Урок 6

Компоненты Vue.js

Реализация компонентного подхода во фреймворке Vue.js.

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, одноимённые 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 мы делали, когда применяли ООП на втором уроке. В каждом серьёзном фреймворке, в том числе и Vue.js, есть инструменты для реализации компонентного подхода.

Создание компонентов

Сделаем основную разметку и подключим Vue:

```
<!DOCTYPE html>
<html>
<head>
  <title>eShop</title>
</head>
<body>
  <div id="app">
  </div>

  <script src="https://cdn.jsdelivr.net/npm/vue@2.5.17/dist/vue.js"></script>
  <script src="script.js"></script>
</body>
</html>
```

```
const app = new Vue({
  el: '#app'
});
```

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', {
  template: '<h1>Hi!</h1>'
});
```

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!</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>
...
```

Этот код будет преобразован в следующий:

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

Можно вызывать один компонент внутри другого:

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

Vue.component('another-component', {
  template: '<i>vue</i>'
});
```

В итоге **<some-component>** превратится в такой html:

```
<h1>Hi! <i>vue</i></h1>
```

Проброс содержимого

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

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

Теперь в основном файле вызовем этот компонент с разным содержимым:

```
...
<div id="app">
  <some-component>Here we go</some-component>
  <some-component>Here we go</some-component>
  <some-component>Here we go again</some-component>
</div>
...
```

Vue преобразует эту запись в

```
...
<div id="app">
  <h1>Here we go</h1>
  <h1>Here we go</h1>
  <h1>Here we go again</h1>
</div>
...
```

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: '<h1></h1>',
  data() {
    return {
      name: 'Frodo'
    }
  }
});
```

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

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

Ограничения шаблона

Важно помнить, что шаблон может содержать только один внешний элемент. Рассмотрим простейшую ситуацию:

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

Такой компонент будет работать нормально. Внутрь абзаца можно вложить другие элементы, и код тоже будет работать:

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

Но если мы добавим новый элемент на том же уровне, что и абзац, то Vue не сможет подставить шаблон:

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

Чтобы это исправить, нужно обернуть все элементы в один:

```
// Работает
Vue.component('some', {
```

```
  template: `
    <div>
      <p></p>
      <p></p>
    </div>
  `
});
```

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: '<h1>{{ name }}</h1>',
  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 передаётся массив с именами всех свойств:

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

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'
  }
});
```

Передадим значение этой переменной в компонент:

```
...
<div id="app">
  <some-component v-bind:name="Frodo Baggins"></some-component>
</div>
...
```

У **v-bind** есть сокращённая запись. Можно опустить само ключевое слово **v-bind** и оставить только двоеточие:

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

Практика

Вынесем список товаров и карточку товара интернет-магазина в отдельные компоненты. Создадим заготовки компонентов:

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

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

Начнём со списка. В первую очередь разберёмся, с какими данными ему предстоит работать. На вход он получит список **filteredGoods**:

```
...
<main>
  <goods-list :goods="filteredGoods"></goods-list>
</main>
...
```

Теперь внутри div'а компонент будет создавать карточки товаров с помощью **v-for** и передавать информацию о товаре в компонент **goods-item**. Добавим это в шаблон:

```
Vue.component('goods-list', {
  props: ['goods'],
  template: `
    <div class="goods-list">
      <goods-item v-for="good in goods" :good="good"></goods-item>
    </div>
  `
});
```

Перейдём к карточке товара. Здесь всё проще: на вход поступает объект с информацией о товаре, и мы просто подставляем в шаблон нужные поля:

```
Vue.component('goods-item', {
  props: ['good'],
  template: `
    <div class="goods-item">
      <h3>{{ good.product_name }}</h3>
      <p>{{ good.price }}</p>
    </div>
  `
});
```

Практическое задание

1. Вынести поиск в отдельный компонент.
2. Вынести корзину в отдельный компонент.
3. * Создать компонент с сообщением об ошибке. Компонент должен отображаться, когда не удаётся выполнить запрос к серверу.

Дополнительные материалы

1. Видеокурс по основам Vue.js.

Используемая литература

1. Официальная документация Vue.js.