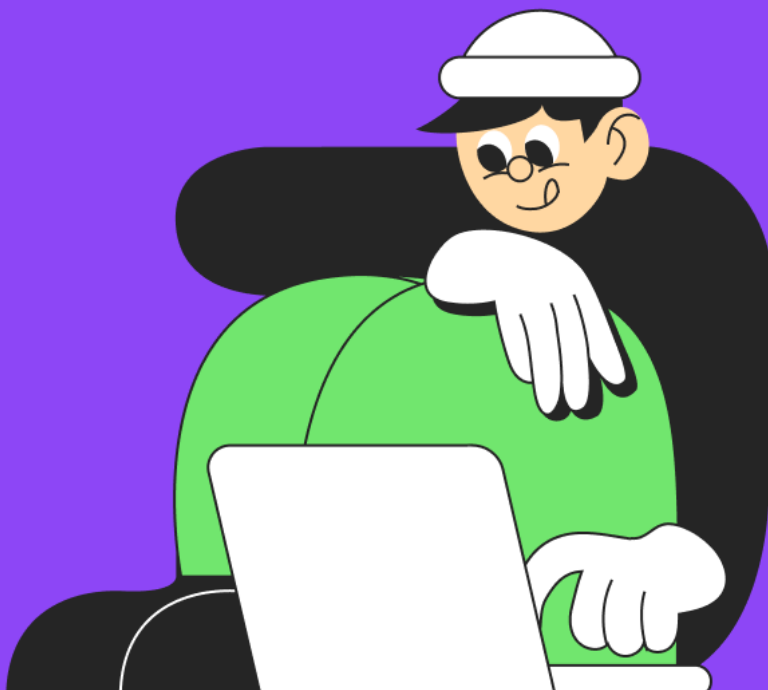
GeekBrains

Компоненты

Vue.js



Оглавление

Введение

Если приложение плохо спроектировано, работать с ним становится тяжело. Когда всё приложение описано в одном файле, в нём становится очень сложно разобраться, а кроме того, разные части приложения могут взаимодействовать непредсказуемо и неконтролируемо. Например, одноименные переменные могут оказаться в одной области видимости и перекрывать друг друга.

Для того, чтобы приложение было легче расширять и поддерживать, его принято делить на компоненты – отдельные независимые части. Нечто похожее реализовано в парадигме ООП, но многие отказываются именно от объектного подхода, так как наглядность в данной ситуации минимальная.Поэтому нам необходимо решить эту задачу плюс оставить наглядность и удобство использования или переиспользования частей кода, для этого как раз и существуют компоненты. В каждом серьезном фреймворке, в том числе и Vue.js, есть инструменты для реализации компонентного подхода.

Создание компонент

Сделаем основную разметку и подключим Vue:

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <title>Example component</title>
</head>

<body>
   <div id="app"></div>
   <script
src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
   <script src="script.js"></script>
</body>

</html>
```

```
const app = new Vue({
   el: '#app',
   data: {

   }
})
```

Теперь объявим новый компонент. Для этого у Vue есть метод **component()**. Этот метод принимает два аргумента: название компонента и объект с настройками:

```
Vue.component('some', {});
```

Название может быть любым, но если оно состоит из нескольких слов, их принято разделять дефисом:

```
Vue.component('some-component', {});
```

Главный параметр компонента – template, html-разметка. Напишем что-нибудь простое:

```
Vue.component('some-component', {
    template: '<h1>Hi Component!</h1>'
});

const app = new Vue({
    el: '#app',
    data: {

    }
})
```

Теперь мы можем вызвать компонент в основном файле с разметкой:

```
    <div id="app">
        <some-component></some-component>
    </div>
```

Эта запись будет эквивалентна следующей:

```
    <div id="app">
        <h1>Hi Component!</h1>
    </div>
```

То есть Vue подставляет компонент туда, где его вызвали. Компонент можно использовать несколько раз:

```
    <div id="app">
        <some-component></some-component>
        <some-component></some-component>
        <some-component></some-component>
        <some-component></some-component>
    </div>
```

Что мы получаем в данной ситуации

```
<div id="app">
    <h1>Hi Component!</h1>
    <h1>Hi Component!</h1>
    <h1>Hi Component!</h1>
    <h1>Hi Component!</h1>
</div>
```

Как мы можем заметить что код работает просто и понятно, а самое главное это функциональность, потому что сразу становится понятным где можно применить компонентный подход, ведь у нас на странице так много элементов которые дублируются, давайте рассмотрим более объемный пример

```
<div id="app">
    <article-website></article-website>
</div>

<script>
    Vue.component('article-website', {
        template: `
            <h3>New article</h3>
             <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nihil obcaecati repellat dolorum commodi aspernatur molestias esse cupiditate eius adipisci illo velit, dignissimos, optio ipsa nesciunt aliquid nemo nam minima id?</p>
        `
    });
</script>
```

Результат работы получается не самым очевидным

```
<div id="app"><h2>New article</h2></div>
 <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nihil obcaecati repellat dolorum commodi aspernatur
        molestias esse cupiditate eius adipisci illo velit, dignissimos, optio ipsa nesciunt aliquid nemo nam minima id?
```

```
</p>
```

Данная проблема называется ограничение шаблона

Ограничения шаблона

Важно помнить, что шаблон может содержать только один внешний элемент.
Будем идти от обратного, рассмотрим простую ситуацию, когда необходимо добавить 2 параграфа

```
Vue.component('some', {
    template: `
        <p></p>
        <p></p>
    `,
});
```

Чтобы это исправить, нужно обернуть все элементы в один:

```
Vue.component('some', {
    template: `
        <div>
            <p></p>
            <p></p>
        </div>
    `,
});
```

Это важная особенность, о которой обязательно нужно помнить.

Важно: мы сейчас говорим про Vue2, данную особенность в Vue3 версии поменяли и можно не использовать блок обёртки

Конечно возникают вопросы, а почему мы используем не самую актуальную версию, но тут ответ будет максимально простым, огромное количество проектов написано именно на Vue2 версии и в новых версиях не меняется полностью синтаксис языка или все конструкции, так что мы еще доберёмся до новшеств Vue3, а сейчас будут просто такие простые сноски на отличия в версиях

Вернемся к нашему шаблону статей, нам нужно обернуть элементы компонента обернуть блоком div и получить готовый шаблон

```
    <div id="app">
        <article-website></article-website>
    </div>

    <script>
        Vue.component('article-website', {
            template: `
                <article>
                    <h3>New article</h3>
                    <p>Lorem ipsum, dolor sit amet consectetur
adipisicing elit. Nihil obcaecati repellat dolorum commodi
aspernatur molestias esse cupiditate eius adipisci illo velit,
dignissimos, optio ipsa nesciunt aliquid nemo nam minima id?</p>
                </article>
            `
        });
        const app = new Vue({ el: '#app' })
```

Теперь давайте представим что на нашем сайте несколько разделов статей и получается что внутри такого раздела сайта может быть несколько статей, получается нам необходимо создавать компонент, который будет содержать в себе другие компоненты. Конечно мы можем встретить такую ситуацию и компоненты легко можно разместить внутри другого

```
<div id="app">
    <section-website></section-website>
</div>

<script>
    Vue.component('section-website', {
        template: `
            <section>
                <h2>New section</h2>
```

```
                <article-website></article-website>
                <article-website></article-website>
            </section>
        `,
    });
    Vue.component('article-website', {
        template: `
            <article>
                <h2>New article</h2>
                        <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Nihil obcaecati repellat dolorum commodi aspernatur molestias esse cupiditate eius adipisci illo velit, dignissimos, optio ipsa nesciunt aliquid nemo nam minima id?</p>
            </article>
        `
    });
    const app = new Vue({ el: '#app' })
```

Теперь на странице может быть несколько разделов сайта и если нам нужно будет переиспользовать компонент, то мы сразу получаем готовое решение.

Конечно использование компонентов это отлично, но какую проблему мы успеваем заметить, конечно же это то что при дублировании компонента статей, его содержимое не меняется, да и заголовок наших секций будет неизменным, получается что нам не хватает проброса содержимого внутри компонента

Проброс содержимого

Содержимое, которое находится между открывающим и закрывающим тегами компонента, можно пробрасывать внутрь и подставлять в шаблон. Для подстановки содержимого используется тег <slot>:

```
<div id="app">
    <some-component>Hello</some-component>
    <some-component>New</some-component>
    <some-component>Heading</some-component>
</div>
```

```
<script>
   Vue.component('some-component', {
       template: '<h2><slot></slot></h2>'
   });
</script>
```

Результат:

```
   <div id="app">
       <h2>Hello</h2>
       <h2>New</h2>
       <h2>Heading</h2>
   </div>
```

Конечно данный подход не кажется самым рациональным, но мы начинаем от самого простого и переходим к более сложному

Работа с данными

В компонентах данные тоже хранятся в поле **data**, но здесь это не объект, а функция, возвращающая объект:

```
       Vue.component('some-component', {
           template: '<h2></h2>',
           data() {
               return {
                   name: 'Frodo',
               };
           },
       });
```

Имя можно подставить в шаблон компонента, используя mustache-синтаксис (усатый синтаксис):

```
       Vue.component('some-component', {
           template: '<h2>{{ name }}</h2>',
           data() {
               return {
                   name: 'Frodo',
```

```
            };
        },
    });
```

Передача свойств

В компоненты можно передавать данные извне с помощью свойств. Это значения, которые устанавливаются во время вызова компонента и которые можно использовать для преобразования и подстановки в шаблон. Вернёмся к примеру выше:

```
    Vue.component('some-component', {
        template: '<h2>{{ name }}</h2>',
        data() {
            return {
                name: 'Frodo',
            };
        },
    });
```

Удалим из него поле data:

```
    Vue.component('some-component', {
        template: '<h1>{{ name }}</h1>'
    });
```

Сейчас переменной name не существует, поэтому компонент выдаст ошибку. Объявим name как свойство при вызове компонента:

```
<div id="app">
   <some-component name="Frodo Baggins"></some-component>
</div>
```

Но компонент будет выдавать ошибку и теперь. Свойство нужно объявить внутри компонента в поле **props**. В это поле передается массив с именами всех свойств:

```
Vue.component('some-component', {
   props: ['name'],
   template: '<h2>{{ name }}</h2>',
```

```
});
```

Теперь значение свойства name будет подставляться в шаблон.
Если нужно пробросить не конкретное значение, а содержимое переменной, то это можно сделать через конструкцию **v-bind**. Например, в основном хранилище данных нашего приложения есть переменная **name**:

```
const app = new Vue({
   el: '#app',
   data: {
       name: 'Frodo'
   }
})
```

И конечно блок html

```
<div id="app">
   <some-component :name="name"></some-component>
</div>
```

Как итог, мы можем передавать любой контент с помощью тега <slot> что оказалось не очень удобным, более правильный подход это в сам компонент передать значение в props и переиспользовать его с помощью “усатого синтаксиса”, ну и конечно мы можем использовать значения в переменных через v-bind

Итоги урока

Самый важный момент который нам нужно понять это то что компоненты очень объемная тема и мы будем продолжать ее изучения на последующих уроках. Мы научились создавать простые компоненты, рассмотрели вызов данных компонентов, научились создавать компоненты внутри компонентов и конечно же научились передавать значения внутри компонента. Чего нам еще не хватает, конечно работы с шаблонизацией, мы можем не только использовать одно значения внутри шаблона, мы можем использовать массив данных и точно так же передать их в наш компонент, поэтому основная задача для вас сейчас научиться создавать простые компоненты, понять где они используются, а как создать более объемный компонент будет чуть позже

Используемая литература

1. https://ru.vuejs.org/