

Директивы

- Введение
- Динамические аргументы директив
- Ограничения
- Сокращенная запись
- Модификаторы событий
- Работа с объектом Event

Введение

Директивы — это специальные атрибуты с префиксом **v-**. В значении директивы ожидается одно выражение JavaScript (за исключением **v-for** и **v-on**, о которых поговорим далее). Работа директивы заключается в **реактивном** применении изменений к DOM, при изменении **значения выражения в скрипте**.

Пример:

```
<!-- директива v-if -->  
<p v-if="show">Да здравствует Король!</p>
```

В этом случае директива **v-if** будет удалять или вставлять элемент **<p>** в зависимости от **логического значения** выражения **show**.

example_1. Директива v-if

```
<div id="app">  
  <div v-if="isShow">  
    <h2>Добро пожаловать в мир прогрессивного фреймворка  
    Vue!</h2>  
  </div>  
</div>  
<script>
```

```
let app = Vue.createApp({  
  data() {  
    return {  
      // значение свойства isShow определяет видимость  
      // блока <div> шаблона  
      isShow: true  
    }  
  }  
}).mount("#app");  
</script>
```

Аргументы директив

Некоторые директивы могут принимать **аргумент**, отделяемый от названия директивы двоеточием. Например, директива **v-bind** используется для реактивного обновления атрибутов HTML:

```
<!-- директива v-bind с аргументом href -->
```

```
<a v-bind:href="url"> ... </a>
```

В этом примере **href** — аргумент, указывающий директиве **v-bind** связать атрибут **href** элемента со значением выражения **url**.

```
<!-- директива v-bind с аргументом id -->
```

```
<h2 v-bind:id="id">...</h2>
```

```
<!-- директива v-bind с аргументом src -->
```

```

```

```
<!-- директива v-bind с аргументом width -->
```

```
<div v-bind:width="width"></div>
```

```
<!-- несколько директив в одном элементе -->
```

```

```

Другим примером может быть директива **v-on**, которая **отслеживает события** DOM:

```
<!-- директивы v-on с аргументом click -->
```

```
<button v-on:click="onShow()">...</button>
```

```
<a href="#" v-on:click="onShow()">...</a>
```

В этом примере аргументом определяется тип прослушиваемого события — **click**. Подробнее обработку событий разберём позднее.

В следующем примере внимательно отследите **зависимость аргумента директивы** и **связанного с ним свойства данных** (JS-выражения).

example_2. Директивы и аргументы

```
<div id="app">

  <!-- все элементы шаблона имеют директивы с атрибутами -->

  <h2 v-bind:id="id">Добро пожаловать в мир прогрессивного
  фреймворка Vue!</h2>

  <p><a v-bind:href="url" target="blank">.. и мы готовы провести
  вас коротким путем ;)</a></p>

  <p><button v-on:click="onShow()">Нажми меня</button></p>

  <div v-if="show">

    

  </div>

</div>

<script>

  const app = Vue.createApp({

    data() {

      return {

        // значения атрибутов в свойствах data

        id: "msg",
```

```
url: "https://vk.com/pechora_pro",  
show: false,  
img: "the-shining.png",  
width: 200  
}  
,  
methods: {  
  onShow() {  
    // инвертируем значение show  
    this.show = !this.show;  
  }  
}  
}).mount("#app");  
</script>
```

Еще раз. Аргумент директивы это атрибут тега.

Динамические аргументы директив

В аргументе директивы можно использовать выражение JavaScript, для этого потребуется обернуть его в квадратные скобки.

К сведению. Для выражений аргументов есть ограничения, подробнее в разделе **Ограничения**.

```
<!-- выражение для директивы v-bind -->  
<a v-bind:[attributeName]="url"> ... </a>  
<!--
```

`v-bind` - директива

`attributeName` - аргумент директивы (атрибут)

`url` - значение аргумента (атрибута)

-->

Если в экземпляре компонента есть свойство данных **attributename** со значением **href**, то это будет эквивалентно привязке **v-bind:href**.

```
<a v-bind:href="url"> ... </a>
```

Свойство данных **attributename** будет рассматриваться как **выражение JavaScript**, а его вычисленное значение использоваться в качестве **финального значения для аргумента**.

Так, в примере **example_3** при создании экземпляра компонента:

- свойство данных **attributename** имело значение: **attributename = "href"**, но после выполнения скрипта было вычислено в **attributename = "onclick"**.
- значение атрибута было **exp = "#"**, после выполнения стало **exp = "alert(...)"**.

Следующий рисунок демонстрирует **состояние шаблона** на момент инициализации экземпляра. Можно посмотреть, если закомментировать строки:

```
// включаем магию  
// app.attributename = "onclick";  
// app.exp = "alert('А ссылки больше нет..., расстроился?')";
```

Обратите внимание на написание свойства **attributename** (регистр букв), подробнее в разделе Ограничения.

```
...▼ <body> == $0
  <h1 align="center">Основы VueJS</h1>
  ▼ <div id="app" data-v-app>
    <!-- динамически определим атрибут и его значение для ссылки -->
    <a href="#" target="_blank">Жду клика, мой генерал!</a>
  </div>
  ▶ <script>...</script>
  ▶ <footer align="center">...</footer>
</body>
</html>
```

Состояние шаблона (свойство **innerHTML**) после выполнения.

```
...▼ <body> == $0
  <h1 align="center">Основы VueJS</h1>
  ▼ <div id="app" data-v-app>
    <!-- динамически определим атрибут и его значение для ссылки -->
    <a target="_blank" onclick="alert('А ссылки больше нет..., расстроился?')">Жду клика, мой
    генерал!</a>
  </div>
  ▶ <script>...</script>
  ▶ <footer align="center">...</footer>
</body>
</html>
```

Экспериментируем с аргументами в примере **example_3**.

example_3. Меняем динамические аргументы директив вручную

```
<div id="app">

  <!--
  динамически определим атрибут и его значение для ссылки
  -->

  <a v-bind:[attributename]="exp" target="_blank">Жду клика, мой
  генерал!</a>

</div>

<script>

  let app = Vue.createApp({

    data() {

      return {
```

```
      attributename: "href",  
      exp: "#"   
    }  
  }  
  
  }).mount("#app");  
  
  // включаем магию  
  
  app.attributename = "onclick";  
  
  app.exp = "alert('А ссылки больше нет..., расстроился?')";  
  
  // было во время создания экземпляра  
  
  // <a href="#" target="blank">Жду клика, мой генерал!</a>  
  
  // стало после выполнения скрипта  
  
  // <a onclick="alert(...)" target="blank">Жду клика, мой  
  генерал!</a>  
  
</script>
```

Если в предыдущем примере вычисления выполнялись автоматически, то теперь будут **инициироваться пользователем** при нажатии кнопки:

```
<button v-on:click="onShowLink">Показать ссылку</button>
```

example_4. Меняем динамические аргументы через веб-интерфейс

```
<div id="app">  
  <div>  
  
    <h2>Добро пожаловать в мир прогрессивного фреймворка  
    Vue!</h2>  
  
    <button v-on:click="onShowLink">Показать ссылку</button>  
  
    <p><a v-bind:[attributeName]="val" target="blank">... и мы  
    готовы провести вас коротким путем ;)</a></p>  
  
  </div>  
</div>
```

```
<script>

let app = Vue.createApp({
  data() {
    return {
      // начальные значения:
      // аргумент директивы
      attributename: "title",
      // значение аргумента
      val: "Кликните кнопку для показа ссылки"
    }
  },
  methods: {
    onShowLink() {
      this.attributename = "href",
      this.val = "https://vk.com/pechora_pro"
    }
  }
}).mount("#app");

// обратите внимание на пару аргумент-значение
// было при инициализации экземпляра:
// title='Кликните кнопку для показа ссылки'
// стало после клика:
// href='https://vk.com/pechora_pro'

</script>
```

Аналогичным образом можно использовать динамические аргументы для директивы **v-on**, в которой **динамически будет определяться имя события**:

```
<!-- выражение для директивы v-on -->
```



```
<a v-on:[eventName]="fnTest"> ... </a>
```

```
<!--
```

где:

v-on - директива

eventName - аргумент директивы

```
-->
```

Если экземпляр компонента содержит свойство `eventname` со значением `focus` — итоговый обработчик будет эквивалентен `v-on:focus`.

```
<a v-on:focus="fnTest"> ... </a>
```

example_5. Создание события для обработчика

```
<div id="app">

  <h2>Добро пожаловать в мир прогрессивного фреймворка Vue!</h2>

  <!-- имя события для обработчика fnTest не задано -->

  <a v-on:[eventName]="fnTest" href="#">Click me</a><p>

  <span v-on:[eventName]="fnTest">And click me</span><p>

  <button v-on:[eventName]="fnTest">Click me too</button>

</div>

<script>

  let app = Vue.createApp({

    data() {

      return {

        // обратите внимание на написание свойства

        // eventname, см. раздел Ограничения

        eventname: "",

      }

    },

  })
```

```
methods: {  
  
  fnTest() {  
  
    alert("Hello, I'm Vue!");  
  
  }  
  
}  
  
}).mount("#app");  
  
// назначим событием - click  
app.eventname = "click";  
  
// или - mousemove  
// app.eventname = "mousemove";  
  
</script>
```

Ограничения

Ограничения динамического аргумента

Ожидается, что динамический аргумент после вычисления будет **строкой**, за исключением **null**. Специальное значение **null** можно использовать для явного **удаления привязки**. Использование любых других не строковых значений будет выбрасывать **предупреждения**.

Ограничения динамического выражения

Для выражения динамического аргумента есть синтаксические ограничения, потому что некоторые символы, такие как **пробелы** и **кавычки**, являются **недопустимыми для имён атрибутов HTML**.

Например:

```
<!--
```

так формировать аргумент не получится

компилятор выдаст предупреждение

```
-->
```

```
<a v-bind:['foo' + bar]="value"> ... </a>
```

Рекомендуется выносить любые сложные выражения в **вычисляемые свойства**, одну из важных фундаментальных частей Vue (познакомимся чуть позже).

При использовании шаблонов в DOM (шаблонов, написанных непосредственно в HTML-файле), следует также избегать имён ключей с **прописными символами**, потому что **браузеры будут принудительно приводить имена атрибутов к нижнему регистру**:

```
<a v-bind:[attributeName]="val" target="blank"> ... </a>
```

```
<button v-on:[eventName]="fnTest"> ... </button>
```

```
<!--
```

в шаблоне DOM это преобразуется соответственно в

```
v-bind:[attributename]
```

```
v-on:[eventname]
```

```
-->
```

```
<!--
```

если в экземпляре Vue не будет свойства `attributename` или `eventname`, то код работать не будет

```
-->
```

Сокращенная запись

Префикс **v-** нужен для визуального обозначения **Vue-специфичных атрибутов** в шаблонах. Это особенно удобно, когда Vue используется для добавления динамического поведения в существующей разметке, но для часто используемых директив может быть многословным.

С другой стороны, потребность в **v-** ещё меньше при создании одностраничных приложений, где Vue управляет каждым шаблоном. Поэтому для двух наиболее часто используемых директив **v-bind** и **v-on** есть **сокращённая запись**



<!-- полный синтаксис -->

```
<a v-bind:href="url"> ... </a>
```

<!-- сокращённая запись -->

```
<a :href="url"> ... </a>
```

<!-- сокращённая запись с динамическим именем аргумента -->

```
<a :[key]="url"> ... </a>
```

Сокращение v-on

<!-- полный синтаксис -->

```
<a v-on:click="fnTestEvent"> ... </a>
```

<!-- сокращённая запись -->

```
<a @click="fnTestEvent"> ... </a>
```

<!-- сокращённая запись с динамическим именем события -->

```
<a @[event]="fnTestEvent"> ... </a>
```

example_6. Сокращенная запись директивы

```
<div id="app">

  <h2>Добро пожаловать в мир прогрессивного фреймворка Vue!</h2>

  <input type="radio" @click="click1()" name="group">
  <label>Click</label>

  <input type="radio" @click="click2()" name="group">
  <label>Mouse Move</label>

  <input type="radio" @click="click3()" name="group">
  <label>Db1 Click</label>

  <p><a @[eventName]="fnTest" href="#">Click me</a></p>

</div>
```

```
<script>

  let app = Vue.createApp({

    data() {

      return {

        eventname: ""

      }

    },

    methods: {

      click1() {this.eventname = "click"},

      click2() {this.eventname = "mousemove"},

      click3() {this.eventname = "dblclick"},

      fnTest() {alert("Привет, вот и я! ;")}

    }

  }).mount("#app");

</script>
```

Подобная запись на вид несколько отличается от обычного HTML-кода, но символы ":" и "@" являются **допустимыми символами в именах атрибутов** и браузеры, которые поддерживает Vue, могут их корректно обработать. Кроме того, в итоговой разметке их уже не будет.

Сокращённый синтаксис полностью опционален.

Модификаторы событий

При работе с событиями очень часто возникает необходимость вызвать **event.preventDefault()** или **event.stopPropagation()** в обработчике события. Кратко напомним, о чем идет речь.

preventDefault

Многие **события** автоматически влекут за собой определенные **действия** браузера. Например:

- клик по ссылке инициирует переход на новый **url**;
- нажатие на кнопку "отправить" в форме – отсылку её на сервер;
- зажатие кнопки мыши над текстом и её движение в таком состоянии – инициирует его выделение.

Если мы обрабатываем событие в JavaScript, то зачастую такое действие браузера нам не нужно. Есть два способа **отменить действие браузера по умолчанию**:

- Основной способ – это воспользоваться объектом **event**. Для отмены действия браузера существует стандартный метод **event.preventDefault()**.
- Если же обработчик назначен через **on<событие>**, то также можно вернуть **false** из обработчика.

В следующем примере клик по ссылке к переходу не приведет :

```
<a href="#" onclick="return false">Нажми здесь</a>
```

```
<!-- или -->
```

```
<a href="#" onclick="event.preventDefault()">Можно здесь</a>
```

stopPropagation

Принцип **всплытия** очень простой. Когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе, затем выше и так далее, **вверх по цепочке предков**.

Например, есть три вложенных элемента **section > div > p** с обработчиком события **onclick** на каждом из них:

```
<section onclick="alert('section')">SECTION
```

```
  <div onclick="alert('div')">DIV
```

```
    <p onclick="alert('p')">P</p>
```

```
  </div>
```

Всплытие гарантирует, что клик по внутреннему элементу **p** инициирует вызов обработчиков **onclick**:

- сначала на самом **<p>**
- потом на внешнем **<div>**
- затем на внешнем **<section>**
- и так далее вверх по цепочке до самого **document**.

При этом любой **промежуточный обработчик** может решить, что событие полностью обработано, и остановить всплытие.

Для остановки всплытия нужно вызвать метод **event.stopPropagation()**.

```
<a href="#" onclick="event.stopPropagation()">Место для  
клика</a>
```

Примеры рассмотрим ниже.

Модификаторы - специальные **постфиксы**, отделяемые точкой, которые обозначают, что директива должна быть привязана каким-то особенным образом.

Несмотря на то, что это легко сделать внутри метода (средствами нативного JavaScript), лучше сохранять чистоту логики, абстрагироваться от деталей реализации событий DOM и сделать это средствами библиотеки **Vue**.

Примечание. Не смешивайте стили программирования. То, что можно сделать библиотекой Vue, сделайте **библиотекой**. Если библиотекой сделать нельзя, сделайте **нативным JavaScript**.

Для решения этой задачи Vue предоставляет **модификаторы событий** для инструкции **v-on**, которые указываются как **постфиксы** и отделяются точкой:

- **prevent**
- **stop**
- **once**

Модификатор prevent

Модификатор **prevent** дает указание директиве **v-on** вызвать **event.preventDefault()** при обработке произошедшего события, например:

```
<a href="#" @click.prevent="fnClick">Предотвращает действие по умолчанию</a>
```

example_7. Модификатор prevent

```
<div id="con">

  <p><a href="https://minobrnauki.gov.ru" v-
on:click.prevent="fnPrevent()">Минобрнауки РФ</a></p>

  <p><a href="https://vk.com/pechora_pro" v-
on:click="fnOutMessage()">Pechora_PRO</a></p>

</div>

<script>

  var app = Vue.createApp({
    methods: {
      fnPrevent () {
        alert("Переход по ссылке запрещен в силу действующего
        законодательства РФ");
      },
      fnOutMessage () {
        alert("Подвешиваем программу методом alert, но
        переход не запрещаем.");
      }
    }
  });

  vm = app.mount("#con");
```



```
</script>
```

Модификатор stop

Модификатор **stop** дает указание директиве **v-on** вызвать метод **stopPropagation()** при обработке произошедшего события, например:

```
<p @click.stop="fClick">Предотвращает всплытие событий</p>
```

example_8. Модификатор stop

```
<div id="con">

  <div @click="fClickDIV3">

    <div @click="fClickDIV2">

      <div @click="fClickDIV1">

        <p @click="fClickP1">Абзац 1 (запускает цепочку
        вызовов)</p>

        <p>Абзац 2 (не имеет своего события onclick)</p>

        <p @click.stop="fClickP3">Абзац 3 (предотвращает
        всплытие событий)</p>

      </div>

    </div>

  </div>

</div>

<script>

  var app = Vue.createApp({

    methods: {

      fClickDIV1: function() { alert("DIV 1") },

      fClickDIV2: function() { alert("DIV 2") },

      fClickDIV3: function() { alert("DIV 3") },

      fClickP1: function() { alert("P 1") },
```

```
fClickP3: function() { alert("P 3") },  
  
}  
  
}).mount("#con");  
</script>
```

Другие примеры модификаторов рассмотрим, когда будем изучать подробнее **обработчики событий**.

Работа с объектом Event

Получить детальную информацию о событии в обработчике можно посредством **объекта события** (Event). Данный объект **создаётся браузером**, когда это событие происходит.

Объект событие всегда **передается обработчику** и содержит массу полезной информации о том где и какое событие произошло. Например, для события **click**: какая клавиша нажата, координаты курсора и др.

Существует два способа передачи **объекта обработчику**, и они зависят от способа его установки и от браузера.

```
// объект event доступен по умолчанию  
  
fClick: function () {  
  
    console.log(event.target.tagName);  
  
}  
  
// объект event передается параметром  
  
fClick: function (e) {  
  
    console.log(e.target.tagName);  
  
}
```

Свойства объекта Event

Ниже представлены некоторые из свойств объекта **Event**:

- **target** – элемент, который создал событие;
- **type** – тип (имя) события.
- **timestamp** – время, когда произошло событие;
- **cancelBubble** – при установке true предотвращает всплытие события, т.е. оно всплывать не будет (является псевдонимом метода `stopPropagation`);
- **defaultPrevented** – показывает, был ли для события вызван метод `preventDefault`;

Методы объекта Event

Помимо свойств объект **Event** обладает методами:

- **preventDefault** – отменяет событие, если его можно отменить;
- **stopPropagation** – предотвращает всплытие события.

example_9. Доступ к нативному event

```
<div id="app">

  <h2>Цитаты и афоризмы Довлатова</h2>

  <label>Показать</label>

  <input type="radio" @click="fClick" name="group" id="show"
checked>

  <label>Скрыть</label>

  <input type="radio" @click="fClick" name="group" id="hide">

  <div id="bl">

    <p>Порядочный человек тот, кто делает гадости без
удовольствия.</p>

    <p>Деньги – это свобода, пространство, капризы... Имея
деньги, так легко переносить нищету...</p>

    <p>Я закуриваю только тогда, когда выпью. А пью я
беспрерывно. Поэтому многие ошибочно думают, что я
```

```
    курю.</p>

    <p>Какое это счастье — говорить, что думаешь! Какая это
    мука — думать, что говоришь!</p>

  </div>

</div>

<script>

  let app = Vue.createApp({
    methods: {

      fClick: function (event) {

        // по идентификатору элемента определяем логику
        переключений

        if (event.target.id == "show") {

          // сбрасываем свойство стиля display

          document.querySelector("#b1").style.display="";

        } else {

          // скрываем блок

          document.querySelector("#b1").style.display="none";

        }

      }

    }

  }).mount("#app");

</script>
```

Задача 1

В файле **index.html** раздаточного материала даны три ссылки. Напишите обработчик события клика по ссылке. В обработчике получите доступ к атрибутам и свойствам ссылки (**id**, **class**, **href**, и др.). Полученные данные

выведите в **консоль браузера**.

Если ссылка не имеет модификатор **prevent**, предотвратите переход по ссылке в **обработчике** через нативный **event**.

Задача 2

В файле **index.html** раздаточного материала вам предложены три блока цитат. Напишите скрипт, выполняющий управление отображением блоков в HTML-документ.

В качестве элементов управления используйте **HTML-элементы** `<input type="radio">`.

Задача 3

В директории **раздаточного материала** вам предложены исходные данные программы. После ревьюирования под стандарт спецификации **ECMAScript 6** (2015) программа "перестала" работать (хотелось как лучше ;).

Восстановите **работоспособность** программы.

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).

- **Практические работы.**