiaomi Mi6']

        }

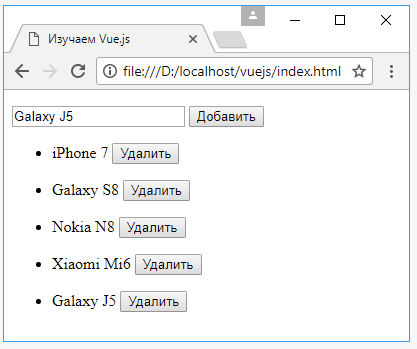
    });

</script>

</body>

</html>

При нажатии на кнопку в массив будет добавлять введенное в текстовое поле значение, которое доступно через свойство newPhone. И в итоге произойдет обновление списка на веб-странице, и мы увидим добавленный элемент. И кроме того, для каждого элемента предусмотрена кнопка, через которую по индексу можно удалить элемент из массива.



**Установка элемента в массиве**

При работе с массивами можно столкнуться с некоторыми ограничениями. В частности, нельзя просто по индексу переустановить элемент массива, присвоив ему новое значение:

this.phones[1] = 'Samsung Galaxy S8';

Для установки значения нам надо использовать метод Vue.set():

Vue.set(массив, индекс\_элемента, новое\_значение);

Например, обновим второй элемент массива:

<div id="app">

    <ul>

      <li v-for="phone in phones">

      <p>{{ phone }}</p>

    </ul>

    <button v-on:click="updateList">Обновить</button>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            newPhone: '',

            phones: ['iPhone 7', 'Galaxy S8', 'Nokia N8', 'Xiaomi Mi6']

        },

        methods:{

            updateList: function(){

                Vue.set(this.phones, 1, 'Samsung Galaxy S8');

            }

        }

    });

</script>

**Возвращение нового массива**

Ряд методов, такие как slice, concat, filter, не изменяют текущий массив, а возвращают новый. Одно из решений может заключаться в переустановке массива. Например, используем метод slice, который возвращает часть массива:

<div id="app">

    <ul>

      <li v-for="phone in phones">

      <p>{{ phone }}</p>

    </ul>

    <button v-on:click="updateList">Обновить</button>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            newPhone: '',

            phones: ['iPhone 7', 'Galaxy S8', 'Nokia N8', 'Xiaomi Mi6']

        },

        methods:{

            updateList: function(){

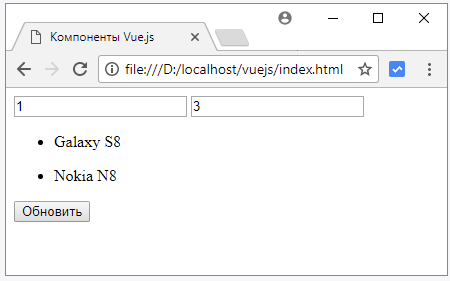
                this.phones = this.phones.slice(1,3);

            }

        }

    });

</script>

Для фиксации нового массива присваиваем полученный массив свойству phones.

Однако данный способ не всегда является оптимальным. Особенно если надо сохранить старый массив и изменять лишь его визуальное представление. И более идеальным вариантом, как правило, является разделение данных и представления этих данных. Для представления данных обычно определяется вычисляемое свойство-список, элементы которого выводятся на веб-страницу:

<div id="app">

    <input v-model="start" type="number" />

    <input v-model="end" type="number" />

    <ul>

      <li v-for="phone in visibleList">

      <p>{{ phone }}</p>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            start:0,

            end:3,

            phones: ['iPhone 7', 'Galaxy S8', 'Nokia N8', 'Xiaomi Mi6']

        },

        computed:{

            visibleList: function(){

                return this.phones.slice(this.start,this.end);

            }

        }

    });

</script>

**Фильтрация и сортировка массива**

Стандартной ситуацией при работе с коллекциями объектов является фильтрация. Как правило, при фильтрации есть некоторый начальный список, а пользователю же возвращается некоторый временный результат. Начальный же список при этом не изменяется. И для этого во Vue.js лучше определить привязку к вычисляемому свойству:

<div id="app">

    <p><input type="text" v-model="company" /></p>

    <ul>

      <li v-for="phone in filteredList">

        <p>{{ phone.title }} - {{ phone.company }}</p>

    </ul>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            company: '',

            phones: [

                {title:'iPhone 7', company:'Apple'},

                {title:'iPhone 6S', company:'Apple'},

                {title:'Galaxy S8', company:'Samsung'},

                {title:'Galaxy S7 Edge', company:'Samsung'},

                {title:'Nokia N8', company:'HMD Global'}]

        },

        computed:{

            filteredList: function(){

                let comp = this.company;

                return this.phones.filter(function (elem) {

                    if(comp==='') return true;

                    else return elem.company.indexOf(comp) > -1;

                })

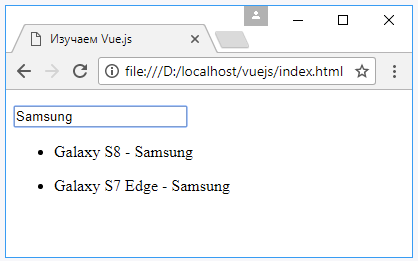
            }

        }

    });

</script>

В данном случае фильтрация работает по принципу живого поиска. При вводе значения в текстовое поле происходит повторное вычисление свойства filteredList. Это свойство представляет результат функции, которая возвращает те объекты, у которых поле company соответствует введенному значению. То есть идет фильтрация телефонов по компании производителя. Если же значение не введено, то возвращаем все элементы из массива phones. В итоге динамически можно фильтровать элементы списка.



Для сортировки списка применяется похожая техника, что и для фильтрации:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

<style>

a:hover{cursor:pointer;}

</style>

</head>

<body>

<div id="app">

    <table>

      <tr>

        <td><a @click="sortParam='title'">Модель</a></td>

        <td><a @click="sortParam='company'">Компания</a></td>

        <td><a @click="sortParam='price'">Цена</a></td></tr>

      <tr v-for="phone in sortedList">

        <td>{{phone.title}}</td><td>{{phone.company}}</td><td>{{phone.price}}</td>

    </table>

</div>

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

<script>

    let app = new Vue({

        el: '#app',

        data: {

            sortParam: '',

            phones: [

                {title:'Galaxy S8', company:'Samsung', price: 45000},

                {title:'iPhone 7', company:'Apple', price: 49000},

                {title:'Nokia N8', company:'HMD Global', price: 25000},

                {title:'Galaxy Note 8', company:'Samsung', price: 50000},

                {title:'iPhone 8', company:'Apple', price: 60000}]

        },

        computed:{

            sortedList () {

                switch(this.sortParam){

                    case 'title': return this.phones.sort(sortByTitle);

                    case 'company': return this.phones.sort(sortByCompany);

                    case 'price': return this.phones.sort(sortByPrice);

                    default: return this.phones;

                }

             }

        }

    });

let sortByTitle = function (d1, d2) {return (d1.title.toLowerCase() > d2.title.toLowerCase()) ? 1 : -1;};

let sortByCompany = function (d1, d2) { return (d1.company.toLowerCase() > d2.company.toLowerCase()) ? 1 : -1; };

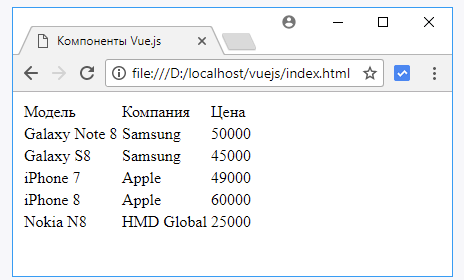
let sortByPrice = function (d1, d2) { return (d1.price > d2.price) ? 1 : -1; };

</script>

</body>

</html>

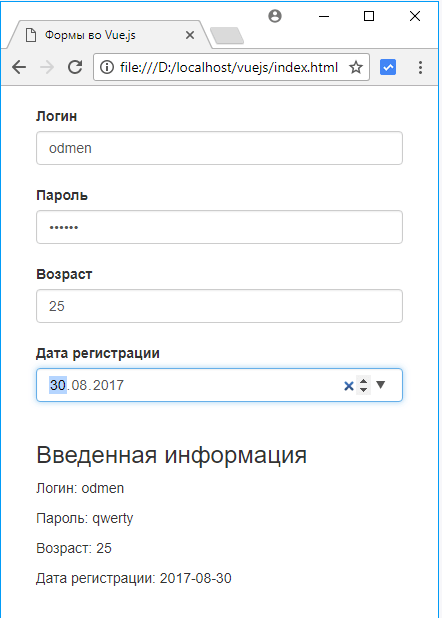
Как и в случае с фильтрацией, привязка устанавливается к вычисляемому свойству. При нажатии на заголовок столбца в таблице происходит переустановка значения свойства sortParam, которое представляет критерий фильтрации. При его изменении повторно вычисляется свойство sortedList, которое сортирует массив phones в соответствии со значением в sortParam. Для фильтрации по трем критериям определены три вспомогательные функции sortByCompany, sortByPrice и sortByTitle.

* 1. 
  2. 10. Работа с формами

**Элементы input и textarea**

Для работы с элементами ввода, в частности, с элементами input, select, textarea, применяется директива v-model, которая устанавливает двустороннюю привязку между элементом ввода и свойством объекта Vue. При этом v-model игнорирует значения атрибутов value, checked и selected, которые имеются у элементов ввода. Например, используем ряд базовых элементов input:

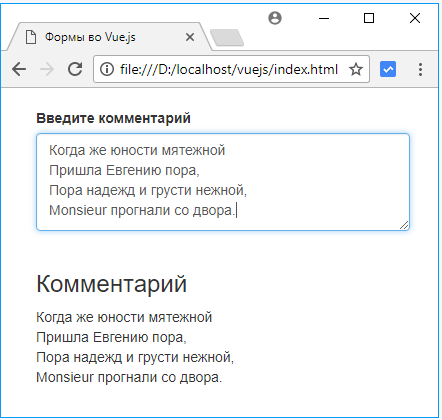
|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>  <head>  <title>Формы во Vue.js</title>  <meta charset="utf-8" />  <link rel="stylesheet" href="<https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css>">  </head>  <body>  <div id="app" class="col-md-4">      <label>Логин</label>      <input type="text" v-model="login" placeholder="Введите логин" class="form-control" /><br>      <label>Пароль</label>      <input type="password" v-model="password" placeholder="Введите пароль" class="form-control" /><br>      <label>Возраст</label>      <input type="number" v-model="age" min="18" placeholder="Введите возраст" class="form-control" /><br>      <label>Дата регистрации</label><br>      <input type="date" v-model="date" placeholder="Введите дату" class="form-control" />      <br>      <div>          <h3>Введенная информация</h3>          <p>Логин: {{login}}</p>          <p>Пароль: {{password}}</p>          <p>Возраст: {{age}}</p>          <p>Дата регистрации: {{date}}</p>      </div>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              login:'',              password:'',              date: '',              age: 22          }      });  </script>  </body>  </html> |



**textarea**

Элемент textarea позволяет вводить многострочный текст и также связывается со свойством во Vue через директиву v-model:

|  |  |
| --- | --- |
|  | <div id="app" class="col-md-4">      <label>Введите комментарий</label>      <textarea v-model="comment" class="form-control"></textarea>      <br>      <div>          <h3>Комментарий</h3>          <p style="white-space: pre-line">{{comment}}</p>      </div>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              comment:'Комментарий....'          }      });  </script> |



В то же время есть некоторые особенности по работе с textarea. Прежде всего, если используется директива v-model, то внутри textarea нельзя поместить текст с помощью интерполяции:

|  |  |
| --- | --- |
|  | <textarea v-model="comment" class="form-control">{{comment}}</textarea> |

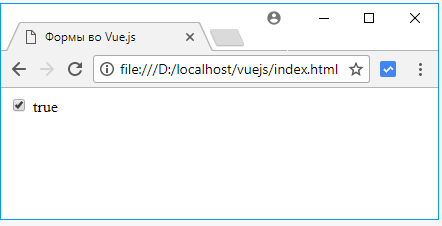
И кроме того, при вводе текста в textarea с помощью клавиши Enter можно переносить строки. Чтобы переносы строк сохранялись при выводе текста на страницу, необходимо применять соответствующие стили css:

|  |  |
| --- | --- |
|  | <p style="white-space: pre-line">{{comment}}</p> |

**Флажки (Checkbox)**

Флажки представляют элементы управления, которые могут находиться в двух состояниях: отмеченном и неотмеченном. Одиночный флажок может быть привязан к логическому значению true или false:

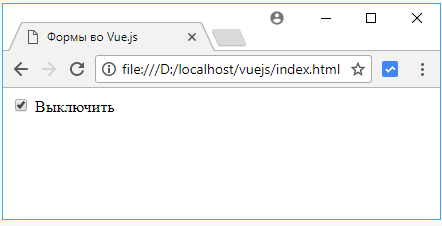
|  |  |
| --- | --- |
|  | <div id="app">      <input type="checkbox" id="checkbox" v-model="checked">      <label for="checkbox">{{ checked }}</label>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              checked: true          }      });  </script> |



По умолчанию для отмеченного состояния используется значение true, а для неотмеченного – false. Но вместо них можно определить свои собственные значения для обоих состояний и между этими значениями переключаться:

|  |  |
| --- | --- |
|  | <div id="app">      <input type="checkbox" v-model="light" v-bind:true-value="on" v-bind:false-value="off" />        <span v-if="light===on">Выключить</span>        <span v-else>Включить</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              on: true,              off:false,              light: true          }      });  </script> |

Вместо значений true и false флажок привязывается к одному из свойств on или off. С помощью атрибута v-bind:true-value="on" отмеченное состояние будет сопоставляться со свойством on. Соответственно в неотмеченном состоянии флажок приобретает значение off. А само выбранное значение попадает в свойство light.

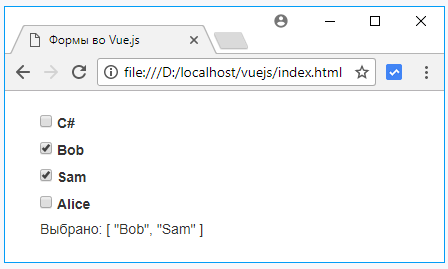


**Привязка к массиву**

Также группа флажков может быть привязана к одному массиву. В этом случае при отметке флажка его значение попадает в массив:

|  |  |
| --- | --- |
|  | <div id="app">      <input type="checkbox" value="Tom" v-model="selectedUsers">      <label>C#</label><br>        <input type="checkbox" value="Bob" v-model="selectedUsers">        <label>Bob</label><br>        <input type="checkbox" value="Sam" v-model="selectedUsers">        <label>Sam</label><br>        <input type="checkbox" value="Alice" v-model="selectedUsers">        <label>Alice</label><br>        <span>Выбрано: {{selectedUsers}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              selectedUsers:[]          }      });  </script> |

В данном случае 4 флажка привязаны к массиву selectedUsers. Отметив определенный флажок, добавим его значение в массив.

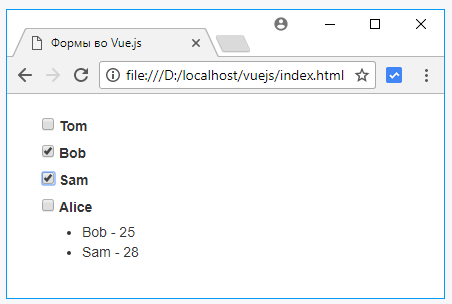


**Привязка к объекту**

В примере выше значение каждого флажка представляло некоторую строку. Но также флажок может быть привязан к целому объекту. Для динамической привязки у элемента html применяется атрибут v-bind:value:

|  |  |
| --- | --- |
|  | <div id="app" class="col-md-4">      <template v-for="user in users">          <input type="checkbox" v-bind:value="user" v-model="selectedUsers">          <label>{{user.name}}</label><br>      </template>        <ul>          <li v-for="user in selectedUsers">{{user.name}} - {{user.age}}</li>      </ul>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              users:[                  {name:'Tom', age:22},                  {name:'Bob', age:25},                  {name:'Sam', age:28},                  {name:'Alice', age:26}              ],              selectedUsers:[]          }      });  </script> |

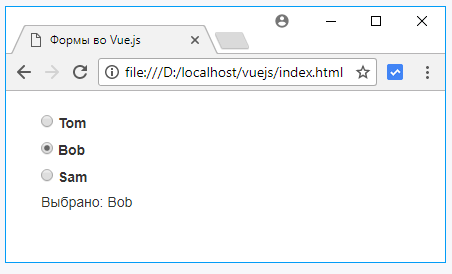
Здесь динамически для каждого объекта в массиве формируется флажок, а при выборе флажка объект попадет в массив selectedUsers.



**Переключатели (radiobutton)**

Переключатели или радиокнопки позволяют определить группу альтернативных вариантов, из которых одномоментно пользователь может выбрать только один. Переключатели привязываются к одному свойству. При выборе переключателя связанное свойство получает выбранное значение:

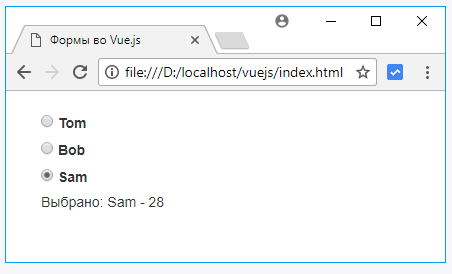
|  |  |
| --- | --- |
|  | <div id="app">      <input type="radio" value="Tom" v-model="user">      <label>Tom</label>      <br>      <input type="radio" value="Bob" v-model="user">      <label>Bob</label>      <br>      <input type="radio" value="Sam" v-model="user">      <label>Sam</label>      <br>      <span>Выбрано: {{ user }}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              user:''          }      });  </script> |



**Привязка к объекту**

Также каждый из переключателей может быть привязан к сложному объекту. В этом случае используется атрибут v-bind:value. Результатом выбора соответственно также будет сложный объект:

|  |  |
| --- | --- |
|  | <div id="app">      <input type="radio" v-bind:value="{name:'Tom', age:22}" v-model="user">      <label>Tom</label>      <br>      <input type="radio" v-bind:value="{name:'Bob', age:25}" v-model="user">      <label>Bob</label>      <br>      <input type="radio" v-bind:value="{name:'Sam', age:28}" v-model="user">      <label>Sam</label>      <br>      <span>Выбрано: {{user.name}} - {{user.age}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              user:{}          }      });  </script> |



**Привязка к массиву объектов**

В случае выше было бы более оптимально хранить данные в массиве и динамически создавать по элементам массива группу радиокнопок:

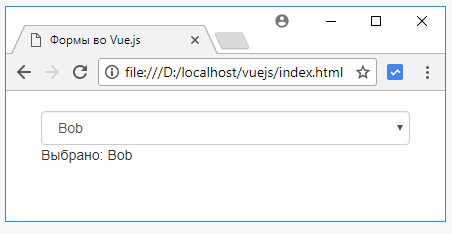
|  |  |
| --- | --- |
|  | <div id="app" class="col-md-4">      <template v-for="user in users">          <input type="radio" v-bind:value="user" v-model="selectedUser">          <label>{{user.name}}</label><br>      </template>      <span v-if="selectedUser!==null">Выбрано: {{selectedUser.name}} - {{selectedUser.age}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              users:[                  {name:'Tom', age:22},                  {name:'Bob', age:25},                  {name:'Sam', age:28},                  {name:'Alice', age:26}              ],              selectedUser:null          }      });  </script> |

**Список select**

Элемент select позволяет создать список с возможностью одиночного и множественного выбора.

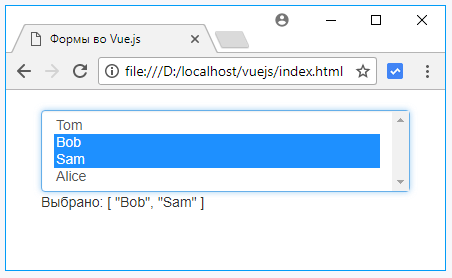
Выпадающий список с одиночным выбором можно привязать к свойству Vue, которое хранит выбранный элемент:

|  |  |
| --- | --- |
|  | <div id="app">      <select v-model="user">          <option>Tom</option>          <option>Bob</option>          <option>Sam</option>      </select>      <span>Выбрано: {{user}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              user:''          }      });  </script> |



Список select с множественным выбором можно связать с массивом, в который попадут все выбранные значения:

|  |  |
| --- | --- |
|  | <div id="app">      <select v-model="users" multiple>          <option>Tom</option>          <option>Bob</option>          <option>Sam</option>          <option>Alice</option>      </select>      <span>Выбрано: {{users}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              users:[]          }      });  </script> |



**Привязка к массиву объектов**

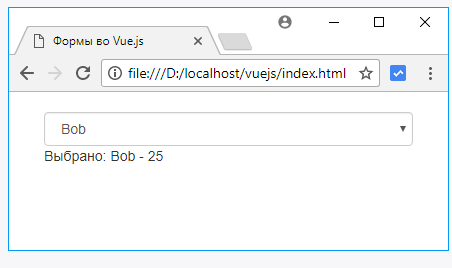
Нередко источником для элементов списка служит массив, элементы которого нередко представляют сложные объекты. Для динамического создания списка из массива применяется директива v-for, а для привязки значения к элементу списка – атрибут v-bind:value:

|  |  |
| --- | --- |
|  | <div id="app">      <select v-model="selectedUser">          <option v-for="user in users" v-bind:value="user.name">{{user.name}}</option>      </select>      <span>Выбрано: {{selectedUser}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              users:[                  {name:'Tom', age:22},                  {name:'Bob', age:25},                  {name:'Sam', age:28},                  {name:'Alice', age:26}              ],              selectedUser:''          }      });  </script> |

В данном случае в списке отображается значение свойства name объекта из массива и это же свойство name будет использоваться в качестве значения при выборе элемента. Однако можно выбирать и весь объект в целом:

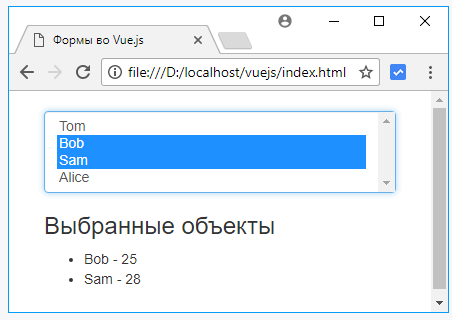
|  |
| --- |
| <div id="app">      <select v-model="selectedUser">          <option v-for="user in users" v-bind:value="user">{{user.name}}</option>      </select>      <span v-if="selectedUser!==null">Выбрано: {{selectedUser.name}} - {{selectedUser.age}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              users:[                  {name:'Tom', age:22},                  {name:'Bob', age:25},                  {name:'Sam', age:28},                  {name:'Alice', age:26}              ],              selectedUser:null          }      });  </script> |

В данном случае выбранный объект из массива будет попадать в свойство selection.



Аналогично можно установить привязку к объектам в списке с множественным выбором. В этом случае выбранные элементы попадут в массив:

|  |  |
| --- | --- |
|  | <div id="app">      <select v-model="selectedUsers" multiple>           <option v-for="user in users" v-bind:value="user">{{user.name}}</option>      </select>      <h3>Выбранные объекты</h3>      <ul>          <li v-for="user in selectedUsers">{{user.name}} - {{user.age}}</li>      </ul>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              users:[                  {name:'Tom', age:22},                  {name:'Bob', age:25},                  {name:'Sam', age:28},                  {name:'Alice', age:26}              ],              selectedUsers:[]          }      });  </script> |



**Модификаторы**

Для управленя вводом с элементов форм можно использовать модификаторы.

*Модификатор .lazy.* По умолчанию директива v-model обновляет связанные с полем ввода данные при каждом вызове события input. То есть при вводе каждого символа в текстовое поле будут изменяться связанные данные. Но вместо этого также можно вызывать обновление не после события input, а после события change. То есть связанные данные будут обновляться только после того, как текстовое поле потеряет фокус. Для этого применяется модификатор .lazy:

|  |  |
| --- | --- |
|  | <div id="app" class="col-md-4">      <input type="text" v-model.lazy="name" class="form-control" />      <span>Введено: {{name}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              name:''          }      });  </script> |

*Модификатор .number.* Модификатор .number позволяет автоматически конвертировать при вводе вводимое значение в число. Поскольку даже если установить у поля input атрибут type="number", то такое поле все равно будет возвращать строку:

|  |  |
| --- | --- |
|  | <div id="app">      <input v-model.number="age" type="number">      <span>Введено: {{age}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              age: 0          }      });  </script> |

*Модификатор .trim.* Модификатор .trim автоматически обрезает начальные и концевые пробелы при вводе строки:

|  |  |
| --- | --- |
|  | <div id="app">      <input v-model.trim="message">      <span>Введено: {{message}}</span>  </div>  <script src="<https://unpkg.com/vue>"></script>  <script>      let app = new Vue({          el: '#app',          data: {              message: ''          }      });  </script> |

Также можно объединять модификаторы:

|  |  |
| --- | --- |
|  | <input v-model.lazy.trim="message">  <input v-model.lazy.number="age" type="number"> |