

Синтаксис шаблона

- Введение
- Интерполяция
- Использование выражений JavaScript

Введение

Vue использует синтаксис шаблонов, основанный на HTML. **Он позволяет декларативно связывать отрисованный DOM с данными экземпляра компонента.** Все шаблоны Vue являются валидным HTML-кодом, который могут распарсить все HTML-парсеры и браузеры, соответствующие спецификациям.

Для работы Vue компилирует шаблоны в render-функции виртуального DOM. Вместе с системой реактивности, Vue может определять минимальное число компонентов для перерисовки и выполняет минимальное количество манипуляций с DOM при изменениях состояния приложения.

Интерполяция

Текст

Наиболее простой способ связывания данных — это **текстовая интерполяция** с использованием синтаксиса двойных фигурных скобок:

```
<!-- простое связывание данных -->
```

```
<span>Сообщение: {{ msg }}</span>
```

Выражение в фигурных скобках будет **заменено** значением свойства **msg** соответствующего **объекта данных**. Кроме того, оно будет **обновлено** при **любом изменении этого свойства**.

Возможно также выполнение интерполяции **однократно**, которая не обновится при изменении данных — с помощью **директивы v-once**. Обратите внимание, это повлияет сразу на все связанные **переменные** внутри данного HTML-элемента:

```
<!-- это сообщение никогда не изменится -->
```

```
<span v-once> {{ msg }} </span>
```

Здесь мы встречаемся с новым понятием, атрибут **v-once**, называется **директивой**. Директивы имеют префикс **v-**, указывающий на их особую природу (подробнее о директивах в следующем уроке).

example_1. Однократная интерполяция текстовых данных

```
<div id="app">

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

  <!--
    свойство appeal будет интерполировано однажды,
    только при инициализации экземпляра Vue
  -->

  <h2 v-once>{{ appeal }}</h2>

</div>

<script>

  const root = {
    data() {
      return {
        message: "Hello, I'm VueJS",
        appeal: "Я - прогрессивный JavaScript-фреймворк"
      }
    }
  };

  let vm = Vue.createApp(root).mount("#app");
```

```
// изменим свойство экземпляра Vue

vm.message = "Привет, я Евгений!";

// это изменение не отразится на экземпляре

vm.appeal = "Я хочу изучить фреймворк VueJS ;)";

</script>
```

Однократно будут интерполироваться все переменные внутри тега с директивой **v-once**.

example_2. Вложение переменных в директиву v-once

```
<div id="app">

  <h2>Цитаты великих</h2>

  <!-- все, что вложено в v-once интерполируется один раз -->

  <div v-once>

    <h2>{{ aphorism }}</h2>

    <div>{{ author }}</div>

  </div>

</div>

<script>

  const root = {

    data() {

      return {

        aphorism: "Если человек не курит и не пьёт, поневоле задумаешься, уж не сволочь ли он?",

        author: "А.П. Чехов"

      }

    }

  };

  let vm = Vue.createApp(root).mount("#app");
```

```
// изменим свойства

vm.apothism = "В сумасшедшем доме каждый мог говорить все,
что взбредет ему в голову, словно в парламенте";

vm.author = "Ярослав Гашек";

</script>
```

Сырой HTML

Значение выражения в двойных фигурных скобках подставляется как простой текст, а не как HTML. Для HTML необходимо использовать **директиву v-html**.

Пример:

```
<!-- фигурные скобки игнорируют html -->

<p>Двойные фигурные скобки: {{ rawHtml }}</p>

<p>Директива v-html: <div v-html="rawHtml"></div></p>
```

Содержимое тега **div** будет заменено значением свойства **rawHtml**, интерпретированного как обычный HTML — все привязки данных игнорируются. Вы не можете использовать **v-html** для вложения шаблонов друг в друга, потому что движок шаблонов Vue не основывается на строках.

example_3. Интерполяция сырого HTML

```
<div id="app">

  <!-- фигурные скобки игнорируют html -->

  <p>Двойные фигурные скобки <b>игнорируют html:</b><br/> {{
    rawHTML }}</p>

  <!-- вывод сырого html -->

  <!-- raw(En) - сырой -->

  <p>Директива <b>v-html:</b><br/><span v-
    html="rawHTML"></span></p>

</div>
```

```
<script>

  var vm = Vue.createApp({

    data() {

      return {

        rawHTML: "<h2>Hello, I'm VueJS</h2>"

      }

    }

  }).mount("#app");

</script>
```

Динамическая отрисовка произвольного HTML-кода на сайте крайне опасна, так как может легко привести к **XSS-уязвимостям**. Используйте интерполяцию HTML только для **доверенного кода**, и никогда не подставляйте туда содержимое, **создаваемое пользователями**.

HTML-атрибуты

Синтаксис двойных фигурных скобок не работает с **HTML-атрибутами**.

Используйте вместо него **директиву v-bind**:

```
<div v-bind:id="dynamicId"></div>

<div v-bind:class="dynamicClass"></div>


```

example_4. Интерполяция HTML-атрибутов

```
<div id="app">

</div>

<script>
```

```
</script>
```

При использовании с булевыми атрибутами (когда их наличие уже означает **true**) **v-bind** работает немного иначе. В примере:

```
<button v-bind:disabled="isButtonDisabled">Кнопка</button>
```

если значением `isButtonDisabled` будет **null**, **undefined** или **false**, то атрибут **disabled** не добавится в элемент `<button>`.

Рассмотрим пример.

example_5. Особенности интерполяции булевых атрибутов

```
<div id="app">

</div>

<script>

</script>
```

Приведу пример работы с атрибутами тега изображения ``.

example_6. Интерполяция атрибутов изображения

```
<div id="app">

<!-- тег для вывода изображения -->



</div>

<script>

  const vm = Vue.createApp({

    data() {

      return {

        // возможные значения:

        // image-1.jpg | image-2.jpg | image-3.jpg
```

```
        src: "image-2.jpg",  
        width: 350,  
        title: "Всем позитива и добра"  
      }  
    }  
  }).mount("#app");  
</script>
```

Использование выражений JavaScript

Пока мы связывали данные со свойствами в шаблонах только по простым ключам. Но на самом деле при связывании данных Vue поддерживает все возможности **выражений JavaScript**.

Выражения в фигурных скобках

В фигурных скобках можно выполнять различный JavaScript код.

Рассмотрим возможность на практическом примере.

Пусть у нас есть два элемента с числами - **var1** и **var2**:

```
var vm = Vue.createApp({  
  data() {  
    return {  
      var1: 5,  
      var2: 2  
    }  
  }  
}).mount("#app");
```

Давайте, например, сложим наши переменные **var1** и **var2**:

```
<div id="app">
  {{ var1 + var2 }}
</div>
```

Результатом этого кода будет следующий HTML:

```
<div id="app">
  7
</div>
```

Выражения будут вычислены как JavaScript-код в **области видимости** (корневом элементе) **соответствующего экземпляра Vue**.

example_7. Выражения в шаблонах

```
<div id="app">

  <p>Сумма чисел: {{ num1 }} + {{ num2 }} + {{ num3 }} = {{
    num1 + num2 + num3 }}</p>

</div>

<script>

  root = {

    data() {

      return {

        num1: 1,

        num2: 2,

        num3: 3

      }

    }

  };

  const app = Vue.createApp(root);

  const vm = app.mount("#app");

</script>
```


До сих пор связывали данные со свойствами в шаблонах только по простым ключам. Но на самом деле Vue поддерживает всю мощь выражений JavaScript **внутри привязок данных**.

Единственное ограничение в том, что допускается лишь одно выражение, поэтому код ниже **не сработает**:

```
<!-- это не вычисляемое выражение, а определение переменной -->

{{ var a = 1 }}

<!--

    операторы условий не сработают,

    используйте условные операторы в тернарной форме

-->

{{ if (ok) { return message } }}
```

example_8. Объекты и условия в выражениях шаблонов

```
<div id="app">

  <p>Случайное число: {{ Math.floor(Math.random() * (max - min
+ 1)) + min }}</p>

  <p>Текущая дата: {{ new Date() }}</p>

  <p><h3>{{ isAdmin ? 'Вы зашли как Администратор' : 'Кто
здесь?' }}</h3></p>

</div>

<script>

  root = {

    data() {

      return {

        max: 5,

        min: 1,

        isAdmin: 1

      }

    }

  }

</script>
```

```
    }  
  }  
};  
  
const app = Vue.createApp(root);  
  
const vm = app.mount("#app");  
  
</script>
```

Выражения в шаблонах имеют доступ только к небольшому белому списку глобальных свойств, таких как **Math** и **Date**. **Не следует пытаться получить доступ к пользовательским глобальным свойствам в выражениях шаблонов.**

Массивы и объекты в выражениях

Вывод содержимого массивов и объектов во Vue осуществляется так же, как и в чистом JavaScript. Посмотрим на примерах.

example_9. Массивы и объекты в выражениях

```
<div id="log">  
  <p>Длина массива: {{ arr.length }}</p>  
  <p>Сумма элементов массива: {{arr[0]}} + {{arr[1]}} +  
    {{arr[2]}} равна {{ arr[0] + arr[1] + arr[2] }}</p>  
  <p>Конкатенация ключей объекта: <h2>{{ obj.surname + ' ' +  
    obj["name"] + ' ' + obj.patronymic}}</h2></p>  
</div>  
  
<script>  
  let app = Vue.createApp({  
    data() {  
      return {  
        // массив
```

```
arr: [1,2,3],  
  
  // объект  
  
  obj: {surname:"Распутин", name:"Григорий",  
        patronymic:"Ефимович"}  
  
    }  
  
  }  
  
  }).mount("#log");  
  
</script>
```

Выражения в HTML-атрибутах

Выражения можно использовать не только в фигурных скобках, но и в атрибутах. В следующем примере демонстрируется динамическое определение атрибута **src** для тега **img**.

example_10. Выражения в HTML-атрибутах

```
<div id="app">  
  
  <!-- формируем атрибут выражением -->  
  
    
  
</div>  
  
<script>  
  
  let vm = Vue.createApp({  
  
    data() {  
  
      return {  
  
        // значение по умолчанию - undefined  
  
        numIMG: undefined  
  
      }  
  
    }  
  
  }).mount("#app");
```

```
// случайным образом получаем значение ключа numIMG  
vm.numIMG = Math.floor(Math.random() * (5 - 1 + 1)) + 1;  
  
</script>
```

Задача 1

В файле **orders.js** хранится заказ в виде простого одномерного **массива**:

```
var orders = [  
  '1',  
  'Утепленная стеганая куртка',  
  'MARCO DI RAD1',  
  'Синий',  
  'Китай',  
  'Есть',  
  '52',  
  '25.590',  
  'steganaya-marco.jpg'  
]
```

Выведите данные заказа в **шаблон** приложения **VueJS**.

Задача 2

Пусть в опции **data** хранится массив с именами файлов-изображений.

```
data() {  
  return {  
    dynamicIMG: ["image-1.jpg", "image-2.jpg", "image-3.jpg"]  
  }  
}
```

Выведите в шаблон **случайное изображение**. Индекс для доступа к элементу массива может быть рассчитан с использованием JS-выражения:

```
Math.floor(Math.random() * (3 - 1 + 1))
```

Задача 3

Пусть в файлах **user.js** и **orders.js** хранятся данные о пользователе и сделанном им заказе. Напишите приложение, **выведите данные** о заказе в браузер. Рассчитайте **конечную** цену заказа и цену с учетом **существующей скидки**.

P.S. Вариантов выполнения вывода может быть **много**. Если ваше решение отличается от моего, смотрим, анализируем, идем дальше.

Здесь и далее я буду **выкладывать решения** предлагаемых задач. Решения вполне могут отличаться от ваших. Более того, решение может быть не оптимальным. Моя задача не предъявить миру свою гениальность, моя задача - **научить студента**. Чаще всего предлагаемый код будет в **контексте рассматриваемого урока / темы**.

P.S.

Для отработки и закрепления учебного курса **донам** группы предоставляется следующий раздаточный материал.

К каждому уроку курса:

- Файлы **демонстрационного кода** (example);
- **Задачи** с решениями в контексте рассматриваемых вопросов урока (task).

К каждой теме курса:

- **Практические работы**.