Компоненты

СОЗДАНИЕ КОМПОНЕНТОВ

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 глобального компонента используется функция
Vue.component(tagName, options),
где параметр
tagName - кастомный элемент html, который будет представлять компонент,
options - представляет конфигурацию компонента.

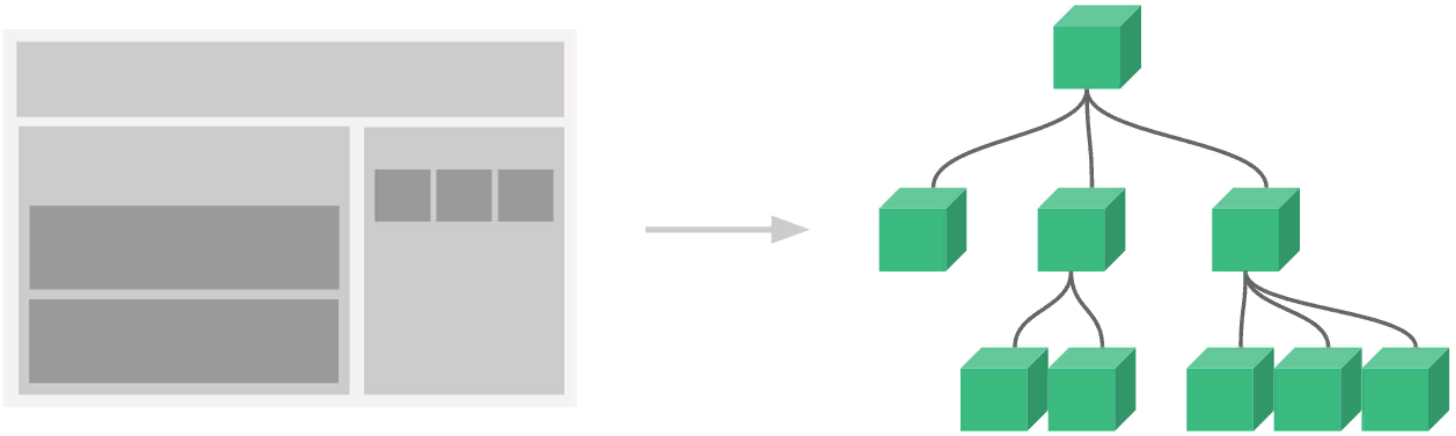
СОЗДАНИЕ КОМПОНЕНТОВ

Пример создания и использования простейшего компонента:

```
<div id="app">
     <hello></hello>
     <hello></hello>
</div>
<script>
      Vue.component('hello', {
            template: '<h2>Hello</h2>'
      });
      new Vue({
            el: "#app"
      });
</script>
```

ОРГАНИЗАЦИЯ КОМПОНЕНТОВ

Обычно приложение организуется в виде дерева вложенных компонентов (одни компоненты выступают родителями для других компонентов).



Например, у вас могут быть компоненты для заголовка, боковой панели, зоны контента, каждый из которых может содержать другие компоненты для навигационных ссылок, постов блога и т.д.

ЛОКАЛЬНАЯ И ГЛОБАЛЬНАЯ РЕГИСТРАЦИЯ КОМПОНЕНТОВ

Чтобы использовать эти компоненты в шаблонах, они должны быть зарегистрированы.

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**.

Глобальный компонент определяется с помощью метода **Vue.component()**.

ЛОКАЛЬНАЯ И ГЛОБАЛЬНАЯ РЕГИСТРАЦИЯ КОМПОНЕНТОВ

```
<div id="app1">
    <section-header></section-header>
    <section-content></section-content>
    <section-footer></section-footer>
</div><hr/>
<script>
   Vue.component('section-header',{
       template:'<h2>Header</h2>'
   });
   new Vue({
       el: "#app1",
       components:{
         'section-content':{ template:'<div>Content 1</div>'},
         'section-footer':{ template:'<p><b>Footer</b></p>'}
       }
   });
</script>
```

ОБХОД ОГРАНИЧЕНИЙ ПРИ ИСПОЛЬЗОВАНИИ DOM

Использование компонентов в рамках DOM связано со специфическими ограничениями. Некоторые HTML-теги, такие как <ul>, <ol>, <table> и <select> могут содержать только строго определенные элементы. Решить это ограничение поможет атрибут is.

Некорректный документ:

```
<div id="app">
    <table>
        <blog-post-row></blog-post-row>
    </table>
</div>
```

Корректный документ:

```
<div id="app">
    <table>
        <tr is="blog-post-row"></tr>
    </table>
</div>
```

ДИНАМИЧЕСКИЕ КОМПОНЕНТЫ

Иногда бывает полезно динамически переключаться между компонентами – это возможно с помощью элемента Vue <component> с привязкой к специальному атрибуту is.

Пример:

```
<div id="app">
    <button v-for="tab in tabs"
      v-bind:key="tab"
      v-on:click="currentTab = tab">
        {{ tab }}
    </button>
    <component v-bind:is="currentTabComponent"></component>
</div>
```

ДИНАМИЧЕСКИЕ КОМПОНЕНТЫ

Пример (продолжение):

```
<script>
    Vue.component('tab-home', { template:'<div>Компонент Home</div>' })
    Vue.component('tab-posts', { template: '<div>Компонент Posts</div>' })
    Vue.component('tab-archive', {
        template: '<div>Компонент Archive</div>'
    })
    new Vue({
        el: '#app',
        data: {
            currentTab: 'Home',
            tabs: ['Home', 'Posts', 'Archive']
        },
        computed: {
            currentTabComponent: function () {
                return 'tab-' + this.currentTab.toLowerCase()
            }
        }
    })
</script>
```

НАЗВАНИЕ КОМПОНЕНТ

Принятый стиль именования компонент – это KebabCase. При регистрации компоненты вы можете использовать KebabCase и CamelCase (классическая и паскальная нотация).

```
<div id="app">
    <table>
        <tr is="blog-first-row"></tr>
        <tr is="blog-second-row"></tr>
        <tr is="blog-third-row"></tr>
 </table>
</div>
<script>
    new Vue({
        el: "#app",
        components:{
            'blogFirstRow':{ template:'<tr><td>Cell1</td</tr>' },
            'BlogSecondRow':{ template:'<tr><td>Cell2</td</tr>' },
            'blog-third-row':{ template:'<tr><td>Cell3</td</tr>' },
        }
    });
</script>
```

СОСТОЯНИЕ И ПОВЕДЕНИЕ КОМПОНЕНТОВ

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:

```
<div id="app">
    <counter></counter>
</div>
<script>
   Vue.component('counter', {
       data: function(){
           return { header: 'Counter Program‘ }
       },
       template:'<div><h2>{{header}}</h2></div>'
   });
   new Vue({
       el: "#app"
   });
</script>
```

СОСТОЯНИЕ И ПОВЕДЕНИЕ КОМПОНЕНТОВ

Компоненты могут определять поведение в виде методов, которые определяются через параметр methods:

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

РАЗДЕЛЯЕМОЕ СОСТОЯНИЕ КОМПОНЕНТОВ

Для каждого компонента можно определить его собственное состояние. Но также можно определять некоторое общее состояние:

```
<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>
```

PROPS

Параметр props хранит массив ключей или свойств, которым извне можно передать значения. Простейший способ использования входных параметров состоит в передаче литералов, то есть обычных строк:

```
<div id="app">
    <message-comp message="hello"></message-comp>
</div>
<script>
   Vue.component('message-comp', {
     props: ['message'],
     template: '<h2>{{ message }}</h2>'
   })
   new Vue({
       el: "#app"
   });
</script>
```

PROPS

Надо учитывать, что мы не можем определить свойство с одним и тем же именем и в props, и в data:

```
<div id="app">
    <message-comp message="hello"></message-comp>
</div>
<script>
   Vue.component('message-comp', {
     props: ['message'],
     data: function(){
       return { message: 'hi all'}
     },
     template: '<h2>{{ message }}</h2>'
   });
   new Vue({
      el: "#app"
   });
</script>
```

PROPS

Можно также устанавливать значения свойств (дочернего компонента) динамически в зависимости от введенных в поля ввода данных (родительского компонента или корневого приложения Vue):

```
<div id="app">
    <input type="text" v-model="welcome" /><br><br>
    <message-comp v-bind:message="welcome"></message-comp>
</div>
<script>
   Vue.component('message-comp', {
     props: ['message'],
     template: '<h2>{{message}}</h2>'
   })
   new Vue({
       el: "#app",
       data: {
           welcome:''
       }
   });
</script>
```

PROPS

Пример привязки к сложным объектам:

```
<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>
   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>
```

ВАЛИДАЦИЯ PROPS

Vue.js позволяет проверять корректность входных параметров, передаваемых родителем. Прежде всего мы можем указать тип для свойств. В качестве типов можно использовать следующие: String, Number, Boolean, Function, Object, Array, Symbol:

```
<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>
   Vue.component('user', {
        props: {name: String, age: Number},
        template: '<div><h2>User</h2><p>Name:{{name}}</p>
                        <p>Age: {{age}}</p></div>'
    })
    new Vue({
        el: "#app",
        data: {
             user: { name: '', age: 18 }
        }
    });
</script>
```

ВАЛИДАЦИЯ PROPS

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.

ВАЛИДАЦИЯ PROPS

Пример более точной валидации свойства:

```
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>`
})
```

ПЕРЕДАЧА МАССИВОВ И СЛОЖНЫХ ОБЪЕКТОВ

Пример описания компонента, которому могут передать объект из вне.

```
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>`
});
```

ПЕРЕДАЧА МАССИВОВ И СЛОЖНЫХ ОБЪЕКТОВ

Пример описания компонента, которому могут передать массив из вне.

```
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>`
});
```

РОДИТЕЛЬСКИЕ И ДОЧЕРНИЕ КОМПОНЕНТЫ

Одни компоненты (родительские компоненты) могут содержать другие (дочерние компоненты):

```
<div id="app">
    <userslist :users="users"></userslist>
</div>
<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>`
    });
    //...
</script>
```

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: от родителя к потомкам.

Кроме того, если мы попробуем 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.

ОДНОНАПРАВЛЕННЫЙ ПОТОК ДАННЫХ

Например:

```
<div id="app">
    <h2>Hello, {{name}}</h2>
    <useredit :user="name"></useredit>
    <button v-on:click="resetName">Set Name</button>
</div>
<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>
```

ОДНОНАПРАВЛЕННЫЙ ПОТОК ДАННЫХ

Значение свойства **name** в родительском объекте Vue никак не будет затронуто.

Почему так происходит?
Дело в том, 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 мы можем изменять массивы или сложные объекты, которые определены в родительских объектах. Часто это становится нежелательным побочным эффектом, которого нужно избегать. **Изменения следует производить только в родительском компоненте** - это фундаментальное правило однонаправленного связывания данных.

ОДНОНАПРАВЛЕННЫЙ ПОТОК ДАННЫХ

Усовершенствованная версия (родитель нам устанавливает свойство только в начале):

```
<div id="app">
    <h2>Hello, {{name}}</h2>
    <useredit :user="name"></useredit>
    <button v-on:click="resetName">Set Name</button>
</div>
<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>
```

СОЗДАНИЕ СОБЫТИЙ

Для передачи данных в обратном направлении от дочернего компонента к родителю 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.

СОЗДАНИЕ СОБЫТИЙ

Пример:

```
<div id="app">
    <h2>Hello, {{name}}</h2>
    <useredit :user="name" v-on:userchange="change"> </useredit>
</div>
<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>
```

ПЕРЕДАЧА ФУНКЦИЙ ОБРАТНОГО ВЫЗОВА В КОМПОНЕНТЫ

Пример передачи функций обратного вызова:

```
<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>
```

ПЕРЕДАЧА ФУНКЦИЙ ОБРАТНОГО ВЫЗОВА В КОМПОНЕНТЫ

Пример передачи функций обратного вызова (продолжение):

```
<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>`
});
</script>
```

ПЕРЕДАЧА ФУНКЦИЙ ОБРАТНОГО ВЫЗОВА В КОМПОНЕНТЫ

Пример передачи функций обратного вызова (продолжение):

```
<script>
    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>
```

ВЗАИМОДЕЙСТВИЕ МЕЖДУ СОСЕДНИМИ КОМПОНЕНТАМИ

Пример взаимодействия между соседними компонентами:

```
<div id="app">
    <h2>User</h2>
    <useredit :name="user" @userchange="change"></useredit>
    <userinfo :name="user" @userreset="reset"></userinfo>
</div>
<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);
        }
    }
});
```

ВЗАИМОДЕЙСТВИЕ МЕЖДУ СОСЕДНИМИ КОМПОНЕНТАМИ

Пример взаимодействия между соседними компонентами (продолжение):

```
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>
```

ВЗАИМОДЕЙСТВИЕ МЕЖДУ СОСЕДНИМИ КОМПОНЕНТАМИ

Еще один способ взаимодействия между компонентами одного уровня представляет применение шины событий:

```
<div id="app">
    <h2>User</h2>
    <useredit :user="user"></useredit>
    <userinfo :user="user"></userinfo>
</div>
<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);
        }
    }
});
```

ВЗАИМОДЕЙСТВИЕ МЕЖДУ СОСЕДНИМИ КОМПОНЕНТАМИ

Еще один способ взаимодействия между компонентами одного уровня представляет применение шины событий (продолжение):

```
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>
```

ДОПОЛНИТЕЛЬНЫЕ ССЫЛКИ

Сравнение Vue.js с другими фреймворками:
https://ru.vuejs.org/v2/guide/comparison.html

Руководство по Vue.js (онлайн учебник)
https://metanit.com/web/vuejs/