**Тема 14. Vue.js. Компоненты. Слоты.**

1. **Создание и регистрация компонентов.**
2. **Состояние и поведение компонентов.**
3. **Параметр props.**
4. **Передача в компонент объектов и массивов.**
5. **Родительские и дочерние компоненты.**
6. **Создание событий.**
7. **Изменение внешних массивов и объектов в компонентах.**
8. **Сестринские компоненты.**
9. **Миксины.**
10. **Именованные слоты.**
11. **Слоты с ограниченной областью видимости.**
12. **Фильтры.**

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

* 1. 1. Создание и регистрация компонентов
  2. Компоненты представляют элементы, к которым компилятор Vue прикрепляет некоторое поведение. Компоненты позволяют инкапсулировать код и затем использовать его многократно в различных частях приложения.

Для создания компонента используется функция Vue.component(tagName, options), где параметр tagName – кастомный элемент html, который будет представлять компонент, а параметр options представляет конфигурацию компонента.

Например, определим простейший компонент:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <hello></hello>

    <hello></hello>

</div>

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

<script>

Vue.component('hello', {

    template: '<h2>Hello</h2>'

})

new Vue({

    el: "#app"

});

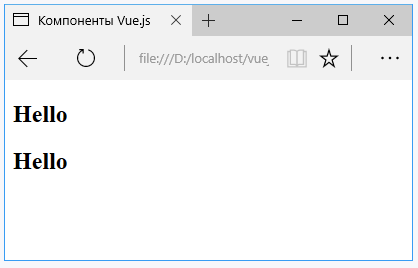
</script>

</body>

</html>

Здесь определен компонент для элемента hello. И неважно, что такого элемента в html нет, создаем его самостоятельно. Более того нельзя использовать для компонента встроенные элементы html типа div или h2. То есть по сути компонент называется "hello".

В объекте options, который передается компоненту, через свойство template можно задать html-разметку, которую будет содержать компонент. В итоге данная разметка будет вставляться вместо элемента <hello></hello>. При этом компонент должен быть определен до элемента Vue, в котором он используется. И при рендеринге страницы вместо элемента hello будет вставлено содержимое компонента:



При этом можно многократно использовать компонент в рамках приложения Vue, и в каждом случае будет происходить рендеринг компонента. Причем компонент может использоваться только в рамках того элемента, к которому прикреплен объект Vue. То есть нельзя написать наподобие, вынеся использование компонента во вне:

<hello></hello>

<div id="app">

</div>

**Локальная и глобальная регистрация компонентов**

Компоненты могут быть зарегистрированы локально и глобально. Глобальные компоненты доступны для любого объекта Vue на веб-странице. Локальные компоненты доступны только в рамках определенных объектов Vue.

Для локальной регистрации компонентов у объекта Vue устанавливается свойство components:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <section-header></section-header>

    <section-content></section-content>

</div>

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

<script>

Vue.component('section-header',{

    template:'<h3>Header</h3>'

});

let comp = {

    template:'<div>Hello World</div>'

};

new Vue({

    el: "#app",

    components:{

        'section-content':comp

    }

});

</script>

</body>

</html>

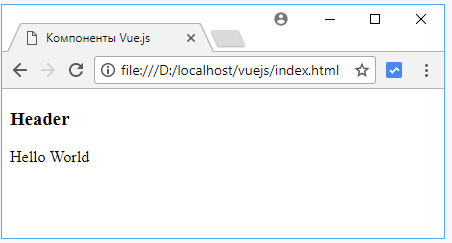
Здесь определен глобальный компонент – section-header. Глобальный компонент определяется с помощью метода Vue.component(). И определен компонент comp. По сути он представляет объект, который передается в качестве второго параметра в Vue.component() и может иметь все те же свойства, например, свойство template. Этот компонент локально регистрируется в объекте Vue:

components:{

    'section-content':comp

}

Причем для его рендеринга будет использоваться элемент <section-content>.

****

Если бы компонент comp не был бы зарегистрирован в объекте Vue, то тогда его нельзя было бы использовать, либо пришлось бы делать его глобальным.

Рассмотрим еще один пример:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app1">

    <section-header></section-header>

    <section-content></section-content>

    <section-footer></section-footer>

</div>

<hr/>

<div id="app2">

    <section-header></section-header>

    <section-content></section-content>

</div>

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

<script>

Vue.component('section-header',{

    template:'<h2>Header</h2>'

});

let comp1 = {

    template:'<div>Content 1</div>'

};

let comp2 = {

    template:'<div>Content 2</div>'

};

let footer = {

    template:'<p><b>Footer</b></p>'

};

new Vue({

    el: "#app1",

    components:{

        'section-content':comp1,

        'section-footer':footer

    }

});

new Vue({

    el: "#app2",

    components:{

        'section-content':comp2

    }

});

</script>

</body>

</html>

Здесь определено два объекта Vue. Компонент section-header является глобальным и поэтому доступен из любого объекта Vue. В дополнение к нему первый объект локально регистрирует два компонента – comp1 и footer, а второй объект Vue – один компонент comp2.

## 

* 1. 2. Состояние и поведение компонентов

**Параметр data**

Как и объекты Vue, компоненты могут содержать некоторые данные или состояние в виде параметра data. Но при этом в компонентах параметр data должен представлять функцию, которая в свою очередь и возвращает состояние компонента. Например:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <counter></counter>

</div>

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

<script>

Vue.component('counter', {

    data: function(){

        return {

            header: 'Counter Program'

        }

    },

    template:'<div><h2>{{header}}</h2></div>'

});

new Vue({

    el: "#app"

});

</script>

</body>

</html>

## 

Подобным образом можно определять данные в локально регистрируемых компонентах:

let comp = {

    template:'<div><h2>{{header}}</h2></div>',

    data: function(){

        return {

            header: 'Counter Program'

        }

    }

};

new Vue({

    el: "#app",

    components:{

        'counter':comp

    }

});

**Параметр methods**

Кроме состояния компоненты могут определять поведение в виде методов, которые определяются через параметр methods, как и в объектах Vue:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <counter></counter>

</div>

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

<script>

Vue.component('counter', {

    data: function(){

        return {

            header: 'Counter Program',

            count:0

        }

    },

    template:`<div><h2>{{header}}</h2>

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

                <span>{{count}}</span>

            </div>`,

    methods:{

        increase:function(){

            this.count++;

        }

    }

});

new Vue({

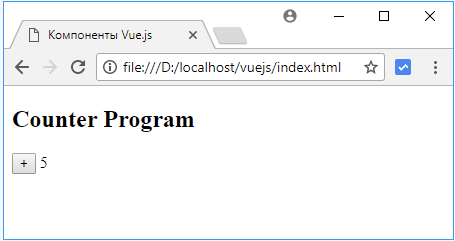
    el: "#app"

});

</script>

</body>

</html>

****Здесь по нажатию на кнопку вызывается метод increase, который увеличивает значение свойства count.

**Разделяемое состояние компонентов**

Для каждого компонента можно определить его собственное состояние. Но также можно определять некоторое общее состояние для нескольких компонентов. Для этого состояние определяется в виде внешнего объекта:

let data = {

        header: 'Counter Program',

        count:0

};

Vue.component('counter', {

    data: function(){

        return data

    },

    template:`<div><h2>{{header}}</h2>

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

                <span>{{count}}</span>

            </div>`,

    methods:{

        increase:function(){

            this.count++;

        }

    }

});

Конкретно в данном примере не будет разницы от вынесения объекта data во вне. В то же время между двумя способами определения data все-таки есть разница. Например, определим следующую веб-страницу:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <counter></counter>

    <counter></counter>

</div>

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

<script>

Vue.component('counter', {

    data: function(){

        return {

            header: 'Counter Program',

            count:0

        }

    },

    template:`<div><h2>{{header}}</h2>

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

                <span>{{count}}</span>

            </div>`,

    methods:{

        increase:function(){

            this.count++;

        }

    }

});

new Vue({

    el: "#app"

});

</script>

</body>

</html>

В данном случае объект Vue использует два раза компонент counter. И каждый объект counter работает со своим состоянием. Изменение свойство count в одном компоненте никак не скажется на другом.

## 

Рассмотрим другую ситуацию, когда компоненты используют одно и то же состояние:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <counter></counter>

    <counter></counter>

</div>

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

<script>

let data = {

        header: 'Counter Program',

        count:0

};

Vue.component('counter', {

    data: function(){

        return data

    },

    template:`<div><h2>{{header}}</h2>

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

                <span>{{count}}</span>

            </div>`,

    methods:{

        increase:function(){

            this.count++;

        }

    }

});

new Vue({

    el: "#app"

});

</script>

</body>

</html>

В этом случае одно и то же состояние будет разделяться обоими компонентами. Поэтому изменения в состоянии в одном компоненте повлияют и на другой компонент.

## 

* 1. 3. Параметр Props

Каждый компонент определяет параметр props, через который можно передать компоненту извне различные данные. Например:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <message-comp message="hello"></message-comp>

</div>

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

<script>

Vue.component('message-comp', {

  props: ['message'],

  template: '<h2>{{ message }}</h2>'

})

new Vue({

    el: "#app"

});

</script>

</body>

</html>

Параметр props хранит массив ключей или свойств, которым извне можно передать значения. В данном случае определено одно свойство message. Это свойство также можно использовать в шаблоне компонента (<span>{{ message }}</span>). Чтобы передать свойству значение, у элемента компонента применяется атрибут, который называется также, как и свойство:

<message-comp message="hello"></message-comp>

## 

В некотором плане свойства props похожи на те свойства, которые определяются через параметр data. При этом надо учитывать, что нельзя определить свойство с одним и тем же именем и в props, и в data:

Vue.component('message-comp', {

  props: ['message'],

  data: function(){

    return { message: 'hi all'}

  },

  template: '<h2>{{ message }}</h2>'

})

И в данном случае будет использоваться свойство message из props, а не из параметра data.

**Динамические свойства**

В примере выше свойству message передается статическое литеральное значение – строка "hello". Однако можно также устанавливать значения свойств динамически в зависимости от введенных в поля ввода данных:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

    <message-comp v-bind:message="welcome"></message-comp>

</div>

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

<script>

Vue.component('message-comp', {

  props: ['message'],

  template: '<h2>{{message}}</h2>'

})

new Vue({

    el: "#app",

    data: {

        welcome:''

    }

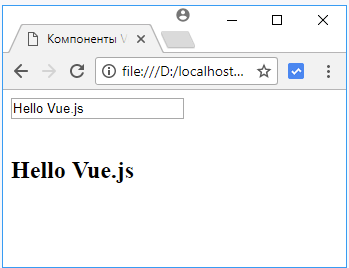
});

</script>

</body>

</html>

Теперь свойство message устанавливается динамически в зависимости от введеного значения:



Для этого применяется механизм привязки – перед атрибутом указывается директива v-bind, которая определяет, к какому значению идет привязка:

<message-comp v-bind:message="welcome"></message-comp>

Также можно использовать сокращенную форму:

<message-comp :message="welcome"></message-comp>

**Привязка к сложным объектам**

Допустим, наш компонент выводит данные о пользователе – имя и возраст. Передадим эти данные в компонент:

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

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

    <user :name="name" :age="age"></user>

</div>

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

<script>

Vue.component('user', {

  props: ['name', 'age'],

  template: '<div><h2>User</h2><p>Name: {{name}}</p><p>Age: {{age}}</p></div>'

})

new Vue({

    el: "#app",

    data: {

        name: '',

        age: 18

    }

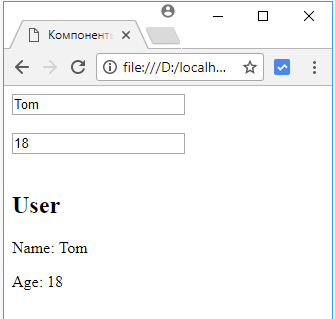
});

</script>

</body>

</html>

Здесь компонент получает значения для своих свойств name и age.



Но в данном случае свойства name и age можно оформить как единый объект и передавать этот объект целиком в компонент:

<div id="app">

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

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

    <user v-bind="user"></user>

</div>

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

<script>

Vue.component('user', {

  props: ['name', 'age'],

  template: '<div><h2>User</h2><p>Name: {{name}}</p><p>Age: {{age}}</p></div>'

})

new Vue({

    el: "#app",

    data: {

        user: {

            name: '',

            age: 18

        }

    }

});

</script>

С помощью атрибута v-bind="user" объект user сразу передается компоненту и автоматически сопоставляется со свойствами name и age, которые определены в компоненте. Но естественно должно быть соответствие между названиями свойств объекта user и названиями свойств компонента. В остальном результат работы будет тот же самый, что и в предыдущем случае. В то же время несмотря на то, что вроде бы идет привязка к объекту user, тем не менее компонент получает значения его свойств по отдельности.

**Валидация props**

С помощью props можно передавать извне данные в компонент. Однако не всегда данные, которые передаются приложению, являются крректными. Например, для возраста передать отрицательное числового значение или при вводе имени ввести пустую строку. Чтобы избежать подобных ситуаций, когда передаваемые данные не отвечают нашим ожиданиям и не являются корректными, применяется механизм валидации.

Прежде всего можно указать тип для свойств. В качестве типов можно использовать следующие: String, Number, Boolean, Function, Object, Array, Symbol. Например, ранее параметр props определялся следующим образом:

props: ['name', 'age']

Но фактически это объект, который можно переписать иным образом:

Vue.component('user', {

  props: {name: String, age: Number},

  template: '<div><h2>User</h2><p>Name: {{name}}</p><p>Age: {{age}}</p></div>'

})

Здесь параметр props определен как объект. name и age в этом объекте выступают в качестве свойств, при этом для каждого свойства устанавливается тип. Для более точной валидации свойства для него можно задать ряд параметров:

* type: тип свойства;
* required: если этот параметр имеет значение true, то для данного свойства обязательно надо ввести значение;
* default: значение по умолчанию, которое устанавливается, если для свойства извне не передается никакого значения;
* validator: функция, которая валидирует значение свойства. Если значение корректно, то функция валидатора должна возвращать true, иначе возвращается false.

Если свойства не проходят валидацию, то в консоли браузера отображается соответствующее предупреждение.

Применим эти параметры:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

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

    <user v-bind="user"></user>

</div>

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

<script>

Vue.component('user', {

  props: {

      name: {

        type: String,

        required: true,

        default: 'Tom',

        validator: function(value){

            return value!='admin' && value!='';

        }

      },

      age: {

        type: Number,

        required: true,

        default: 18,

        validator: function(value){

            return value >= 0 && value < 100;

        }

      }

  },

  template: '<div><h2>User</h2><p>Name: {{name}}</p><p>Age: {{age}}</p></div>'

})

new Vue({

    el: "#app",

    data: {

        user: {

            name: '',

            age: 0

        }

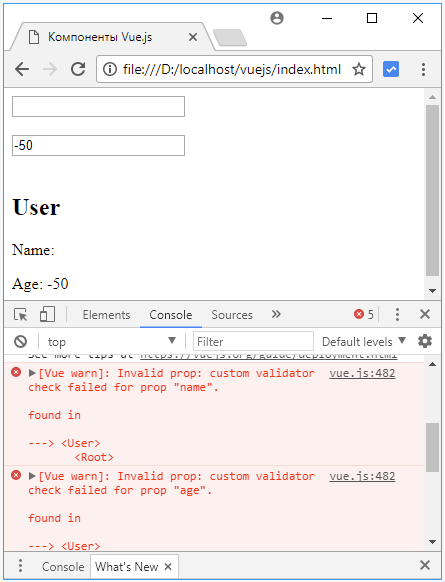
    }

});

</script>

</body>

</html>



* 1. 4. Передача в компонент объектов и массивов

Отдельно рассмотрим передачу в компонент через props сложных объектов и массивов. Например, передадим сложный объект:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

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

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

    <userinfo v-bind:user="user"></userinfo>

</div>

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

<script>

Vue.component('userinfo', {

  props: ["user"],  //как альтернатива  props: {user: Object}

  template: `<div>

                <h2>User</h2>

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

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

            </div>`

});

new Vue({

    el: "#app",

    data: {

        user: {

            name: 'Tom',

            age: 18

        }

    }

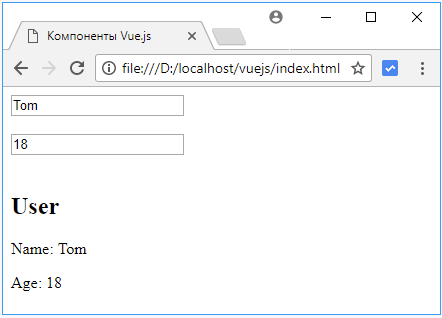
});

</script>

</body>

</html>

Здесь компонент получает данные в целом именно как один объект user. И после этого в шаблоне компонента можно обращаться к свойствам этого объекта через user.name и user.age.



Но если не передать значение для свойства user из props:

<userinfo></userinfo>

То в этом случае свойство user в компоненте будет не определено, и программа завершит свою работу с ошибкой. В этом случае можно предусмотреть значение по умолчанию для этого свойства с помощью параметра default. При этом для сложных объектов, а также для массивов этот параметр должен представлять функцию, которая возвращает начальное значение для свойства.

В частности, изменим компонент следующим образом:

Vue.component('userinfo', {

  props: {

        user: {

            type: Object,

            default: function(){

                return  {

                    name: 'Bob',

                    age: 22

                }

            }

        }

  },

  template: `<div>

                <h2>User</h2>

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

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

            </div>`

});

Теперь если извне не передается значение для props, программа будет использовать значение по умолчанию.

**Массивы**

Рассмотрим передачу массивов:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <userslist :users="users"></userslist>

</div>

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

<script>

Vue.component('userslist', {

  props: ["users"],

  template: `<ul>

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

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

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

                </li>

            </ul>`

});

new Vue({

    el: "#app",

    data: {

        users: [{

            name: 'Tom',

            age: 18

        },{

            name: 'Bob',

            age: 23

        },{

            name: 'Alice',

            age: 21

        }]

    }

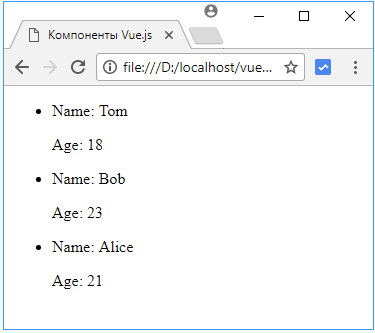
});

</script>

</body>

</html>

Здесь извне компоненту передается массив users, состоящий из сложных объектов. В компоненте можно с помощью директивы v-for пробежаться по всему массиву и получить отдельные его элементы.



Также можно выполнить более точную настройку свойства users, например, задать для него значение по умолчанию:

Vue.component('userslist', {

    props: {

        users:{

            type: Array,

            default: function(){

                return []

            }

        }

    },

    template: `<ul>

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

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

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

                </li>

            </ul>`

});

* 1. 5. Родительские и дочерние компоненты

Одни компоненты (родительские компоненты) могут содержать другие (дочерние компоненты). Например, один компонент выводит список объект, а для вывода отдельного объекта используется еще один компонент:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

<style>

.userdetails{

    border-bottom: 1px solid #888;

}

</style>

</head>

<body>

<div id="app">

    <userslist :users="users"></userslist>

</div>

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

<script>

Vue.component('userdetails', {

    props: ["user"],

    template: `<div class="userdetails">

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

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

                </div>`

});

Vue.component('userslist', {

    props: ["users"],

    template: `<div>

                <userdetails v-for="user in users" :key="user.name" :user="user"></userdetails>

            </div>`

});

new Vue({

    el: "#app",

    data: {

        users: [{

            name: 'Tom',

            age: 18

        },{

            name: 'Bob',

            age: 23

        },{

            name: 'Alice',

            age: 21

        }]

    }

});

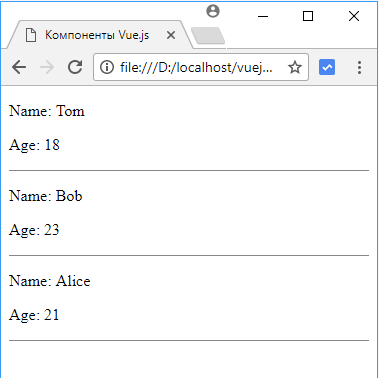
</script>

</body>

</html>

Здесь в компонент userslist передается список объектов users. Для вывода каждого отдельного объекта userslist использует еще один компонент userdetails. С помощью директивы v-for происходит перебор списка объектов, и каждый объект передается в компонент userdetails.

В принципе можно определить все в одном компоненте, однако выделение отдельного компонента userdetails позволяет развивать и обновлять его разметку отдельно от родительского компонента. Например, если потребуется изменить структуру разметки html в компоненте, то достаточно это сделать в коде компонента userdetails. К тому же можно повторно использовать данный компонент в других компонентах и частях программы.



**Получение состояния вложенного компонента**

С помощью ссылки this.$refs из объекта Vue можно ссылаться на различные элементы веб-станицы внутри объекта шаблона Vue. Но кроме того, подобным образом можно ссылаться также и на вложенные компоненты и, таким образом, обращаться к внутреннему состоянию компонента. Например, установим в объекте Vue изменение свойства, которое определено в компоненте:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <userdetails :user="user" ref="details"></userdetails>

    <button v-on:click="toggle()">Show</button>

</div>

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

<script>

Vue.component('userdetails', {

    props: ["user"],

    template: `<div>

                <h2>Информация о пользователе</h2>

                <div v-if="visible">

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

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

                </div>

            </div>`,

    data: function(){

        return{

            visible: false

        }

    }

});

new Vue({

    el: "#app",

    data: {

        user:{

            name: 'Tom',

            age: 18

        }

    },

    methods: {

        toggle: function(){

            this.$refs.details.visible = !this.$refs.details.visible;

        }

    }

});

</script>

</body>

</html>

Здесь для компонента userdetails с помощью атрибута ref установлена ссылка details, через которую можно ссылаться на данный компонент.

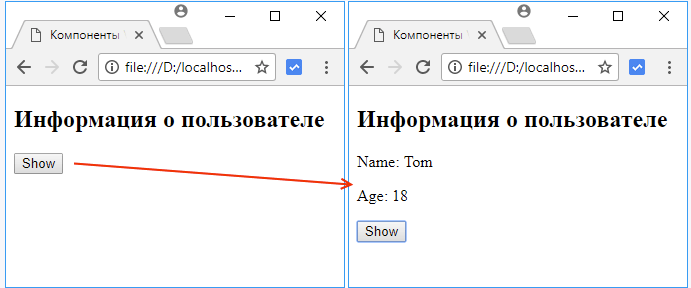
<userdetails :user="user" ref="details"></userdetails>

В самом компоненте userdetails определено свойство visible, которое управляет видимостью частью шаблона компонента. Для изменения значения этого свойства в объекте Vue предусмотрена кнопка, по нажатию на которую срабатывает метод toggle():

toggle: function(){

    this.$refs.details.visible = !this.$refs.details.visible;

}



**Однонаправленный поток данных**

При использовании props в компонентах следует учитывать, что данные в props представляют однонаправленный поток данных от родительского компонента к дочерним компонентам. Изменение свойства родителя приведет к изменению в дочерних компонентах. Однако дочерние компоненты не могут изменить свойство родителя. То есть поток данных идет только в одном направлении: родителя к потомкам. Кроме того, если изменить в дочерних компонентах значения, переданные через props, то на консоль браузера будет выведено предупреждение о том, что это не следует делать.

Более того при любом обновлении родительского компонента у дочерних компонентов обновляются значения, передаваемые через props. Например, рассмотрим следующую ситуацию:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>Hello, {{name}}</h2>

    <useredit :user="name"></useredit>

    <button v-on:click="resetName">Set Name</button>

</div>

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

<script>

Vue.component('useredit', {

    props: ["user"],

    template: '<div><input type="text" v-model="user" /><p>Name: {{user}}</p></div>'

});

new Vue({

    el: "#app",

    data: {

        name: 'Tom'

    },

    methods:{

        resetName: function(){

            this.name = 'Bob';

        }

    }

});

</script>

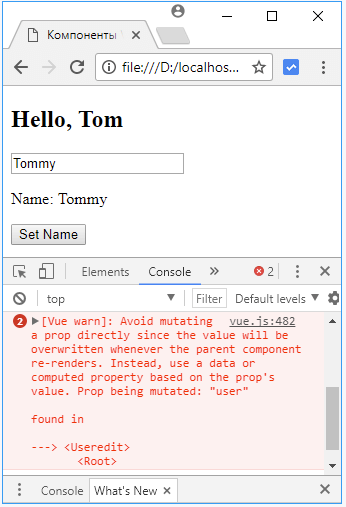
</body>

</html>

Из родителя – объекта Vue в компонент useredit через props передается значение name. В этом компоненте можно попробовать управлять переданным значением через поле ввода и механизм двусторонней привязки:

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

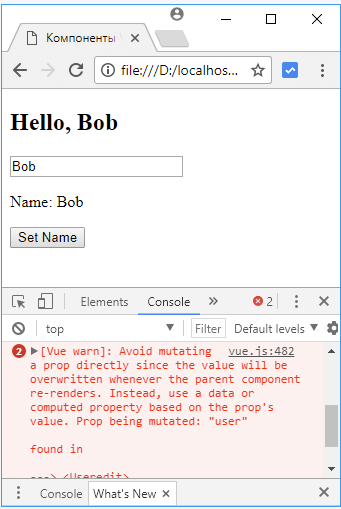
Но если изменить через поле ввода данное значение, то оно изменится только в рамках компонента useredit, а на консоль будет выведено предупреждение:



Значение свойства name в родительском объекте Vue никак не будет затронуто. Так происходит потому, что данные простых типов – String, Number, Boolean передаются по значению, то есть в дочерний компонент передается копия значения. И в данном случае в компонент useredit как раз передается копия строки name. Соответственно все действия с этой копией никак не повлияют на родительский объект.

В отличие от простых типов сложные объекты и массивы передаются по ссылке. Поэтому из дочерних компонентов нельзя изменять массивы или сложные объекты, которые определены в родительских объектах.

Более того в объекте Vue посредством нажатия кнопки нельзя вызвать метод resetName, который изменит свойство name с Tom на Bob:



То есть несмотря на то, что внутри компонента useredit меняется значение из props, но изменение родителя – объекта Vue приведет к изменению дочернего компонента.

Однако могут быть ситуации, когда все-таки нужно изменить значения в props и при этом не допустить изменения этих значений извне. В этом случае можно определить локальную переменную. Через props устанавливается ее начальное значение, и затем компонент работает только с этой переменной:

<div id="app">

    <h2>Hello, {{name}}</h2>

    <useredit :user="name"></useredit>

    <button v-on:click="resetName">Set Name</button>

</div>

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

<script>

Vue.component('useredit', {

    props: ["user"],

    data: function () {

      return { userName: this.user}

    },

    template: '<div><input type="text" v-model="userName" /><p>Name: {{userName}}</p></div>'

});

new Vue({

    el: "#app",

    data: {

        name: 'Tom'

    },

    methods:{

        resetName: function(){

            this.name = 'Bob';

        }

    }

});

</script>

Теперь все действия будут идти с локальной переменной userName. Значения из props в компоненте не будут изменяться, а изменения в родительском компоненте никак не повлияют на значение переменной userName.

* 1. 6. Создание событий

Через props можно передать данные от родительского компонента в дочерний. Если надо передавать данные и в обратном направлении: от дочернего компонента к родителю, то необходимо определить свои события. Дочерний компонент будет генерировать событие с помощью вызова this.$emit(имя\_события), а родительский компонент будет отлавливать это событие с помощью установки атрибута v-on:название\_события и при получении события производить определенные действия. Например:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>Hello, {{name}}</h2>

    <useredit :user="name" v-on:userchange="change"></useredit>

</div>

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

<script>

Vue.component('useredit', {

    props: ["user"],

    data: function () {

      return { userName: this.user}

    },

    template: '<div><input type="text" v-model="userName" v-on:input="onUserChange" /><p>Name: {{userName}}</p></div>',

    methods: {

        onUserChange: function(){

            this.$emit('userchange', this.userName);

        }

    }

});

new Vue({

    el: "#app",

    data: {

        name: 'Tom'

    },

    methods:{

        change: function(value){

            this.name = value;

        }

    }

});

</script>

</body>

</html>

В данном случае определяется событие userchange. В реальности в JavaScript не существует подобного события, мы его определяем сами.

В шаблоне компонента элемент ввода связан двусторонней привязкой со значением со свойством userName. Также для этого элемента установлен обработчик события input:

<input type="text" v-model="userName" v-on:input="onUserChange" />

То есть при вводе в это текстовое поле будет вызываться метод onUserChange. В этом методе через вызов this.$emit('userchange', this.userName); генерируем событие userchange и передаем в этом событии измененное значение свойства userName (так как это свойство связано двусторонней привязкой с полем ввода, то при вводе в текстовое поле изменится и свойство user).

onUserChange: function(){

    this.$emit('userchange', this.userName);

}

Чтобы отловить это событие в объекте Vue на дочернем компоненте определен атрибут v-on:userchange:

<useredit :user="name" v-on:userchange="change"></useredit>

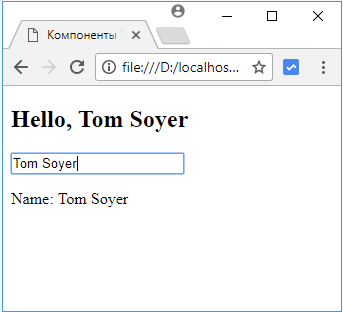
То есть поскольку в дочернем компоненте генерируется событие userchange, то соответственно здесь и атрибут имеет определение v-on:userchange. Этому событию сопоставлен метод change:

change: function(value){

    this.name = value;

}

Параметр value представляет то значение, которое передается в дочернем компоненте через вызов this.$emit('userchange', this.userName);, то есть this.userName. В итоге при вводе данных в дочернем компоненте useredit сработает обработчик события input – метод onUserChange, который сгенерирует событие namechange, которое в свою очередь будет обработано в родительском объекте в методе change.



**Модификатор sync**

Начиная с версии 2.3 был добавлен модификатор sync, который позволяет в ряде случаев сократить код использования событий. В частности, перепишем предыдущий пример следующим образом:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>Hello, {{name}}</h2>

    <useredit :user.sync="name"></useredit>

</div>

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

<script>

Vue.component('useredit', {

    props: ["user"],

    data: function () {

      return { userName: this.user}

    },

    template: '<div><input type="text" v-model="userName" v-on:input="onUserChange" /><p>Name: {{userName}}</p></div>',

    methods: {

        onUserChange: function(){

            this.$emit('update:user', this.user)

        }

    }

});

new Vue({

    el: "#app",

    data: {

        name: 'Tom'

    }

});

</script>

</body>

</html>

В данном случае строка

<useredit :user.sync="name"></useredit>

будет сокращенем строки:

<useredit :user="name" v-on:update:user="val => name = val"></useredit>

Но в дочернем компоненте также необходимо вызывать данное событие:

onUserChange: function(){

    this.$emit('update:user', this.userName)

}

* 1. 7. Изменение внешних массивов и объектов в компонентах

В отличие от строк, чисел, логических значений true и false массивы и сложные объекты передаются в компоненты по ссылке. То есть получив ссылку, можно манипулировать значением, которое хранится по этой ссылке.

**Изменение массивов**

Определим следующую веб-станицу:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>Список пользователей</h2>

    <userform :users="users"></userform>

    <div>

        <useritem v-for="(user, index) in users"

                    :user="user"

                    :index="index"

                    :key="index"

                    v-on:userdelete="remove">

        </useritem>

    </div>

</div>

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

<script>

Vue.component('userform', {

    props: ["users"],

    data: function () {

        return {

            user: {name:'', age:18}

        }

    },

    template: `<div>

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

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

                    <button  v-on:click="userAdd">Add</button>

                </div>`,

    methods: {

        userAdd: function(event){

            this.users.push({name:this.user.name, age: this.user.age});

        }

    }

});

Vue.component('useritem', {

    props: ["user", "index"],

    template: `<div>

                    <p>Name: {{user.name}} <br> Age: {{user.age}}</p>

                    <button  v-on:click="userDelete(index)">Delete</button>

                </div>`,

    methods: {

        userDelete: function(index){

            this.$emit('userdelete', index);

        }

    }

});

new Vue({

    el: "#app",

    data: {

        users:[

            {name: 'Tom', age: 23},

            {name: 'Bob', age: 26},

            {name: 'Alice', age: 28}

        ]

    },

    methods:{

        remove: function(index){

            this.users.splice(index, 1)

        }

    }

});

</script>

</body>

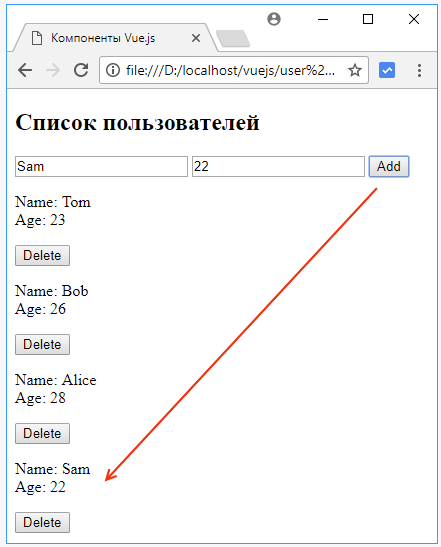
</html>

Для добавления нового элемента в массив определен компонент userform, который представляет форму добавления. Для вывода одного элемента из массива определен компонент useritem, который также содержит кнопку для удаления элемента.

По сути здесь используются две различные техники для работы с массивами. Компонент userform получает ссылку на массив и при нажатии на кнопку вызывает метод userAdd(), в котором через полученную ссылку в массив добавляется новый элемент.

Компонент useritem использует другую технику. Он получает элемент для вывода и по нажатию на кнопку удаления генерирует событие userdelete, передавая с ним индекс удаляемого элемента. Родительский объект получает это событие с индексом и по индексу собственно удаляет элемент из массива.

То есть можно либо напрямую передавать ссылку в дочерний компонент и в нем изменять значения в массиве, либо генерировать в дочернем компоненте событие, которое передается родительскому объекту.



**Изменение объектов**

Для изменения внешних объектов в компоненте можно использовать аналогичные техники. В частности, изменим страницу следующим образом:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>Список пользователей</h2>

    <userform :user="user"></userform>

    <div>

        <useritem v-for="(user, index) in users"

                    :key="index"

                    :user="user"

                    :index="index"

                    v-on:userchange="change">

        </useritem>

    </div>

</div>

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

<script>

Vue.component('userform', {

    props: ["user"],

    template: `<div>

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

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

                </div>`,

});

Vue.component('useritem', {

    props: ["user", "index"],

    template: `<div>

                    <p>Name: {{user.name}} <br> Age: {{user.age}}</p>

                    <button  v-on:click="userChange(index)">Change</button>

                </div>`,

    methods: {

        userChange: function(index){

            this.$emit('userchange', index);

        }

    }

});

new Vue({

    el: "#app",

    data: {

        users:[

            {name: 'Tom', age: 23},

            {name: 'Bob', age: 26},

            {name: 'Alice', age: 28}

        ],

        user:{}

    },

    methods:{

        change: function(index){

            this.user = this.users[index];

        }

    }

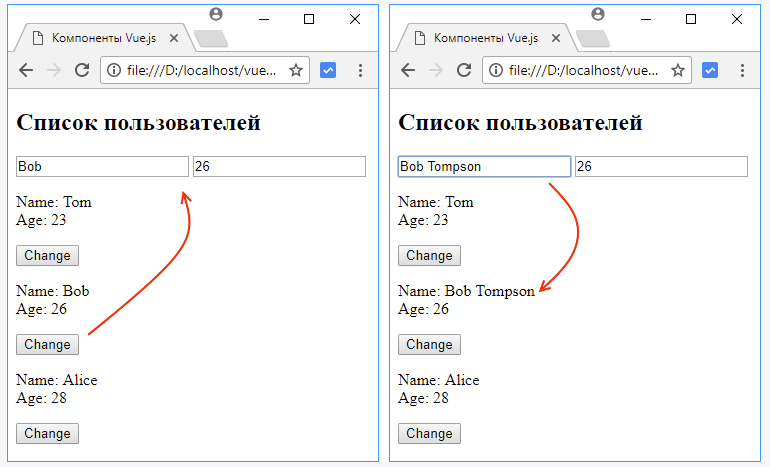
});

</script>

</body>

</html>

Здесь компонент userform через props получает объект user, свойства которого связываются двухсторонней привязкой с полями ввода. При изменении введеных значений в этих полях изменится и значение в массиве:



**Передача функций обратного вызова в компоненты**

Функции обратного вызова представляют еще один способ взаимодействия между родительским и дочерним компонентами: родительские компоненты могут определять функции, а вызываются они в дочерних компонентах. Например, определим следующую веб-страницу:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>Список пользователей</h2>

    <userform :addfn="add"></userform>

    <div>

        <useritem v-for="(user, index) in users"

                    :user="user"

                    :key="index"

                    :index="index"

                    :removefn="remove">

        </useritem>

    </div>

</div>

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

<script>

Vue.component('userform', {

    props: ["addfn"],

    data: function () {

        return {

            user: {name:'', age:18}

        }

    },

    template: `<div>

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

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

                    <button  v-on:click="addfn({name:user.name, age: user.age})">Add</button>

                </div>`

});

Vue.component('useritem', {

    props: ["user", "index", "removefn"],

    template: `<div>

                    <p>Name: {{user.name}} <br> Age: {{user.age}}</p>

                    <button  v-on:click="removefn(index)">Delete</button>

                </div>`

});

new Vue({

    el: "#app",

    data: {

        users:[

            {name: 'Tom', age: 23},

            {name: 'Bob', age: 26},

            {name: 'Alice', age: 28}

        ]

    },

    methods:{

        remove: function(index){

            this.users.splice(index, 1);

        },

        add: function(user){

            this.users.push(user);

        }

    }

});

</script>

</body>

</html>

Объект Vue выводит на страницу массив элементов и определяет два метода для управления элементами: add (для добавления) и remove (для удаления).

Для добавления нового элемента определен компонент userform, который представляет форму с полями ввода. В этот компонент передается метод add в виде функции addfn:

<userform :addfn="add"></userform>

Причем функция addfn определена в компоненте userform через props:

Vue.component('userform', {

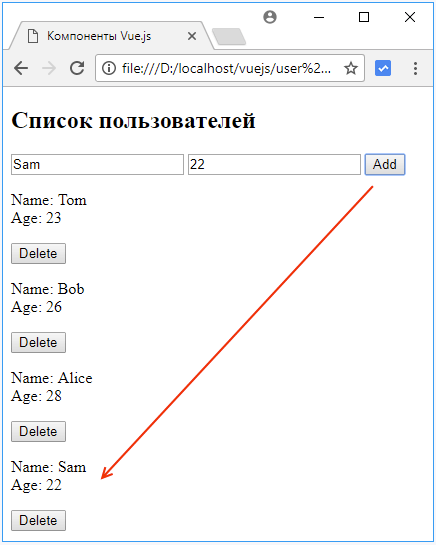
    props: ["addfn"],

// …

При нажатии на кнопку эта функция будет вызываться, и ей будут передаваться введенные данные. Что фактически приведет к вызову метода add в родительском объекте Vue.

Аналогичная ситуация с компонентом useritem, который получает метод remove в виде функции removefn, которая также определяется через props. При нажатии на кнопку удаления вызывается функция removefn, ей передается индекс удаляемого элемента, и фактически будет идти вызов метода remove из объекта Vue.

Таким образом, в самих дочерних компонентах никаких действий по управлению массивом и объектами и по генерации событий не определено. Дочерние компоненты просто вызывают те методы, которые определены в родительском объекте.



* 1. 8. Сестринские компоненты

Во Vue.js также можно организовать взаимодействие между компонентами, которые находятся на одном уровне, то есть являются сестринскими. В этом случае взаимодействие происходит через родительский компонент. Например, определим два дочерних компонента, которые взаимодействуют через родительский компонент:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>User</h2>

    <useredit :name="user" @userchange="change"></useredit>

    <userinfo :name="user" @userreset="reset"></userinfo>

</div>

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

<script>

Vue.component('useredit', {

    props: ["name"],

    template: '<div><input v-model="name" /><button @click="save">Save</button></div>',

    methods:{

        save(){

            this.$emit("userchange", this.name);

        }

    }

});

Vue.component('userinfo', {

    props: ["name"],

    template: `<div><p>Имя: {{name}}</p><button @click="reset">Reset</button></div>`,

    methods:{

        reset(){

            this.$emit("userreset");

        }

    }

});

new Vue({

    el: "#app",

    data:{

        user:'Tom'

    },

    methods:{

        change(name){

            this.user = name;

        },

        reset(){

            this.user = 'Tom';

        },

    }

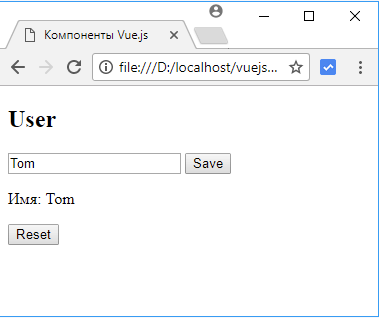
});

</script>

</body>

</html>

В данном случае оба дочерних компонента принимают от объекта Vue имя пользователя. То есть оба компонента используют те данные, которые определены в объекте Vue. Если один компонент хочет изменить эти данные, то он генерирует событие. Объект Vue обрабатывает это событие и изменяет данные. После этого данные автоматически изменяются во втором компоненте. То есть объект Vue здесь служит медиатором, а непосредственно между собой компоненты не взаимодействуют.



**Шина событий**

Еще один способ взаимодействия между компонентами одного уровня представляет применение шины событий. Ее суть заключается, в определении объекта Vue, который будет выполнять роль шины событий:

let eventBus = new Vue();

В компоненте, который хочет изменить данные, вызывается через эту шину событие:

eventBus.$emit("событие", данные);

А в другом компоненте в методе created устанавливаем обработчик этого события:

created(){

    eventBus.$on("событие", (данные)=>{

        // действия

    });

}

Например, рассмотрим следующий пример:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <h2>User</h2>

    <useredit :user="user"></useredit>

    <userinfo :user="user"></userinfo>

</div>

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

<script>

let eventBus = new Vue();

Vue.component('useredit', {

    props: ["user"],

    template: `<div>

                    <input v-model="userName" />

                    <button @click="save">Save</button>

                </div>`,

    data: function () {

      return { userName: this.user}

    },

    methods:{

        save(){

            eventBus.$emit("userchange", this.userName);

        }

    }

});

Vue.component('userinfo', {

    props: ["user"],

    template: '<div><p>Имя: {{userName}}</p></div>',

    data: function () {

      return { userName: this.user}

    },

    created(){

        eventBus.$on("userchange", (name)=>{

            this.userName = name;

        });

    }

});

new Vue({

    el: "#app",

    data:{

        user:'Tom'

    }

});

</script>

</body>

</html>

Здесь объект Vue не обрабатывает никаких событий и просто передает в компоненты имя пользователя. В самих компонентах привязка идет не к свойству из props, а к свойству из параметра data. То есть в данном случае даже если бы надо было изменить значения свойств через обработку событий в объекте Vue, то нельзя было бы это сделать. Однако благодаря шине событий это все же можно сделать. Компонент useredit вызывает через шину событий событие userchange, а второй компонент – userinfo обрабатывает это событие.

* 1. 9. Миксины

Миксины представляют способ передачи функциональности из одной сущности в другую, аналогичный наследованию. Миксин может содержать любые свойства. Если компонент использует миксин, то все свойства миксина добавляются к свойствам данного компонента. Допустим есть два компонента:

<!DOCTYPE html>

<html>

<head>

<title>Миксины Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <light></light>

    <enabled></enabled>

</div>

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

<script>

Vue.component('light', {

  template: `<div><h3>{{header}}</h3>

                <button v-on:click="toggle" v-show="state">{{on}}</button>

                <button v-on:click="toggle" v-show="!state">{{off}}</button>

                </div>`,

  data: function(){

    return { state: true, on: "ON", off: "OFF"}

  },

  methods:{

    toggle: function(){

        this.state= !this.state;

    }

  },

  computed:{

    header(){

        return this.state==true?"Свет включен":"Свет выключен";

    }

  }

});

Vue.component('enabled', {

  template: `<div>Включить функцию

                <button v-on:click="toggle" v-show="state">{{on}}</button>

                <button v-on:click="toggle" v-show="!state">{{off}}</button>

                </div>`,

  data: function(){

    return { state: true, on: "ON", off: "OFF"}

  },

  methods:{

    toggle: function(){

        this.state= !this.state;

    }

  }

});

new Vue({

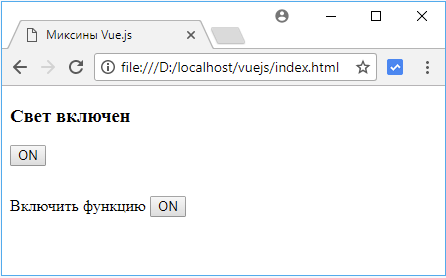
    el: "#app"

});

</script>

</body>

</html>



Здесь определено два компонента. Функциональность обоих компонентов в некоторых моментах сходится. И весь этот общий функционал можно вынести в миксин:

<div id="app">

    <light></light>

    <enabled></enabled>

</div>

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

<script>

let toggleMixin = {

    data: function(){

        return { state: true, on: "ON", off: "OFF"}

    },

    methods:{

        toggle: function(){

            this.state= !this.state;

        }

    },

};

Vue.component('light', {

    template: `<div><h3>{{header}}</h3>

                <button v-on:click="toggle" v-show="state">{{on}}</button>

                <button v-on:click="toggle" v-show="!state">{{off}}</button>

                </div>`,

    mixins:[toggleMixin],

    computed:{

        header(){

            return this.state==true?"Свет включен":"Свет выключен";

        }

    }

});

Vue.component('enabled', {

  template: `<div>Включить функцию

                <button v-on:click="toggle" v-show="state">{{on}}</button>

                <button v-on:click="toggle" v-show="!state">{{off}}</button>

                </div>`,

    mixins:[toggleMixin]

});

new Vue({

    el: "#app"

});

</script>

Теперь общий код вынесен в отдельный объект – миксин toggleMixin. С помощью параметра mixins миксин передается компоненту:

mixins:[toggleMixin]

Данный параметр представляет массив, поэтому можно передать несколько миксинов. В итоге функциональность миксина будет объединяться с функциональностью, которая определена непосредственно в компоненте. А результат работы кода будет тот же самый, что и в предыдущем примере. И если потребуется изменить логику нажатия кнопок обоих компонентов, достаточно это будет сделать в одном месте – в миксине.

Следует отметить, что если миксины и компоненты определяют методы и свойства с одним и тем же именем, то подобная функциональность миксинов отбрасывается. Но в данном случае компоненты содержат также частично повторяемый шаблон – две кнопки. И установку шаблона можно также перенести в миксин. Для этого воспользуемся событие жизненного цикла created, которое вызывается после создания компонента и миксина.

При этом стоит учитывать, что вначале вызываются события жизненного цикла миксина, а потом события жизненного цикла компонента, который использует данный миксин. В частности, перепишем код миксина и компонентов:

let toggleMixin = {

    data: function(){

        return { state: true, on: "ON", off: "OFF"}

    },

    methods:{

        toggle: function(){

            this.state= !this.state;

        }

    },

    created () {

        console.log("Mixin has been created");

        this.$options.template = `<div>` + this.$options.template +

            `<button v-on:click="toggle" v-show="state">{{on}}</button>

            <button v-on:click="toggle" v-show="!state">{{off}}</button></div>`

    }

};

Vue.component('light', {

    template: `<h3>{{header}}</h3>`,

    mixins:[toggleMixin],

    computed:{

        header(){

            return this.state==true?"Свет включен":"Свет выключен";

        }

    },

    created(){

        console.log("Light component has been created");

    }

});

Vue.component('enabled', {

    template: `Включить функцию`,

    mixins:[toggleMixin],

    created(){

        console.log("enabled component has been created");

    }

});

С помощью значения this.$options.template можно получить в миксине шаблон компонента и изменить его. В итоге результат работы компонентов будет тот же, что и ранее.

**Глобальные миксины**

Vue.js позволяет задавать миксины глобально, то есть для всех компонентов. Для этого используется функция Vue.mixin:

<div id="app">

    <message></message>

</div>

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

<script>

Vue.component('message', {

    template: `<h2>Hello</h2>`

});

Vue.mixin({

  created: function () {

    console.log("Global mixin has been created");

  }

})

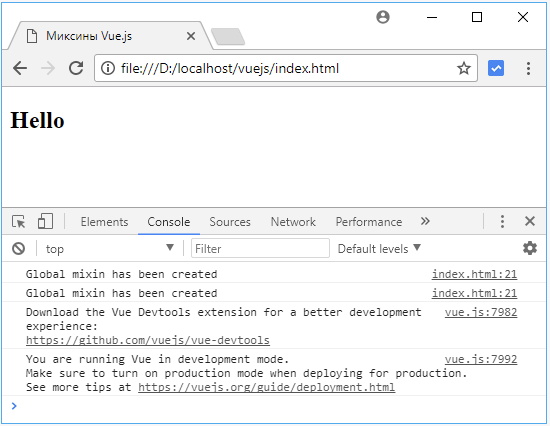
new Vue({

    el: "#app"

});

</script>

В данном случае глобальный миксин использует событие created и выводит на консоль браузера строку:



Поскольку в данном случае определен один компонент и объект Vue, то глобальный миксин будет вызываться два раза.

* 1. 10. Именованные слоты

Слоты представляют способ создания фиксированной структуры компонента, при котором содержимое для различных частей компонента определяет родительский компонент. Во Vue.js слоты реализуются через элемент <slot>, вместо которого родительский компонент вставляет содержимое в дочерний компонент.

Рассмотрим следующий пример:

<!DOCTYPE html>

<html>

<head>

<title>Slots in Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <ads></ads>

    <ads>

        <div>

            <p>Открыт набор на бесплатные курсы Udacity.</p>

            <p>Заявки принимаются до 16 октября</p>

        </div>

    </ads>

</div>

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

<script>

Vue.component('ads', {

  template: `<div>

               <h3>Объявление</h3>

               <slot>Здесь могла бы ваша реклама</slot>

             </div>`

});

new Vue({

  el: '#app'

});

</script>

</body>

</html>

Здесь определен компонент ads, который представляет блок под рекламное объявление. Само рекламное объявление будет добавляться в компонент через элемент <slot>. При этом на момент определения компонента может быть неизвестно будет ли предоставляться какое-либо содержимое. Поэтому можно определить в элементе <slot> некоторое содержимое по умолчанию.

<slot>Здесь могла бы ваша реклама</slot>

При использовании компонента можно вложить в него некоторый контент, который будет вставляться в элемент <slot> и заменять в нем содержимое по умолчанию:

<div id="app">

    <ads></ads>

    <ads>

        <div>

            <p>Открыт набор на бесплатные курсы Udacity.</p>

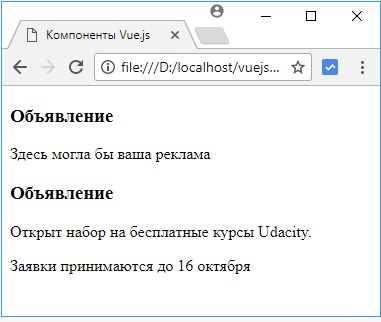
            <p>Заявки принимаются до 16 октября</p>

        </div>

    </ads>

</div>

Причем в первом случае в компонент ads не передается никакого контента, поэтому первый компонент ads будет использовать для slot содержимое по умолчанию. Второй же компонент ads принимает контент и заменяет им содержимое в элементе <slot>.



**Именованные слоты**

Каждому слоту можно назначить определенное имя и через это имя передавать в него некоторое содержимое из родительского компонента. Для этого в элементе <slot> используется атрибут name. Например, определим следующую веб-страницу:

<!DOCTYPE html>

<html>

<head>

<title>Slots in Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <container>

        <h1 slot="header">Заголовок статьи</h1>

        <p slot="header">Дата статьи: 13/10/2017</p>

        <p>Первый абзац.</p>

        <p>Второй абзац.</p>

        <p slot="footer">Футер</p>

     </container>

</div>

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

<script>

Vue.component('container', {

    template: `<div>

                <header>

                    <slot name="header">Заголовок по умолчанию</slot>

                </header>

                <article>

                    <slot></slot>

                </article>

                <footer>

                    <slot name="footer"></slot>

                </footer>

            </div>`

});

new Vue({

  el: '#app'

});

</script>

</body>

</html>

Компонент container определяет ряд слотов. Причем один из слотов является обычным, а остальные два – именованными. Именованные слоты как и обычные могут иметь содержимое по умолчанию:

<div>

    <header>

        <slot name="header">Заголовок по умолчанию</slot>

    </header>

    <article>

        <slot></slot>

    </article>

    <footer>

        <slot name="footer"></slot>

    </footer>

</div>

В родительском компоненте для слотов передается содержимое, которое также помещается внутрь дочернего компонента. Причем для передачи контента для именованных слотов html-элементы должны определять атрибут slot="имя\_слота". Этому атрибуту передается имя слота.

<div id="app">

    <container>

        <h1 slot="header">Заголовок статьи</h1>

        <p slot="header">Дата статьи: 13/10/2017</p>

        <p>Первый абзац.</p>

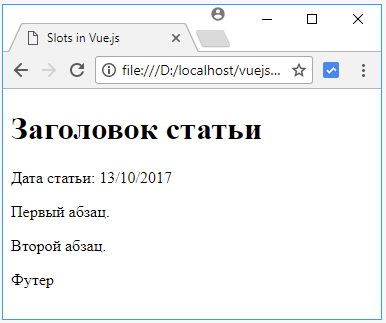
        <p>Второй абзац.</p>

        <p slot="footer">Футер</p>

     </container>

</div>

Те элементы, которые не имеют подобного атрибута, будут передаваться в качестве содержимого для неименованного обычного слота.



* 1. 11. Слоты с ограниченной областью видимости

Слоты с ограниченной областью видимости (scoped slots) позволяют передавать данные между слотом и родительским компонентом. Подобные слоты определяют шаблон, в который можно передавать данные из дочернего компонента. И эти данные ограничены слотом.

Для определения шаблона в родительском компоненте применяется какой-нибудь элемент, как правило, это элемент <template>, в котором устанавливается специальный атрибут slot-scope. Значение этого атрибута – название объекта с данными, которые передаются от дочернего компонента.

Например, определим следующую веб-страницу:

<!DOCTYPE html>

<html>

<head>

<title>Slots in Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

    <user>

        <template slot-scope="props">

          <h3>Данные о пользователе</h3>

          <p>Имя: {{ props.username }}</p>

          <p>Имя: {{ props.userage }}</p>

        </template>

     </user>

</div>

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

<script>

Vue.component('user', {

    template: `<div class="user">

                    <slot username="Tom" userage="31"></slot>

            </div>`

});

new Vue({

  el: '#app'

});

</script>

</body>

</html>

Для слота с ограниченной областью видимости здесь определен шаблон:

<template slot-scope="props">

    <h3>Данные о пользователе</h3>

    <p>Имя: {{ props.username }}</p>

    <p>Имя: {{ props.userage }}</p>

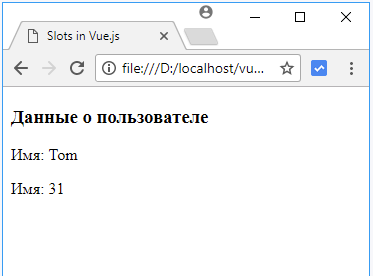
</template>

К примеру, дочерний компонент вывод некоторые данные о пользователе. И через элемент template эти данные будут выводиться в слот. Для получения данные в элементе определен атрибут slot-scope="props". Props – это тот объект с данными, который передается от дочернего компонента. Внутри шаблона можно обращаться через props к переданным данным. Для определения данных в дочернем компоненте устанавливается элемент <slot> с соответствующими атрибутами:

<div class="user">

    <slot username="Tom" userage="31"></slot>

</div>



Другой показательный пример представляет использование списков, где родительский компонент может определить шаблон для элемента списка:

<!DOCTYPE html>

<html>

<head>

<title>Slots in Vue.js</title>

<meta charset="utf-8" />

<style>

.user{

    font-size:14px;

    font-family: Verdana;

    padding:6px;

}

</style>

</head>

<body>

<div id="app">

    <userslist :users="users">

     <li slot-scope="props" class="user">

        {{ props.username }}

      </li>

    </userslist>

</div>

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

<script>

Vue.component('userslist', {

    props: ["users"],

    template: `<ul>

                <slot v-for="user in users" :username="user"></slot>

            </ul>`

});

new Vue({

  el: '#app',

  data:{

    users: ['Tom', 'Sam', 'Bob']

  }

});

</script>

</body>

</html>

В данном случае компонент userslist получает извне список users. Для каждого элемента из этого списка создается свой элемент slot:

<slot v-for="user in users" :username="user"></slot>

Для слота определяется одно свойство – username.

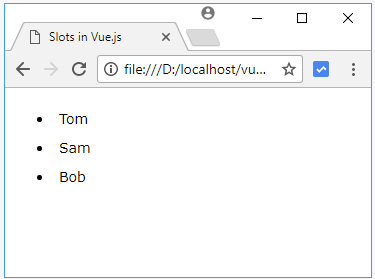
В родительском компоненте определяется шаблон, в котором выводится значение свойсва username:

<li slot-scope="props" class="user">

        {{ props.username }}

</li>

Причем в данном случае родительский компонент сам определяет шаблон для элементов списка и правила их стилизации.



* 1. 12. Фильтры

Фильтры позволяют отформатировать текст определенным образом перед его выводом на веб-страницу, то есть осуществляют предобработку значений. Фильтры могут быть глобальными, то есть использоваться для любого компонента в приложении. И также фильтры могут быть локальными, то есть определенными только для одного компонента.

**Глобальные фильтры**

Для определения глобального фильтра во Vue.js вызывается метод Vue.filter, который имеет следующий формальный синтаксис:

Vue.filter(название\_фильтра, function (oldValue) {

    // преобразование из oldValue в newValue

    return newValue

})

Первый параметр представляет название фильтра. Второй параметр – функция, которая в качестве параметра принимает обрабатываемое значение. В самой функции некоторый образом обрабатывается это значение, из него создается новое значение, которое затем возвращается в качестве результата.

После определения фильтра его можно применить для форматирования какого-либо определенного значения. Это можно сделать в выражении интерполяции:

{message | filter}

Или в выражениях атрибута v-bind:

v-bind="message | filter"

Например, создадим фильтр, который будет делать первую букву заглавной:

<!DOCTYPE html>

<html>

<head>

<title>Компоненты Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

  {{message | capitalize }}

</div>

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

<script>

Vue.filter('capitalize', function (value) {

          let capitalFirst = value.charAt(0).toUpperCase()

          let noCaseTail = value.slice(1, value.length)

            return capitalFirst + noCaseTail;

});

new Vue({

    el:'#app',

    data: {

        message: "hello word!"

    }

})

</script>

</body>

</html>

В данном случае фильтр называется capitalize. Функция в фильтре вырезает первую букву переданного слова, делает ее заглавной и соединяет с остальной частью слова.

Для применения фильтра после определенного значения ставится прямая вертикальная черта, после которой идет название фильтра:

{{message | capitalize }}

**Локальные фильтры**

Локальные фильтры определяются в компоненте через свойство filters:

<!DOCTYPE html>

<html>

<head>

<title>Фильтры Vue.js</title>

<meta charset="utf-8" />

</head>

<body>

<div id="app">

  <h2>{{header | uppercase }}</h2>

  <message></message>

</div>

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

<script>

Vue.component('message', {

    template: '<h2>{{text | uppercase}}</h2>',

    data: function(){

        return {

            text: 'vue filters'

        }

    },

    filters: {

        uppercase(value) {

            return value.toUpperCase();

        }

    }

});

new Vue({

    el:'#app',

    data: {

        header: "hello word!"

    }

})

</script>

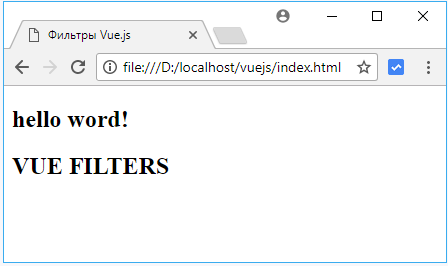
</body>

</html>

Здесь компонент message имеет параметр filters, который представляет объект. В этом объекте фильтры определяются в виде функцию, которые возвращают отформатированное значение. В данном случае определен только один фильтр uppercase, который просто возвращает строку с символами в верхнем регистре. И внутри компонента можно применять данный фильтр. Однако вне компонента нельзя его применить, так как фильтр локальный. И в частности, применение фильтра в следующей строке:

<h2>{{header | uppercase }}</h2>

работать не будет. И браузер в консоли выведет предупреждение, хотя ошибки в этом случае также не будет.



# 