Бесплатная электронная книга

УЧУСЬ

Vue.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#vue.js

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: vue-js

It is an unofficial and free Vue.js ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Vue.js.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с Vue.js

замечания

`Vue.js` - это быстро развивающаяся инфраструктура front-end для `JavaScript` , вдохновленная `Angular.js` , `Reactive.js` и `Rivets.js` которая предлагает упрощенный дизайн пользовательского интерфейса, манипулирование и глубокую реактивность.

Он описывается как структура с `MVVM` , `Model-View View-Model` , которая основана на концепции *двухсторонней привязки* данных к компонентам и представлениям. Это невероятно быстро, превосходит скорости других систем `JS` верхнего уровня и очень удобен для легкой интеграции и прототипирования.

Версии

Версия	Дата выхода
2.4.1	2017-07-13
2.3.4	2017-06-08
2.3.3	2017-05-09
2.2.6	2017-03-26
2.0.0	2016-10-02
1.0.26	2016-06-28
1.0.0	2015-10-26
0.12.0	2015-06-12
0.11.0	2014-11-06

Examples

"Привет, мир!" программа

Чтобы начать использовать Vue.js , убедитесь, что у вас есть файл сценария, включенный в ваш HTML. Например, добавьте в свой HTML-код следующее.

```
<script src="https://npmcdn.com/vue/dist/vue.js"></script>
```

Простой пример

Шаблон HTML

```
<div id="app">
  {{ message }}
</div>
```

JavaScript

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

См. Живую демонстрацию этого примера.

Вы также можете проверить пример «Hello World», сделанный Vue.js.

Hello World в Vue 2 (путь JSX)

JSX не предназначен для интерпретации браузером. Он должен быть сначала преобразован в стандартный Javascript. Для использования JSX вам необходимо установить плагин для babel `babel-plugin-transform-vue-JSX`

Выполните команду ниже:

```
npm install babel-plugin-syntax-jsx babel-plugin-transform-vue-jsx babel-helper-vue-jsx-merge-
props --save-dev
```

и добавьте его в свой `.babelrc` следующим образом:

```
{
  "presets": ["es2015"],
  "plugins": ["transform-vue-jsx"]
}
```

Пример кода с VUE JSX:

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
  el: '#app',
```

```
  methods: {
    handleClick () {
      alert('Hello!')
    }
  },
  render (h) {
    return (
      <div>
        <h1 on-click={this.handleClick}>Hello from JSX</h1>
        <p> Hello World </p>
      </div>
    )
  }
})
```

Используя JSX, вы можете писать сжатые структуры HTML / XML в том же файле, что и код JavaScript.

Поздравляю, ты готов!

Обработка ввода пользователя

VueJS может использоваться для легкого управления вводом пользователя, а двухсторонняя привязка с использованием v-модели позволяет легко легко изменять данные.

HTML:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<div id="app">
    {{message}}
<input v-model="message">
</div>
```

JS:

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

Очень просто сделать двустороннюю привязку в VueJS, используя директиву `v-model` .

Посмотрите живой пример здесь.

Прочитайте Начало работы с Vue.js онлайн: https://riptutorial.com/ru/vue-js/topic/1057/начало-работы-с-vue-js

глава 2: VueJS + Redux с Vua-Redux (лучшее решение)

Examples

Как использовать Vua-Redux

Установка Vua Redux из NPM:

Установите через:

```
npm i vua-redux --save
```

Инициализировать:

==========

// main.js

```
import Vue from 'vue';
import { reduxStorePlugin } from 'vua-redux';
import AppStore from './AppStore';
import App from './Component/App';

// install vua-redux
Vue.use(reduxStorePlugin);

new Vue({
    store: AppStore,
    render(h) {
        return <App />
    }
});
```

// AppStore.js

```
import { createStore } from 'redux';

const initialState = {
  todos: []
};

const reducer = (state = initialState, action) => {
  switch(action.type){
    case 'ADD_TODO':
      return {
        ...state,
        todos: [...state.todos, action.data.todo]
      }
```

```
    default:
      return state;
    }
}

const AppStore = createStore(reducer);

export default AppStore;
```

Использование в вашем компоненте:

// components / App.js

```
import { connect } from 'vua-redux';

const App = {
    props: ['some-prop', 'another-prop'],

    /**
     * everything you do with vue component props
     * you can do inside collect key
     */
    collect: {
        todos: {
            type: Array,
        },
        addTodo: {
            type: Function,
        },
    },

    methods: {
        handleAddTodo() {
            const todo = this.$refs.input.value;
            this.addTodo(todo);
        }
    },

    render(h) {
        return <div>
            <ul>
                {this.todos.map(todo => <li>{todo}</li>)}
            </ul>

            <div>
                <input type="text" ref="input" />
                <button on-click={this.handleAddTodo}>add todo</button>
            </div>
        </div>
    }
};

function mapStateAsProps(state) {
    return {
        todos: state.todos
    };
}

function mapActionsAsProps(dispatch) {
```

```
    return {
        addTodo(todo) {
            dispatch({
                type: 'ADD_TODO',
                data: { todo }
            })
        }
    }
}

export default connect(mapStateAsProps, mapActionsAsProps)(App);
```

Прочитайте VueJS + Redux с Vua-Redux (лучшее решение) онлайн:
https://riptutorial.com/ru/vue-js/topic/7396/vuejs-plus-redux-c-vua-redux--лучшее-решение-

глава 3: Vuex

Вступление

Vuex - это шаблон управления состоянием + библиотека для приложений Vue.js. Он служит централизованным хранилищем для всех компонентов приложения, с правилами, гарантирующими, что состояние может быть мутировано только предсказуемым образом. Он также интегрируется с официальным расширением инструментальных средств Vue для предоставления дополнительных функций, таких как отладка с нулевым временем отклика и экспорт / импорт моментальных снимков состояния.

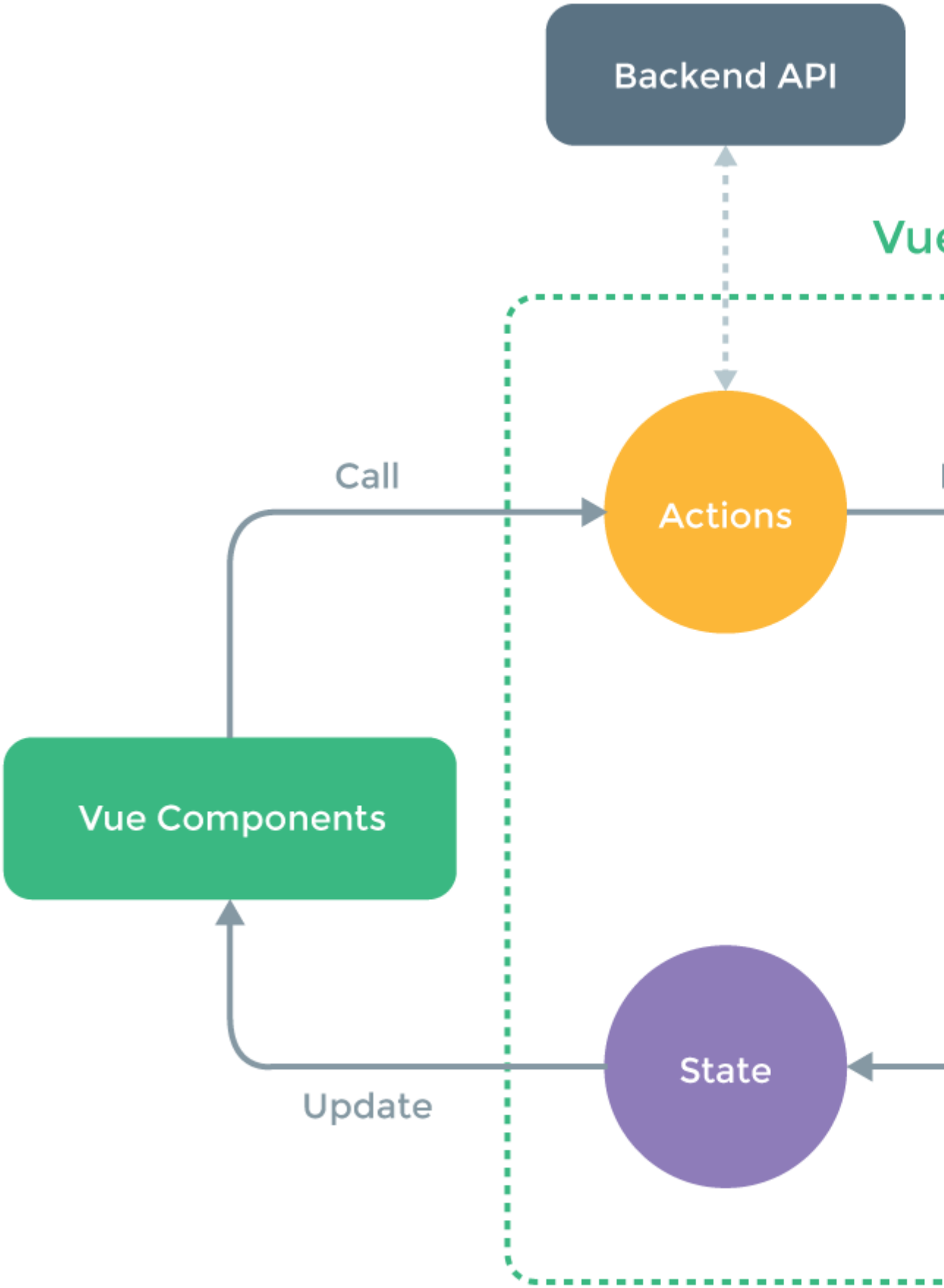
Examples

Что такое Vuex?

Vuex является официальным плагином для Vue.js, который предлагает централизованное хранилище данных для использования в вашем приложении. Это сильно зависит от архитектуры приложения Flux, которая имеет однонаправленный поток данных, что приводит к упрощению проектирования и рассуждений.

В приложении Vuex хранилище данных хранит все состояние **общего приложения** . Это состояние изменяется **мутациями,** которые выполняются в ответ на **действие,** вызывающее событие мутации через **диспетчер** .

Пример потока данных в приложении Vuex описан на диаграмме ниже.


Backend API
Vue
Call
Actions
Vue Components
State
Update

Диаграмма, используемая в соответствии с лицензией MIT , первоначально из официального репо Vuex GitHub .

Отдельные компоненты приложения Vue.js могут получить доступ к объекту хранилища для извлечения данных через **геттеры** , которые являются чистыми функциями, возвращающими постоянную копию желаемых данных.

Компоненты могут иметь **действия,** которые являются функциями, которые выполняют изменения в собственной копии данных компонента, а затем используют **диспетчер** для отправки события мутации. Затем это событие обрабатывается хранилищем данных, которое обновляет состояние по мере необходимости.

Изменения автоматически отражаются во всем приложении, так как все компоненты реагируют на хранилище через свои геттеры.

Пример , иллюстрирующий использование vuex в проекте вю.

```
const state = {
    lastClickTime: null
}

const mutations = {
    updateLastClickTime: (state, payload) => {
      state.lastClickTime = payload
  }
}

const getters = {
  getLastClickTime: state => {
    return new Date(state.lastClickTime)
  }
}

const actions = {
    syncUpdateTime: ({ commit }, payload) => {
    commit("updateLastClickTime", payload)
  },
  asyncUpdateTime: ({ commit }, payload) => {
    setTimeout(() => {
      commit("updateLastClickTime", payload)
    }, Math.random() * 5000)
  }
}

const store = new Vuex.Store({
  state,
  getters,
  mutations,
  actions
})

const { mapActions, mapGetters } = Vuex;

// Vue
const vm = new Vue({
```

```
    el: '#container',
  store,
  computed: {
      ...mapGetters([
        'getLastClickTime'
    ])
  },
  methods: {
      ...mapActions([
        'syncUpdateTime',
      'asyncUpdateTime'
    ]),
    updateTimeSyncTest () {
        this.syncUpdateTime(Date.now())
    },
    updateTimeAsyncTest () {
        this.asyncUpdateTime(Date.now())
    }
  }
})
```

И HTML-шаблон для этого же:

```
<div id="container">
  <p>{{ getLastClickTime || "No time selected yet" }}</p>
  <button @click="updateTimeSyncTest">Sync Action test</button>
  <button @click="updateTimeAsyncTest">Async Action test</button>
</div>
```

1. Здесь **состояние** содержит свойство **lastClickTime,** инициализированное как null. Эта настройка значений по умолчанию важна для сохранения **реактивности** . **Свойства, не упомянутые в состоянии,** будут доступны, но изменения, сделанные после этого **, не будут доступны** с помощью геттеров.

2. Используемый getter предоставляет вычисленное свойство, которое будет обновляться каждый раз, когда мутация обновляет значение свойства state.

3. **Только мутациям** разрешено изменять состояние и его свойства, при этом он делает это **только синхронно** .

4. Действие может быть использовано в случае асинхронных обновлений, в которых вызов API (здесь издевается случайно заданный setTimeout) может быть сделан в действии, и после получения ответа мутация может быть зафиксирована, чтобы внести изменения в состояние ,

Зачем использовать Vuex?

При создании больших приложений, таких как Single Pages Apps (SPA), которые обычно состоят из многих компонентов многократного использования, они могут быстро стать сложными для создания и поддержки. Обмен данными и состоянием между этими компонентами также может быстро разрушаться и становится трудно отлаживать и

поддерживать.

Используя хранилище данных централизованного приложения, все состояние приложения может быть представлено в одном месте, что делает приложение более организованным. Благодаря использованию однонаправленного потока данных, мутаций и доступа к областям данных, доступным только для требуемых данных, становится намного проще рассуждать о роли компонента и о том, как он должен влиять на состояние приложения.

Компоненты VueJS являются отдельными объектами, и они не могут легко обмениваться данными между собой. Для того, чтобы обмениваться данными без vuex мы должны `emit` событие с данными , а затем слушать и поймать это событие с `on` .

компонент 1

```
this.$emit('eventWithDataObject', dataObject)
```

компонент 2

```
this.$on('eventWithDataObject', function (dataObject) {
    console.log(dataObject)
})
```

С установленным vuex мы можем просто получить доступ к своим данным из любого компонента без необходимости прослушивания событий.

```
this.$store.state.myData
```

Мы также можем синхронно изменять данные с помощью *мутаторов* , использовать асинхронные *действия* и получать данные с функциями *геттера* .

Функции Getter могут работать как глобальные вычислительные функции. Мы можем получить к ним доступ из компонентов:

```
this.$store.getters.myGetter
```

Действия - это глобальные методы. Мы можем отправить их из компонентов:

```
this.$store.dispatch('myAction', myDataObject)
```

И мутации - единственный способ изменить данные в vuex. Мы можем зафиксировать изменения:

```
this.$store.commit('myMutation', myDataObject)
```

Код Vuex будет выглядеть следующим образом:

```
state: {
  myData: {
    key: 'val'
  }
},
getters: {
  myGetter: state => {
     return state.myData.key.length
  }
},
actions: {
  myAction ({ commit }, myDataObject) {
    setTimeout(() => {
      commit('myMutation', myDataObject)
    }, 2000)
  }
},
mutations: {
  myMutation (state, myDataObject) {
    state.myData = myDataObject
  }
}
```

Как установить Vuex?

Большую часть времени, когда вы будете использовать Vuex, будут в приложениях с большими компонентами, где вы, вероятно, будете использовать набор модулей, например Webpack или Browserify, вместе с Vueify, если вы используете отдельные файлы.

В этом случае самый простой способ получить Vuex - это NPM. Выполните следующую команду, чтобы установить Vuex и сохранить его в зависимости от вашего приложения.

```
 npm install --save vuex
```

Убедитесь, что вы загружаете ссылку Vuex с настройкой Vue, поместив следующую строку после инструкции `require('vue')` .

```
Vue.use(require('vuex'))
```

Vuex также доступен на CDN; вы можете взять последнюю версию cdnjs здесь .

Автоотложные уведомления

Этот пример будет динамически регистрировать vuex-модуль для хранения пользовательских уведомлений, которые могут автоматически отклоняться

notifications.js

разрешить хранилище vuex и определить некоторые константы

```
//Vuex store previously configured on other side
```

```
import _store from 'path/to/store';

//Notification default duration in milliseconds
const defaultDuration = 8000;

//Valid mutation names
const NOTIFICATION_ADDED = 'NOTIFICATION_ADDED';
const NOTIFICATION_DISMISSED = 'NOTIFICATION_DISMISSED';
```

установить начальное состояние модуля

```
const state = {
  Notifications: []
}
```

установить наши модули

```
const getters = {
  //All notifications, we are returning only the raw notification objects
  Notifications: state => state.Notifications.map(n => n.Raw)
}
```

установить наш модуль

```
const actions = {
  //On actions we receive a context object which exposes the
  //same set of methods/properties on the store instance
  //{commit} is a shorthand for context.commit, this is an
  //ES2015 feature called argument destructuring
  Add({ commit }, notification) {
    //Get notification duration or use default duration
    let duration = notification.duration || defaultDuration

    //Create a timeout to dismiss notification
    var timeOut = setTimeout(function () {
      //On timeout mutate state to dismiss notification
      commit(NOTIFICATION_DISMISSED, notification);
    }, duration);

    //Mutate state to add new notification, we create a new object
    //for save original raw notification object and timeout reference
    commit(NOTIFICATION_ADDED, {
      Raw: notification,
      TimeOut: timeOut
    })
  },
  //Here we are using context object directly
  Dismiss(context, notification) {
    //Just pass payload
    context.commit(NOTIFICATION_DISMISSED, notification);
  }
}
```

установить наши мутации модуля

```
const mutations = {
```

```
  //On mutations we receive current state and a payload
  [NOTIFICATION_ADDED](state, notification) {
    state.Notifications.push(notification);
  },
  //remember, current state and payload
  [NOTIFICATION_DISMISSED](state, rawNotification) {
    var i = state.Notifications.map(n => n.Raw).indexOf(rawNotification);
    if (i == -1) {
      return;
    }

    clearTimeout(state.Notifications[i].TimeOut);
    state.Notifications.splice(i, 1);
  }
}
```

Зарегистрируйте наш модуль с определенным состоянием, геттерами, действиями и мутацией

```
_store.registerModule('notifications', {
  state,
  getters,
  actions,
  mutations
});
```

использование

componentA.vue

Эти компоненты отображают все уведомления в виде предупреждений бутстрапа в правом верхнем углу экрана, а также позволяют вручную отклонять каждое уведомление.

```
<template>
<transition-group name="notification-list" tag="div" class="top-right">
  <div v-for="alert in alerts" v-bind:key="alert" class="notification alert alert-dismissible"
v-bind:class="'alert-'+alert.type">
    <button v-on:click="dismiss(alert)" type="button" class="close" aria-label="Close"><span
aria-hidden="true">&times;</span></button>
    <div>
      <div>
        <strong>{{alert.title}}</strong>
      </div>
      <div>
        {{alert.text}}
      </div>
    </div>
  </div>
</transition-group>
</template>

<script>
export default {
  name: 'arc-notifications',
  computed: {
    alerts() {
      //Get all notifications from store
```

```
      return this.$store.getters.Notifications;
    }
  },
  methods: {
    //Manually dismiss a notification
    dismiss(alert) {
      this.$store.dispatch('Dismiss', alert);
    }
  }
}
</script>
<style lang="scss" scoped>
$margin: 15px;

.top-right {
    top: $margin;
    right: $margin;
    left: auto;
    width: 300px;
    //height: 600px;
    position: absolute;
    opacity: 0.95;
    z-index: 100;
    display: flex;
    flex-wrap: wrap;
    //background-color: red;
}
.notification {
    transition: all 0.8s;
    display: flex;
    width: 100%;
    position: relative;
    margin-bottom: 10px;
    .close {
        position: absolute;
        right: 10px;
        top: 5px;
    }

    > div {
        position: relative;
        display: inline;
    }
}
.notification:last-child {
    margin-bottom: 0;
}
.notification-list-enter,
.notification-list-leave-active {
    opacity: 0;
    transform: translateX(-90px);
}
.notification-list-leave-active {
    position: absolute;
}
</style>
```

Фрагмент для добавления уведомлений в любой другой компонент

```
//payload could be anything, this example content matches with componentA.vue
```

```
this.$store.dispatch('Add', {
  title = 'Hello',
  text = 'World',
  type = 'info',
  duration = 15000
});
```

Прочитайте Vuex онлайн: https://riptutorial.com/ru/vue-js/topic/3430/vuex

глава 4: Автобус событий

Вступление

Коды событий - полезный способ связи между компонентами, которые не связаны напрямую, т. Е. Не имеют отношений между родителями и дочерними элементами.

Это просто пустой экземпляр `vue` , который можно использовать для `$emit` events или прослушивания `$on` указанных событиях.

Синтаксис

1. экспорт по умолчанию новый Vue ()

замечания

Используйте vuex, если в вашем приложении много компонентов, требующих данных друг друга.

Examples

eventBus

```
// setup an event bus, do it in a separate js file
var bus = new Vue()

// imagine a component where you require to pass on a data property
// or a computed property or a method!

Vue.component('card', {
  template: `<div class='card'>
    Name:
      <div class='margin-5'>
          <input v-model='name'>
    </div>
    <div class='margin-5'>
      <button @click='submit'>Save</button>
    </div>
  </div>`,
  data() {
    return {
      name: null
    }
  },
  methods: {
    submit() {
      bus.$emit('name-set', this.name)
    }
```

```
  }
})

// In another component that requires the emitted data.
var data = {
  message: 'Hello Vue.js!'
}

var demo = new Vue({
  el: '#demo',
  data: data,
  created() {
    console.log(bus)
    bus.$on('name-set', (name) => {
      this.message = name
    })
  }
})
```

Прочитайте Автобус событий онлайн: https://riptutorial.com/ru/vue-js/topic/9498/автобус-событий

глава 5: Вычислимые свойства

замечания

Данные по вычислительным свойствам

Основная разница в использовании для `data` и `computed` свойств экземпляра `Vue` зависит от потенциального состояния или вероятности изменения данных. При определении того, какой категории должен быть определенный объект, эти вопросы могут помочь:

- Это постоянное значение? (**данные**)
- Имеет ли это возможность изменить? (**вычисленные** или **данные**)
- Является ли значение этого значения зависящим от значения других данных? (**вычисленный**)
- Нужно ли, чтобы дополнительные данные или вычисления были полными до их использования? (**вычисленный**)
- Будет ли значение изменяться при определенных обстоятельствах? (**данные**)

Examples

Основной пример

шаблон

```
<div id="example">
  a={{ a }}, b={{ b }}
</div>
```

JavaScript

```
var vm = new Vue({
  el: '#example',
  data: {
    a: 1
  },
  computed: {
    // a computed getter
    b: function () {
      // `this` points to the vm instance
      return this.a + 1
    }
  }
})
```

Результат

```
a=1, b=2
```

Здесь мы объявили вычисленное свойство `b` . Функция, которую мы предоставили, будет использоваться в качестве функции геттера для свойства `vm.b` :

```
console.log(vm.b) // -> 2
vm.a = 2
console.log(vm.b) // -> 3
```

Значение `vm.b` всегда зависит от значения `vm.a`

Вы можете привязать данные к вычисленным свойствам в шаблонах так же, как и нормальное свойство. Vue знает, что `vm.b` зависит от `vm.a` , поэтому он будет обновлять любые привязки, зависящие от `vm.b` при изменении `vm.a`

Вычислительные свойства против часов

шаблон

```
<div id="demo">{{fullName}}</div>
```

пример просмотра

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  }
})

vm.$watch('firstName', function (val) {
  this.fullName = val + ' ' + this.lastName
})

vm.$watch('lastName', function (val) {
  this.fullName = this.firstName + ' ' + val
})
```

Вычисленный пример

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
```

```
  }
})
```

Вычислительные сеттеры

Вычислимые свойства будут автоматически пересчитываться всякий раз, когда любые данные, от которых зависит вычисление, изменяются. Однако, если вам нужно вручную изменить вычисленное свойство, Vue позволяет вам создать метод setter для этого:

Шаблон (из основного примера выше):

```
<div id="example">
  a={{ a }}, b={{ b }}
</div>
```

Javascript:

```
var vm = new Vue({
  el: '#example',
  data: {
    a: 1
  },
  computed: {
    b: {
      // getter
      get: function () {
        return this.a + 1
      },
      // setter
      set: function (newValue) {
        this.a = newValue - 1
      }
    }
  }
```

Теперь вы можете вызвать либо геттер, либо сеттер:

```
console.log(vm.b)       // -> 2
vm.b = 4                // (setter)
console.log(vm.b)       // -> 4
console.log(vm.a)       // -> 3
```

`vm.b = 4` будет вызывать установщик и установить this.a в 3; по расширению, vm.b будет оцениваться до 4.

Использование вычисленных сеттеров для v-модели

Возможно, вам понадобится `v-model` для вычисленного свойства. Как правило, v-модель не будет обновлять значение вычисленного значения.

Шаблон:

```
<div id="demo">
  <div class='inline-block card'>
    <div :class='{onlineMarker: true, online: status, offline: !status}'></div>
    <p class='user-state'>User is {{ (status) ? 'online' : 'offline' }}</p>
  </div>

  <div class='margin-5'>
    <input type='checkbox' v-model='status'>Toggle status (This will show you as offline to
others)
  </div>
</div>
```

Стайлинг:

```
#demo {
  font-family: Helvetica;
  font-size: 12px;
}
.inline-block > * {
  display: inline-block;
}
.card {
  background: #ddd;
  padding:2px 10px;
  border-radius: 3px;
}
.onlineMarker {
  width: 10px;
  height: 10px;
  border-radius: 50%;
  transition: all 0.5s ease-out;
}

.online {
  background-color: #3C3;
}

.offline {
  background-color: #aaa;
}

.user-state {
  text-transform: uppercase;
  letter-spacing: 1px;
}

.margin-5 {
  margin: 5px;
}
```

Компонент:

```
var demo = new Vue({
    el: '#demo',
    data: {
        statusProxy: null
```

```
    },
    computed: {
        status: {
            get () {
                return (this.statusProxy === null) ? true : this.statusProxy
            }
        }
    }
})
```

скрипка. Здесь вы увидите, что щелчок по кнопке не имеет никакого значения, ваш статус по-прежнему в сети.

```
var demo = new Vue({
    el: '#demo',
    data: {
        statusProxy: null
    },
    computed: {
        status: {
            get () {
                return (this.statusProxy === null) ? true : this.statusProxy
            },
            set (val) {
                this.statusProxy = val
            }
        }
    }
})
```

fiddle И теперь вы можете видеть, что переключатель происходит, когда флажок установлен / снят флажок.

Прочитайте Вычислимые свойства онлайн: https://riptutorial.com/ru/vue-js/topic/2371/вычислимые-свойства

глава 6: вя-маршрутизатор

Вступление

vue-router - официально поддерживаемая библиотека маршрутизации для vue.js.

Синтаксис

- `<router-link to="/path">Link Text</router-link> <!-- Creates a link to the route that matches the path -->`
- `<router-view></router-view> <!-- Outlet for the currently matched route. It's component will be rendered here. -->`

Examples

Основная маршрутизация

Самый простой способ встать и работать с vue-router - использовать версию, предоставленную через CDN.

HTML:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="router-example">
    <router-link to="/foo">Link to Foo route</router-link>
    <router-view></router-view>
</div>
```

JavaScript (ES2015):

```
const Foo = { template: <div>This is the component for the Foo route</div> }

const router = new VueRouter({
    routes: [
       { path: '/foo', component: Foo}
    ]
})

const routerExample = new Vue({
    router
}).$mount('#router-example')
```

Прочитайте вя-маршрутизатор онлайн: https://riptutorial.com/ru/vue-js/topic/9654/вя-маршрутизатор

глава 7: Динамические компоненты

замечания

`<component>` - зарезервированный элемент компонента, не путайте с экземпляром компонентов.

`v-bind` - это директива. Директивы имеют префикс *v-*, чтобы указать, что они являются особыми атрибутами, предоставленными Vue.

Examples

Пример простых динамических компонентов

Динамически переключаться между несколькими компонентами с помощью элемента `<component>` и передавать данные в *v-bind: is* attribute:

Javascript:

```
new Vue({
  el: '#app',
  data: {
    currentPage: 'home'
  },
  components: {
    home: {
      template: "<p>Home</p>"
    },
    about: {
      template: "<p>About</p>"
    },
    contact: {
      template: "<p>Contact</p>"
    }
  }
})
```

HTML:

```
<div id="app">
   <component v-bind:is="currentPage">
       <!-- component changes when currentPage changes! -->
       <!-- output: Home -->
   </component>
</div>
```

Snippet:

Демо-версия

Навигация по страницам с сохранением активности

Иногда вы хотите сохранить отключенные компоненты в памяти, чтобы это произошло, вы должны использовать элемент `<keep-alive>` :

Javascript:

```
new Vue({
  el: '#app',
  data: {
    currentPage: 'home',
  },
  methods: {
     switchTo: function(page) {
            this.currentPage = page;
     }
  },
  components: {
    home: {
      template: `<div>
                 <h2>Home</h2>
                 <p>{{ homeData }}</p>
                 </div>`,
      data: function() {
         return {
            homeData: 'My about data'
         }
       }
     },
    about: {
      template: `<div>
                 <h2>About</h2>
                 <p>{{ aboutData }}</p>
                 </div>`,
      data: function() {
         return {
            aboutData: 'My about data'
         }
      }
    },
    contact: {
      template: `<div>
             <h2>Contact</h2>
             <form method="POST" @submit.prevent>
             <label>Your Name:</label>
             <input type="text" v-model="contactData.name" >
             <label>You message: </label>
             <textarea v-model="contactData.message"></textarea>
             <button type="submit">Send</button>
             </form>
             </div>`,
       data: function() {
```

```
          return {
            contactData: { name:'', message:'' }
          }
        }
      }
    }
})
```

HTML:

```
<div id="app">
  <div class="navigation">
    <ul>
      <li><a href="#home" @click="switchTo('home')">Home</a></li>
      <li><a href="#about" @click="switchTo('about')">About</a></li>
      <li><a href="#contact" @click="switchTo('contact')">Contact</a></li>
    </ul>
  </div>

  <div class="pages">
    <keep-alive>
      <component :is="currentPage"></component>
    </keep-alive>
  </div>
</div>
```

CSS:

```
.navigation {
  margin: 10px 0;
}

.navigation ul {
  margin: 0;
  padding: 0;
}

.navigation ul li {
  display: inline-block;
  margin-right: 20px;
}

input, label, button {
  display: block
}

input, textarea {
  margin-bottom: 10px;
}
```

Snippet:

Демо-версия

Прочитайте Динамические компоненты онлайн: https://riptutorial.com/ru/vue-js/topic/7702/динамические-компоненты

глава 8: Зрителей

Examples

Как это устроено

Вы можете просмотреть свойство данных любого экземпляра Vue. При просмотре свойства вы вызываете метод при изменении:

```
export default {
    data () {
        return {
            watched: 'Hello World'
        }
    },
    watch: {
        'watched' () {
            console.log('The watched property has changed')
        }
    }
}
```

Вы можете получить старое значение и новое:

```
export default {
    data () {
        return {
            watched: 'Hello World'
        }
    },
    watch: {
        'watched' (value, oldValue) {
            console.log(oldValue) // Hello World
            console.log(value) // ByeBye World
        }
    },
    mounted () {
        this.watched = 'ByeBye World'
    }
}
```

Если вам нужно посмотреть вложенные свойства объекта, вам нужно будет использовать свойство `deep` :

```
export default {
    data () {
        return {
            someObject: {
                message: 'Hello World'
            }
        }
    },
```

```
    watch: {
        'someObject': {
            deep: true,
            handler (value, oldValue) {
                console.log('Something changed in someObject')
            }
        }
    }
}
```

Когда обновляются данные?

Если вам нужно вызвать наблюдателя перед внесением новых изменений в объект, вам нужно использовать метод `nextTick()` :

```
export default {
    data() {
        return {
            foo: 'bar',
            message: 'from data'
        }
    },
    methods: {
        action () {
            this.foo = 'changed'
            // If you juste this.message = 'from method' here, the watcher is executed after.
            this.$nextTick(() => {
                this.message = 'from method'
            })
        }
    },
    watch: {
        foo () {
            this.message = 'from watcher'
        }
    }
}
```

Прочитайте Зрителей онлайн: https://riptutorial.com/ru/vue-js/topic/7988/зрителей

глава 9: игровые автоматы

замечания

Важный! Слоты после рендера не гарантируют порядок позиций для слотов. Слот, который был первым, может иметь другую позицию после рендера.

Examples

Использование отдельных слотов

Отдельные слоты используются, когда дочерний компонент определяет только один `slot` в своем шаблоне. Компонент `page` выше использует один слот для распространения контента.

Пример шаблона компонента `page` , использующего один слот, приведен ниже:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <slot>
            This will only be displayed if there is no content
            to be distributed.
        </slot>
    </body>
</html>
```

Чтобы проиллюстрировать, как работает слот, мы можем настроить страницу следующим образом.

```
<page>
    <p>This content will be displayed within the page component</p>
</page>
```

Конечным результатом будет:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <p>This content will be displayed within the page component</p>
    </body>
</html>
```

Если бы мы не помещали что-либо между тегами `page` вместо этого вместо `<page></page>` мы получили бы следующий результат, так как в `<page></page>` компонента `page` есть

содержимое по умолчанию между тегами `slot` .

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        This will only be displayed if there is no content
        to be distributed.
    </body>
</html>
```

Что такое слоты?

Слоты предлагают удобный способ распространения контента из родительского компонента в дочерний компонент. Этот контент может быть любым из текста, HTML или даже других компонентов.

Иногда бывает полезно подумать о слотах, как о средствах инъекции контента непосредственно в шаблон дочернего компонента.

Слоты особенно полезны, когда состав компонентов под родительским компонентом не всегда одинаковый.

Возьмем следующий пример, где у нас есть компонент `page` . Содержимое страницы может меняться в зависимости от того, отображается ли эта страница, например, статья, сообщение в блоге или форму.

Статья

```
<page>
    <article></article>
    <comments></comments>
</page>
```

Сообщение блога

```
<page>
    <blog-post></blog-post>
    <comments></comments>
</page>
```

форма

```
<page>
    <form></form>
</page>
```

Обратите внимание, как содержимое компонента `page` может измениться. Если бы мы не использовали слоты, это было бы сложнее, поскольку внутренняя часть шаблона была бы

исправлена.

Помните: *«Все в родительском шаблоне скомпилировано в родительской области: все в дочернем шаблоне скомпилировано в области содержимого».*

Использование именованных слотов

Именованные слоты работают аналогично отдельным слотам, но вместо этого позволяют распространять контент в разных регионах в шаблоне дочернего компонента.

Возьмите компонент `page` из предыдущего примера, но измените его шаблон, чтобы он выглядел следующим образом:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <aside>
            <slot name="sidebar"></slot>
        </aside>
        <main>
            <slot name="content"></slot>
        </main>
    </body>
</html>
```

При использовании компонента `page` мы можем теперь определить, где содержимое помещено через атрибут `slot` :

```
<page>
    <p slot="sidebar">This is sidebar content.</p>
    <article slot="content"></article>
</page>
```

Результатом будет страница:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <aside>
            <p>This is sidebar content.</p>
        </aside>
        <main>
            <article></article>
        </main>
    </body>
</html>
```

Если `slot` определен без `name` атрибута тогда любое содержимое , которое находится в

пределах компонентов тегов не задающих в `slot` атрибут будет помещен в этот слот.

См. Пример множественной вставки в официальных документах Vue.js.

Использование слотов в Vue JSX с помощью 'babel-plugin-transform-vue-jsx'

Если вы используете VueJS2 и хотите использовать JSX вместе с ним. В этом случае, чтобы использовать слот, решение с примером ниже. Мы должны использовать `this.$slots.default` Это почти как `this.props.children` в React JS.

Component.js:

```
export default {
    render(h) {  //eslint-disable-line
        return (
            <li>
                { this.$slots.default }
            </li>
        );
    }
};
```

ParentComponent.js

```
import Component from './Component';

export default {
    render(h) {  //eslint-disable-line
        return (
            <ul>
                <Component>
                    Hello World
                </Component>
            </ul>
        );
    }
};
```

Прочитайте игровые автоматы онлайн: https://riptutorial.com/ru/vue-js/topic/4484/игровые-автоматы

глава 10: Использование «this» в Vue

Вступление

Одной из наиболее распространенных ошибок, которые мы обнаруживаем в коде Vue на StackOverflow, является неправильное использование `this` . Наиболее распространенные ошибки обычно падают в двух областях, используя `this` в обратных вызовах для обещаний или других асинхронных функций и используя функции стрелок для определения методов, вычисленных свойств и т. Д.

Examples

НЕПРАВИЛЬНО! Использование «this» в обратном вызове внутри метода Vue.

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomethingAsynchronous(){
      setTimeout(function(){
        // This is wrong! Inside this function,
        // "this" refers to the window object.
        this.foo = "baz";
      }, 1000);
    }
  }
})
```

НЕПРАВИЛЬНО! Использование «этого» внутри обещания.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    $.getJSON("http://swapi.co/api/people/", function(data){
      // Again, this is wrong! "this", here, refers to the window.
      this.people = data.results;
    })
  }
})
```

ПРАВО! Используйте закрытие для захвата "этого"

Вы можете захватить правильное `this` с помощью закрытия .

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    // Before executing the web service call, save this to a local variable
    var self = this;
    $.getJSON("http://swapi.co/api/people/", function(data){
      // Inside this call back, because of the closure, self will
      // be accessible and refers to the Vue object.
      self.people = data.results;
    })
  }
})
```

ПРАВО! Используйте bind.

Вы можете связать функцию обратного вызова.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted:function(){
    $.getJSON("http://swapi.co/api/people/", function(data){
      this.people = data.results;
    }.bind(this));
  }
})
```

ПРАВО! Используйте функцию стрелки.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    $.getJSON("http://swapi.co/api/people/", data => this.people = data.results);
  }
})
```

Внимание! Функции Arrow - это синтаксис, введенный в Ecmascript 2015. Он еще не поддерживается, но *все* современные браузеры, поэтому используйте его только в том случае, если вы нацеливаете браузер, который, как вы *знаете,* поддерживает его, или если вы компилируете свой javascript до синтаксиса ES5, используя что-то вроде babel ,

НЕПРАВИЛЬНО! Используя функцию стрелки для определения метода,

который ссылается на «это»,

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    // This is wrong! Arrow functions capture "this" lexically
    // and "this" will refer to the window.
    doSomething: () => this.foo = "baz"
  }
})
```

ПРАВО! Определить методы с типичным синтаксисом функции

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomething: function(){
      this.foo = "baz"
    }
  }
})
```

Кроме того, если вы используете компилятор javascript или браузер, поддерживающий Ecmascript 2015

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomething(){
      this.foo = "baz"
    }
  }
})
```

Прочитайте Использование «this» в Vue онлайн: https://riptutorial.com/ru/vue-js/topic/9350/использование--this--в-vue

глава 11: Компоненты

замечания

В компоненте (-ах):

props - это массив строковых литералов или ссылок на объекты, используемые для передачи данных из родительского компонента. Он также может быть в виде объекта, когда желательно иметь более мелкозернистый контроль, например, задавать значения по умолчанию, тип принятых данных, независимо от того, требуется ли это или необязательно

данные должны быть функцией, которая возвращает объект вместо простого объекта. Это потому, что мы требуем, чтобы каждый экземпляр компонента имел свои собственные данные для цели повторного использования.

события - это объект, содержащий слушателей для событий, на которые компонент может реагировать поведением

метод, содержащий функции, определяющие поведение, связанное с компонентом

вычисляемые свойства точно так же, как наблюдатели или наблюдаемые, всякий раз, когда любая зависимость изменяет свойства, пересчитываются автоматически, и изменения отражаются в DOM мгновенно, если DOM использует любые вычисленные значения

ready - это цикл жизненного цикла экземпляра Vue

Examples

Компонент (не глобальный)

демонстрация

HTML

```
<script type="x-template" id="form-template">
    <label>{{inputLabel}} :</label>
  <input type="text" v-model="name" />
</script>

<div id="app">
  <h2>{{appName}}</h2>
  <form-component title="This is a form" v-bind:name="userName"></form-component>
</div>
```

JS

```
// Describe the form component
// Note: props is an array of attribute your component can take in entry.
// Note: With props you can pass static data('title') or dynamic data('userName').
// Note: When modifying 'name' property, you won't modify the parent variable, it is only
descendent.
// Note: On a component, 'data' has to be a function that returns the data.
var formComponent = {
  template: '#form-template',
  props: ['title', 'name'],
  data: function() {
    return {
      inputLabel: 'Name'
    }
  }
};

// This vue has a private component, it is only available on its scope.
// Note: It would work the same if the vue was a component.
// Note: You can build a tree of components, but you have to start the root with a 'new
Vue()'.
var vue = new Vue({
  el: '#app',
  data: {
    appName: 'Component Demo',
    userName: 'John Doe'
  },
  components: {
    'form-component': formComponent
  }
});
```

Какие компоненты и как определить компоненты?

Компоненты в Vue похожи на виджеты. Они позволяют нам писать повторно используемые пользовательские элементы с желаемым поведением.

Они представляют собой объекты, которые могут содержать любые / все параметры, которые могут содержать корневой или любой экземпляр Vue, включая HTML-шаблон для рендеринга.

Компоненты состоят из:

- HTML-разметка: шаблон компонента
- Стили CSS: как будет отображаться разметка HTML
- Код JavaScript: данные и поведение

Каждый из них может быть записан в отдельный файл или в виде одного файла с расширением `.vue` . Ниже приведены примеры, показывающие оба способа:

.VUE - как отдельный файл для компонента

```
<style>
    .hello-world-compoment{
        color:#eeeeee;
        background-color:#555555;
    }
</style>

<template>
    <div class="hello-world-component">
        <p>{{message}}</p>
        <input @keyup.enter="changeName($event)"/>
    </div>
</template>

<script>
    export default{
        props:[ /* to pass any data from the parent here... */ ],
        events:{ /* event listeners go here */},
        ready(){
            this.name= "John";
        },
        data(){
           return{
              name:''
           }
        },
        computed:{
            message(){
                return "Hello from " + this.name;
            }
        },
        methods:{
            // this could be easily achieved by using v-model on the <input> field, but just
to show a method doing it this way.
            changeName(e){
                this.name = e.target.value;
            }
        }
    }
</script>
```

Отдельные файлы

hello-world.js - файл JS для объекта компонента

```
export default{
    template:require('./hello-world.template.html'),
    props:[ /* to pass any data from the parent here... */ ],
    events:{ /* event listeners go here */ },
    ready(){
        this.name="John";
    },
    data(){
        return{
            name:''
        }
    },
    computed:{
        message(){
```

```
            return "Hello World! from " + this.name;
        }
    },
    methods:{
        changeName(e){
            let name = e.target.value;
            this.name = name;
        }
    }
}
```

привет-world.template.html

```
<div class="hello-world-component">
    <p>{{message}}</p>
    <input class="form-control input-sm" @keyup.enter="changeName($event)">
</div>
```

привет-world.css

```
 .hello-world-compoment{
    color:#eeeeee;
    background-color:#555555;
}
```

В этих примерах используется синтаксис es2015, поэтому Babel потребуется для их компиляции в es5 для старых браузеров.
Для компиляции `hello-world.vue` потребуется **Babel** вместе с **Browserify + vueify** или **Webpack + vue-loader** .

Теперь, когда мы определили компонент `hello-world` , мы должны зарегистрировать его с помощью Vue.

Это можно сделать двумя способами:

Зарегистрируйтесь как глобальный компонент
В файле `main.js` (точка входа в приложение) мы можем зарегистрировать любой компонент по всему миру с помощью `Vue.component` :

```
import Vue from 'vue'; // Note that 'vue' in this case is a Node module installed with 'npm
install Vue'
Vue.component('hello-world', require('./hello-world');  // global registeration

new Vue({
    el:'body',

    // Templates can be defined as inline strings, like so:
    template:'<div class="app-container"><hello-world></hello-world></div>'
});
```

Или зарегистрируйте его локально в родительском компоненте или корневом компоненте

```
import Vue from 'vue'; // Note that 'vue' in this case is a Node module installed with 'npm
install Vue'
import HelloWorld from './hello-world.js';

new Vue({
    el:'body',
    template:'<div class="app-container"><hello-world></hello-world></div>",

    components:{HelloWorld}  // local registeration
});
```

Глобальные компоненты могут использоваться в любом месте приложения Vue.

Локальные компоненты доступны только для использования в родительском компоненте, с которым они зарегистрированы.

Компонент фрагмента
Вы можете получить консольную ошибку, сообщающую вам, что вы не можете что-то сделать, потому что ваш *компонент* является *фрагментом* . Чтобы решить эту проблему, просто оберните шаблон компонента внутри одного тега, например, `<div>` .

Локальная регистрация компонентов

Компонент может быть зарегистрирован либо глобально, либо локально (привязка к другому конкретному компоненту).

```
var Child = Vue.extend({
    // ...
})

var Parent = Vue.extend({
    template: '...',
    components: {
         'my-component': Child
    }
})
```

Новый компонент () будет доступен только внутри области (шаблона) родительского компонента.

Встроенная регистрация

Вы можете продлить и зарегистрировать компонент за один шаг:

```
Vue.component('custom-component', {
    template: '<div>A custom component!</div>'
})
```

Также, когда компонент зарегистрирован локально:

```
var Parent = Vue.extend({
    components: {
        'custom-component': {
            template: '<div>A custom component!</div>'
        }
    }
})
```

Регистрация данных в компонентах

Передача объекта в свойство `data` при регистрации компонента приведет к тому, что все экземпляры компонента будут указывать на одни и те же данные. Чтобы решить эту проблему, нам нужно вернуть `data` из функции.

```
var CustomComponent = Vue.extend({
    data: function () {
        return { a: 1 }
    }
})
```

События

Одним из способов взаимодействия компонентов со своими предками / потомками является использование пользовательских событий связи. Все экземпляры Vue также являются эмиттерами и реализуют настраиваемый интерфейс событий, который облегчает обмен данными внутри дерева компонентов. Мы можем использовать следующее:

- `$on` : Слушайте события, испускаемые этими компонентами предков или потомков.
- `$broadcast` : Выдает событие, распространяющееся вниз ко всем потомкам.
- `$dispatch` : испускает событие, которое сначала запускает сам компонент и распространяется до всех предков.
- `$emit` : запускает событие для себя.

Например, мы хотим скрыть конкретный компонент кнопки внутри компонента формы, когда форма отправляется. На родительском элементе:

```
var FormComponent = Vue.extend({
  // ...
  components: {
    ButtonComponent
  },
  methods: {
    onSubmit () {
        this.$broadcast('submit-form')
    }
  }
})
```

О дочернем элементе:

```
var FormComponent = Vue.extend({
  // ...
  events: {
    'submit-form': function () {
        console.log('I should be hiding');
    }
  }
})
```

Некоторые вещи нужно иметь в виду:

- Всякий раз , когда событие находит компонент , который слушает его и получает срабатывает, он остановится распространяющимся , если функция обратного вызов в этом компоненте не возвращает `true` .
- `$dispatch()` всегда запускает сначала компонент, который его испустил.
- Мы можем передать любое количество аргументов обработчику событий. Выполнение `this.$broadcast('submit-form', this.formData, this.formStatus)` позволяет нам получить доступ к этим аргументам, таким как `'submit-form': function (formData, formStatus) {}`

Прочитайте Компоненты онлайн: https://riptutorial.com/ru/vue-js/topic/1775/компоненты

глава 12: Компоненты VUE для одного файла

Вступление

Опишите, как создавать отдельные файловые компоненты в файле .vue.

Специально проектные решения, которые могут быть сделаны.

Examples

Пример файла компонента .vue

```
<template>
  <div class="nice">Component {{title}}</div>
</template>

<script>
export default {
    data() {
        return {
            title: "awesome!"
        };
    }
}
</script>

<style>
.nice {
    background-color: red;
    font-size: 48px;
}
</style>
```

Прочитайте Компоненты VUE для одного файла онлайн: https://riptutorial.com/ru/vue-js/topic/10118/компоненты-vue-для-одного-файла

глава 13: Крючки жизненного цикла

Examples

Крючки для Vue 1.x

- **`init`**

  Вызывается синхронно после того, как экземпляр был инициализирован и до любого начального наблюдения данных.

- **`created`**

  Вызывается синхронно после создания экземпляра. Это происходит до установки `$el` , но после `data observation computed properties` , `watch/event callbacks` и `methods` .

- **`beforeCompile`**

  Непосредственно перед компиляцией экземпляра Vue.

- **`compiled`**

  Сразу после завершения компиляции. Все `directives` связаны, но все еще до появления `$el` .

- **`ready`**

  Происходит после компиляции *и* `$el` завершаются, и экземпляр вводится в DOM впервые.

- **`attached`**

  Возникает, когда `$el` привязывается к DOM `directive` или экземпляром, вызывает `$appendTo()` .

- **`detached`**

  Вызывается, когда `$el` удаляется / удаляется из метода DOM или экземпляра.

- **`beforeDestroy`**

  Непосредственно перед тем, как экземпляр Vue будет уничтожен, но все еще полностью работоспособен.

- **`destroyed`**

Вызывается после уничтожения экземпляра. Все `bindings` и `directives` уже были несвязаны, а дочерние экземпляры также были уничтожены.

Использование в экземпляре

Поскольку *все* крюки в течение жизненного цикла `Vue.js` являются только `functions` , вы можете поместить *любого* из них непосредственно в экземпляре declaraction.

```
//JS
new Vue({

    el: '#example',

    data: {
        ...
    },

    methods: {
        ...
    },

    //LIFECYCLE HOOK HANDLING
    created: function() {
        ...
    },

    ready: function() {
        ...
    }

});
```

Общие ошибки: доступ к DOM с помощью `ready ()` hook

Обычным способом использования `ready()` является доступ к DOM, например, для запуска плагина Javascript, получения размеров элемента и т. Д.

Эта проблема

Из-за асинхронного механизма обновления DOM Vue не гарантируется, что DOM полностью обновляется при вызове hook `ready()` . Обычно это приводит к ошибке, поскольку элемент не определен.

Решение

Для этой ситуации метод `$nextTick()` может помочь. Этот метод отменяет выполнение предоставленной функции обратного вызова до следующего тика, что означает, что он запускается, когда все обновления DOM гарантированно завершены.

Пример:

```
module.exports {
  ready: function () {
    $('.cool-input').initiateCoolPlugin() //fails, because element is not in DOM yet.

    this.$nextTick(function() {
      $('.cool-input').initiateCoolPlugin() // this will work because it will be executed
after the DOM update.
    })
  }
}
```

Прочитайте Крючки жизненного цикла онлайн: https://riptutorial.com/ru/vue-js/topic/1852/крючки-жизненного-цикла

глава 14: Модификаторы

Вступление

Существуют некоторые часто используемые операции, такие как `event.preventDefault()` или `event.stopPropagation()` внутри обработчиков событий. Хотя мы можем сделать это легко внутри методов, было бы лучше, если бы методы могли быть чисто логикой данных, а не иметь дело с деталями событий DOM.

Examples

Модификаторы событий

Vue предоставляет модификаторы событий для `v-on` , вызывая директивы postfixes, обозначенные точкой.

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`

Например:

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- use capture mode when adding the event listener -->
<div v-on:click.capture="doThis">...</div>

<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div v-on:click.self="doThat">...</div>
```

Модификаторы клавиш

При прослушивании событий клавиатуры нам часто приходится проверять наличие общих кодов клавиш. Помнить, что все ключевые коды - это проблема, поэтому Vue предоставляет псевдонимы для наиболее часто используемых ключей:

- `.enter`
- `.tab`
- `.delete` (фиксирует клавиши «Delete» и «Backspace»)
- `.esc`
- `.space`

- `.up`
- `.down`
- `.left`
- `.right`

Например:

```
<input v-on:keyup.enter="submit">
```

Модификаторы ввода

- `.trim`

Если вы хотите, чтобы пользовательский ввод был обрезан автоматически, вы можете добавить модификатор `trim` к управляемым входам `v-model` :

```
<input v-model.trim="msg">
```

- `.number`

Если вы хотите, чтобы пользовательский ввод автоматически определялся как число, вы можете сделать следующее:

```
<input v-model.number="age" type="number">
```

- `.lazy`

Как правило, `v-model` синхронизирует вход с данными после каждого входного события, но вы можете добавить `lazy` модификатор вместо синхронизации после событий изменения:

```
<input v-model.lazy="msg" >
```

Прочитайте Модификаторы онлайн: https://riptutorial.com/ru/vue-js/topic/8612/модификаторы

глава 15: Плагины

Вступление

Плагины Vue добавляют глобальную функциональность, как глобальные методы, директивы, переходы, фильтры, методы экземпляра, объекты и вводят некоторые параметры компонента, используя mixins

Синтаксис

- MyPlugin.install = function (Vue, options) {}

параметры

название	Описание
Вью	Конструктор Vue, введенный Vue
опции	Дополнительные параметры, если необходимо

замечания

В большинстве случаев вам нужно явно указать Vue на использование плагина

```
// calls `MyPlugin.install(Vue)`
Vue.use(MyPlugin)
```

Чтобы передать параметры

```
Vue.use(MyPlugin, { someOption: true })
```

Examples

Простой регистратор

```
//myLogger.js
export default {

  install(Vue, options) {
     function log(type, title, text) {
       console.log(`[${type}] ${title} - ${text}`);
     }
```

```
    Vue.prototype.$log = {
      error(title, text) { log('danger', title, text) },
      success(title, text) { log('success', title, text) },
      log
    }
  }
}
```

Перед тем, как ваш основной экземпляр Vue скажет вам зарегистрировать ваш плагин

```
//main.js
import Logger from './path/to/myLogger';

Vue.use(Logger);

var vm = new Vue({
  el: '#app',
  template: '<App/>',
  components: { App }
})
```

Теперь вы можете вызвать `this.$log` на любом дочернем компоненте

```
//myComponent.vue
export default {
  data() {
    return {};
  },
  methods: {
    Save() {
      this.$log.success('Transaction saved!');
    }
  }
}
```

Прочитайте Плагины онлайн: https://riptutorial.com/ru/vue-js/topic/8726/плагины

глава 16: Пользовательские директивы

Синтаксис

- `Vue.directive(id, definition);`
- `Vue.directive(id, update); //when you need only the update function.`

параметры

параметр	подробности
`id`	String - идентификатор директивы, который будет использоваться без префикса `v-` . (Добавьте префикс `v-` при его использовании)
`definition`	Объект. Объект определения может предоставлять несколько функций перехвата (все необязательные): `bind` , `update` и `unbind`

Examples

ОСНОВЫ

Помимо стандартного набора директив, поставляемых в ядре, Vue.js также позволяет регистрировать пользовательские директивы. Пользовательские директивы предоставляют механизм для преобразования изменений данных в произвольное поведение DOM.

Вы можете зарегистрировать глобальную настраиваемую директиву с помощью `Vue.directive(id, definition)` , передавая идентификатор директивы, за которым следует объект определения. Вы также можете зарегистрировать локальную настраиваемую директиву, включив ее в параметр `directives` компонента.

Крюковые функции

- **bind** : вызов только один раз, когда директива сначала привязана к элементу.
- **update** : вызывается в первый раз сразу после `bind` с начальным значением, затем снова, когда изменяется значение привязки. В качестве аргумента представлены новое значение и предыдущее значение.
- **unbind** : вызов только один раз, когда директива не связана с элементом.

```
Vue.directive('my-directive', {
     bind: function () {
       // do preparation work
       // e.g. add event listeners or expensive stuff
```

```
    // that needs to be run only once
  },
  update: function (newValue, oldValue) {
    // do something based on the updated value
    // this will also be called for the initial value
  },
  unbind: function () {
    // do clean up work
    // e.g. remove event listeners added in bind()
  }
})
```

После регистрации, вы можете использовать его в шаблонах Vue.js , как это (не забудьте добавить `v-` префикс):

```
<div v-my-directive="someValue"></div>
```

Когда вам нужна только функция `update` , вы можете передать только одну функцию вместо объекта определения:

```
Vue.directive('my-directive', function (value) {
  // this function will be used as update()
})
```

Свойства экземпляра директивы

Все функции крючков будут скопированы в фактическую директиве объект, который вы можете получить доступ к этим функциям в качестве `this` контекста. Объект директивы предоставляет некоторые полезные свойства:

- **el** : элемент, к которому привязана директива.
- **vm** : контекст ViewModel, которому принадлежит эта директива.
- **выражение** : выражение привязки, исключая аргументы и фильтры.
- **arg** : аргумент, если он присутствует.
- **name** : имя директивы, без префикса.
- **модификаторы** : объект, содержащий модификаторы, если они есть.
- **дескриптор** : объект, содержащий результат синтаксического анализа всей директивы.
- **params** : объект, содержащий атрибуты param. Разъясняется ниже.

  Вы должны рассматривать все эти свойства как доступные только для чтения и никогда не изменять их. Вы также можете прикреплять настраиваемые свойства к объекту директивы, но будьте осторожны, чтобы случайно не перезаписать существующие внутренние.

Пример пользовательской директивы, использующей некоторые из этих свойств:

HTML

```
<div id="demo" v-demo:hello.a.b="msg"></div>
```

JavaScript

```
Vue.directive('demo', {
  bind: function () {
    console.log('demo bound!')
  },
  update: function (value) {
    this.el.innerHTML =
      'name - '       + this.name + '<br>' +
      'expression - ' + this.expression + '<br>' +
      'argument - '   + this.arg + '<br>' +
      'modifiers - '  + JSON.stringify(this.modifiers) + '<br>' +
      'value - '      + value
  }
})
var demo = new Vue({
  el: '#demo',
  data: {
    msg: 'hello!'
  }
})
```

Результат

```
name - demo
expression - msg
argument - hello
modifiers - {"b":true,"a":true}
value - hello!
```

Объект Literal

Если ваша директива требует нескольких значений, вы также можете передать литерал объекта JavaScript. Помните, что директивы могут принимать любое допустимое выражение JavaScript:

HTML

```
<div v-demo="{ color: 'white', text: 'hello!' }"></div>
```

JavaScript

```
Vue.directive('demo', function (value) {
  console.log(value.color) // "white"
  console.log(value.text) // "hello!"
})
```

Литературный модификатор

Когда директива используется с модификатором literal, ее значение атрибута будет

интерпретироваться как простая строка и передаваться непосредственно в метод `update` . Метод `update` также будет вызываться только один раз, потому что простая строка не может быть реактивной.

HTML

```
<div v-demo.literal="foo bar baz">
```

JavaScript

```
Vue.directive('demo', function (value) {
  console.log(value) // "foo bar baz"
})
```

Прочитайте Пользовательские директивы онлайн: https://riptutorial.com/ru/vue-js/topic/2368/пользовательские-директивы

глава 17: Пользовательские компоненты с v-образной моделью

Вступление

Часто нам приходится создавать некоторые компоненты, которые выполняют некоторые действия / операции с данными, и мы требуем этого в родительском компоненте. В большинстве случаев `vuex` будет лучшим решением, но в тех случаях, когда поведение дочернего компонента не имеет ничего общего с состоянием приложения, например: слайдер диапазона, выбор даты / времени, программа чтения файлов

Наличие отдельных магазинов для каждого компонента каждый раз, когда они используются, усложняется.

замечания

Чтобы иметь `v-model` на компоненте, вам нужно выполнить два условия.

1. Он должен иметь прозвище с именем 'value'
2. Он должен излучать `input` событие со значением, ожидаемым родительскими компонентами.

```
<component v-model='something'></component>
```

это просто синтаксический сахар для

```
<component
    :value="something"
    @input="something = $event.target.value"
>
</component>
```

Examples

v-модель на счетчике

Здесь `counter` - дочерний компонент, к которому обращается `demo` являющаяся родительским компонентом с использованием `v-model` .

```
// child component
Vue.component('counter', {
  template: `<div><button @click='add'>+1</button>
```

```
  <button @click='sub'>-1</button>
  <div>this is inside the child component: {{ result }}</div></div>`,
  data () {
    return {
      result: 0
    }
  },
  props: ['value'],
  methods: {
    emitResult () {
      this.$emit('input', this.result)
    },
    add () {
      this.result += 1
      this.emitResult()
    },
    sub () {
      this.result -= 1
      this.emitResult()
    }
  }
})
```

Этот дочерний компонент будет выдавать `result` каждый раз, когда вызываются методы `sub()` или `add()` .

```
// parent component
new Vue({
  el: '#demo',
  data () {
    return {
      resultFromChild: null
    }
  }
})

// parent template
<div id='demo'>
  <counter v-model='resultFromChild'></counter>
  This is in parent component {{ resultFromChild }}
</div>
```

Так как `v-model` присутствует на компоненте ребенка, опора с именем `value` был послан в то же время, является входным событием на `counter` , который , в свою очередь обеспечивают значение от компонента ребенка.

Прочитайте Пользовательские компоненты с v-образной моделью онлайн: https://riptutorial.com/ru/vue-js/topic/9353/пользовательские-компоненты-с-v-образной-моделью

глава 18: Пользовательские фильтры

Синтаксис

- `Vue.filter(name, function(value){});` // Basic
- `Vue.filter(name, function(value, begin, end){});` // Основные с обертывающими значениями
- `Vue.filter(name, function(value, input){});` // Динамические
- `Vue.filter(name, { read: function(value){}, write: function(value){} });` // Двусторонний

параметры

параметр	подробности
название	Строка - требуемое имя вызываемого имени фильтра
значение	[Обратный вызов] Любое значение данных, передаваемых в фильтр
начать	[Обратный звонок] Любое - значение, которое необходимо перед переданными данными
конец	[Обратный вызов] Любое - значение, которое следует после переданных данных
вход	[Обратный вызов] Любой пользовательский вход, связанный с экземпляром Vue для динамических результатов

Examples

Двухсторонние фильтры

С `two-way filter` мы можем назначить операцию `read` *и* `write` для одного `filter` который изменяет значение *одних и* тех *же* данных между `view` и `model` .

```
//JS
Vue.filter('uppercase', {
    //read : model -> view
    read: function(value) {
        return value.toUpperCase();
    },

    //write : view -> model
    write: function(value) {
        return value.toLowerCase();
    }
```

```
});

/*
 * Base value of data: 'example string'
 *
 * In the view : 'EXAMPLE STRING'
 * In the model : 'example string'
 */
```

основной

Пользовательские фильтры в `Vue.js` могут быть легко созданы в одном вызове функции `Vue.filter` .

```
//JS
Vue.filter('reverse', function(value) {
    return value.split('').reverse().join('');
});

//HTML
<span>{{ msg | reverse }}</span> //'This is fun!' => '!nuf si sihT'
```

Хорошая практика заключается в том, чтобы хранить все пользовательские фильтры в отдельных файлах, например, в `./filters` как тогда легко повторить использование своего кода в следующем приложении. Если вы пойдете так, вам придется **заменить часть JS** :

```
//JS
Vue.filter('reverse', require('./filters/reverse'));
```

Вы также можете определить свои собственные `begin` и `end` обертки.

```
//JS
Vue.filter('wrap', function(value, begin, end) {
    return begin + value + end;
});

//HTML
<span>{{ msg | wrap 'The' 'fox' }}</span> //'quick brown' => 'The quick brown fox'
```

Прочитайте Пользовательские фильтры онлайн: https://riptutorial.com/ru/vue-js/topic/1878/пользовательские-фильтры

глава 19: Предостережения об изменении размера массива

Вступление

Когда вы пытаетесь установить значение элемента в определенном индексе массива, инициализированного в параметре данных, vue не может обнаружить изменение и не вызывает обновление состояния. Чтобы преодолеть это предостережение, вы должны либо использовать vue vue. $ Set, либо использовать метод Array.prototype.splice

Examples

Использование Vue. $ Set

В вашем методе или любом крючке жизненного цикла, который изменяет элемент массива в определенном индексе

```
new Vue({
    el: '#app',
    data:{
        myArr : ['apple', 'orange', 'banana', 'grapes']
    },
    methods:{
        changeArrayItem: function(){
            //this will not work
            //myArr[2] = 'strawberry';

            //Vue.$set(array, index, newValue)
            this.$set(this.myArr, 2, 'strawberry');
        }
    }
})
```

Вот ссылка на скрипку

Использование Array.prototype.splice

Вы можете выполнить одно и то же изменение вместо использования `Vue.$set` с использованием `splice()` прототипа Array `splice()`

```
new Vue({
    el: '#app',
    data:{
        myArr : ['apple', 'orange', 'banana', 'grapes']
    },
    methods:{
        changeArrayItem: function(){
```

```
            //this will not work
            //myArr[2] = 'strawberry';

            //Array.splice(index, 1, newValue)
            this.myArr.splice(2, 1, 'strawberry');
        }
    }
})
```

Для вложенного массива

Если yoi имеет вложенный массив, можно сделать следующее:

```
new Vue({
    el: '#app',
    data:{
        myArr : [
            ['apple', 'banana'],
            ['grapes', 'orange']
        ]
    },
    methods:{
        changeArrayItem: function(){
            this.$set(this.myArr[1], 1, 'strawberry');
        }
    }
})
```

Вот ссылка на jsfiddle

Массив объектов, содержащих массивы

```
new Vue({
    el: '#app',
    data:{
        myArr : [
            {
                name: 'object-1',
                nestedArr: ['apple', 'banana']
            },
            {
                name: 'object-2',
                nestedArr: ['grapes', 'orange']
            }
        ]
    },
    methods:{
        changeArrayItem: function(){
            this.$set(this.myArr[1].nestedArr, 1, 'strawberry');
        }
    }
})
```

Вот ссылка на скрипку

Прочитайте Предостережения об изменении размера массива онлайн:
https://riptutorial.com/ru/vue-js/topic/10679/предостережения-об-изменении-размера-массива

глава 20: Примеси

Examples

Глобальный миксин

Вы также можете применить микс по всему миру. Соблюдайте осторожность! Как только вы примените mixin глобально, это повлияет на **каждый** экземпляр Vue, созданный впоследствии. При правильном использовании это можно использовать для ввода логики обработки для настраиваемых опций:

```
// inject a handler for `myOption` custom option
Vue.mixin({
  created: function () {
    var myOption = this.$options.myOption
    if (myOption) {
      console.log(myOption)
    }
  }
})

new Vue({
  myOption: 'hello!'
})
// -> "hello!"
```

> Используйте глобальные миксины редко и осторожно, потому что это влияет на каждый экземпляр Vue, включая сторонние компоненты. В большинстве случаев вы должны использовать его только для пользовательской обработки параметров, как показано в примере выше.

Стратегии слияния пользовательских опций

Когда пользовательские параметры объединяются, они используют стратегию по умолчанию, которая просто перезаписывает существующее значение. Если вы хотите, чтобы пользовательская опция была объединена с использованием пользовательской логики, вам необходимо прикрепить функцию к `Vue.config.optionMergeStrategies` :

```
Vue.config.optionMergeStrategies.myOption = function (toVal, fromVal) {
  // return mergedVal
}
```

Для большинства объектных параметров вы можете просто использовать ту же стратегию, которая используется `methods` :

```
var strategies = Vue.config.optionMergeStrategies
strategies.myOption = strategies.methods
```

основы

Mixins - это гибкий способ распространения возможностей повторного использования компонентов Vue. Объект mixin может содержать любые параметры компонента. Когда компонент использует mixin, все параметры в mixin будут «смешаны» с собственными параметрами компонента.

```
// define a mixin object
var myMixin = {
  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// define a component that uses this mixin
var Component = Vue.extend({
  mixins: [myMixin]
})

var component = new Component() // -> "hello from mixin!"
```

Вариант слияния

Когда mixin и сам компонент содержат перекрывающиеся опции, они будут «объединены» с использованием соответствующих стратегий. Например, функции перехвата с тем же именем объединяются в массив, чтобы все они вызывались. Кроме того, микшерные крючки будут вызываться **перед** собственными крючками компонента:

```
var mixin = {
  created: function () {
    console.log('mixin hook called')
  }
}

new Vue({
  mixins: [mixin],
  created: function () {
    console.log('component hook called')
  }
})

// -> "mixin hook called"
// -> "component hook called"
```

Параметры, которые ожидают значения объектов, например `methods` , `components` и `directives` , будут объединены в один и тот же объект. Параметры компонента будут иметь приоритет, если в этих объектах имеются конфликтующие ключи:

```
var mixin = {
  methods: {
    foo: function () {
      console.log('foo')
    },
    conflicting: function () {
      console.log('from mixin')
    }
  }
}

var vm = new Vue({
  mixins: [mixin],
  methods: {
    bar: function () {
      console.log('bar')
    },
    conflicting: function () {
      console.log('from self')
    }
  }
})

vm.foo() // -> "foo"
vm.bar() // -> "bar"
vm.conflicting() // -> "from self"
```

Обратите внимание, что те же стратегии слияния используются в `Vue.extend()` .

Прочитайте Примеси онлайн: https://riptutorial.com/ru/vue-js/topic/2562/примеси

глава 21: Реквизит

замечания

camelCase <=> кебаб-чехол

При определении имен ваших `props` всегда помните, что имена атрибутов HTML не зависят от регистра. Это означает, что если вы определяете `prop` в случае верблюда в определении вашего компонента ...

```
Vue.component('child', {
    props: ['myProp'],
    ...
});
```

... вы должны назвать это в своем HTML-компоненте как my-prop.

Examples

Передача данных от родителя к ребенку с помощью реквизита

В Vue.js каждый экземпляр компонента имеет ***свою изолированную область видимости*** , что означает, что если родительский компонент имеет дочерний компонент, дочерний компонент имеет свою изолированную область, а родительский компонент имеет свою изолированную область.

Для любого приложения от среднего до большого размера в соответствии с рекомендациями по наилучшей практике предотвращается много головных болей на этапе разработки, а затем после технического обслуживания. Одна из таких вещей заключается в том, чтобы ***избежать ссылок на / исходные данные родительских данных непосредственно из дочернего компонента*** . Итак, как мы ссылаемся на родительские данные из дочернего компонента?

Независимо от того, какие родительские данные необходимы в дочернем компоненте, он должен быть передан ребенку в качестве `props` родителя.

Пример использования . Предположим, у нас есть пользовательская база данных с двумя `users` и `addresses` таблиц со следующими полями:
таблица `users`

название	Телефон	Эл. адрес
Джон Мклане	(1) 234 5678 9012	john@dirhard.com

название	Телефон	Эл. адрес
Джеймс Бонд	(44) 777 0007 0077	bond@mi6.com

`addresses` таблица

блок	улица	город
Башни Накатоми	Бродвей	Нью-Йорк
Mi6 House	Buckingham Road	Лондон

и мы хотим иметь три компонента для отображения соответствующей пользовательской информации в любом месте нашего приложения

пользовательский component.js

```
export default{
    template:`<div class="user-component">
                <label for="name" class="form-control">Name: </label>
                <input class="form-control input-sm" name="name" v-model="name">
                <contact-details :phone="phone" :email="email"></contact-details>
              </div>`,
    data(){
        return{
            name:'',
            phone:'',
            email:''
        }
    },
}
```

контакт-details.js

```
import Address from './address';
export default{
    template:`<div class="contact-details-component>
                <h4>Contact Details:</h4>
                <label for="phone" class="form-control">Phone: </label>
                <input class="form-control input-sm" name="phone" v-model="phone">
                <label for="email" class="form-control">Email: </label>
                <input class="form-control input-sm" name="email" v-model="email">

                <h4>Address:</h4>
                <address :address-type="addressType"></address>
                //see camelCase vs kebab-case explanation below
            </div>`,
    props:['phone', 'email'],
    data:(){
        return:{
            addressType:'Office'
        }
    },
    components:{Address}
```

```
}
```

address.js

```
export default{
    template:`<div class="address-component">
                <h6>{{addressType}}</h6>
                <label for="block" class="form-control">Block: </label>
                <input class="form-control input-sm" name="block" v-model="block">
                <label for="street" class="form-control">Street: </label>
                <input class="form-control input-sm" name="street" v-model="street">
                <label for="city" class="form-control">City: </label>
                <input class="form-control input-sm" name="city" v-model="city">
              </div>`,
    props:{
        addressType:{
            required:true,
            type:String,
            default:'Office'
        },
    data(){
        return{
            block:'',
            street:'',
            city:''
        }
    }
}
```

main.js

```
import Vue from 'vue';

Vue.component('user-component', require'./user-component');
Vue.component('contact-details', require'./contact-details');

new Vue({
    el:'body'
});
```

index.html

```
...
<body>
    <user-component></user-component>
        ...
</body>
```

Мы показываем данные `phone` и `email` , которые являются свойствами `user-component` в `contact-details` которые не содержат телефонных или электронных данных.

Передача данных в виде реквизита

Таким образом, внутри `user-component.js` в свойстве ***шаблона*** , где мы включаем компонент `<contact-details>` , мы передаем ***телефон*** и данные ***электронной почты*** из `<user-component>` (

родительский компонент) в `<contact-details>` (дочерний компонент), динамически привязывая его к **реквизитам** - `:phone="phone"` и `:email="email` который аналогичен `v-bind:phone="phone"` и `v-bind:email="email"`

Опоры - Динамическое связывание

Поскольку мы динамически связываем реквизиты с любыми изменениями в ***телефоне*** или ***электронной почте*** внутри родительского компонента, то есть `<user-component>` будет немедленно отражен в дочернем компоненте, т.е. `<contact-details>` .

Реквизит - как литералы

Однако, если бы мы передали значения ***телефона*** и ***электронной почты в*** виде строковых литералов, таких как `phone="(44) 777 0007 0077" email="bond@mi6.com"` то это не отразило бы какие-либо изменения данных, которые происходят в родительском составная часть.

Односторонняя привязка

По умолчанию направление изменений сверху вниз, т.е. любое изменение динамически привязанных реквизитов в родительском компоненте будет распространяться на дочерний компонент, но любое изменение значений prop в дочернем компоненте не будет распространяться на родителя.

Например, если из `<contact-details>` мы меняем электронное письмо с `bond@mi6.com` на `jamesbond@mi6.com` , родительские данные, т. `bond@mi6.com jamesbond@mi6.com` данных телефона в `<user-component>` все равно будут содержать значение `bond@mi6.com` .

Однако, если мы изменим значение электронной почты от `bond@mi6.com` на `jamesbond@mi6.co` в родительском компоненте (`<user-component>` в нашем случае использования), то значение электронной почты в дочернем компоненте (`<contact-details>` в нашем случае использования) автоматически изменится на `jamesbond@mi6.com` - изменение родительского `jamesbond@mi6.com` мгновенно распространяется на ребенка.

Двусторонняя привязка

Если мы хотим двустороннюю привязку, мы должны явно указать двустороннюю привязку как `:email.sync="email"` вместо `:email="email"` . Теперь, если мы изменим значение prop в дочернем компоненте, изменение будет отражено и в родительском компоненте.

В средстве и крупном приложении изменение состояния родителя из дочернего состояния будет очень сложно обнаружить и отслеживать особенно во время отладки - **будьте осторожны** .

В Vue.js 2.0 не будет никакой опции .sync. **Двусторонняя привязка для реквизита устарела в Vue.js 2.0** .

Одноразовая привязка

Также можно определить явное **одноразовое** привязку как `:email.once="email` , оно более или менее похоже на передачу литерала, потому что любые последующие изменения в значении родительского свойства не будут распространяться на ребенка.

ПРЕДОСТЕРЕЖЕНИЕ
Когда **Object или Array** передаются как prop, они **ВСЕГДА ПРОХОДА ПО ССЫЛКЕ** , что означает независимо от типа привязки, явно определенного `:email.sync="email"` или `:email="email"` или `:email.once="email"` , если адрес электронной почты является объектом или массивом в родительском объекте, то независимо от типа привязки любое изменение значения prop в дочернем компоненте также влияет на значение в родительском.

Реквизит в виде массива

В файле `contact-details.js` мы определили `props:['phone', 'email']` в качестве массива, что прекрасно, если мы не хотим, чтобы мелкозернистый элемент управления с реквизитами.

Реквизит как объект

Если мы хотим более мелкозернистого контроля над реквизитами, например

- если мы хотим определить, какие типы значений приемлемы в качестве опоры
- что должно быть значением по умолчанию для опоры
- является ли значение ДОЛЖНЫ (требуется) для передачи пропеллера или оно необязательно

то нам нужно использовать объектную нотацию для определения реквизита, как это было сделано в `address.js` .

Если мы создаем повторно используемые компоненты, которые могут быть использованы другими разработчиками в команде, то хорошей практикой является определение реквизита в качестве объектов, чтобы кто-либо, использующий компонент, имел четкое представление о том, каким должен быть тип данных, и он является обязательным или необязательным.

Он также называется ***проверкой реквизита*** . **Тип** может быть любым из следующих встроенных конструкторов:

- строка
- Число
- логический
- массив
- объект
- функция
- или пользовательский конструктор

Некоторые примеры проверки профайла, взятые из http://vuejs.org/guide/components.html#Props

```
Vue.component('example', {
   props: {
       // basic type check (`null` means accept any type)
       propA: Number,
       // multiple possible types (1.0.21+)
       propM: [String, Number],
       // a required string
       propB: {
         type: String,
         required: true
       },
       // a number with default value
       propC: {
          type: Number,
          default: 100
       },
       // object/array defaults should be returned from a
       // factory function
       propD: {
          type: Object,
          default: function () {
             return { msg: 'hello' }
         }
       },
       // indicate this prop expects a two-way binding. will
       // raise a warning if binding type does not match.
       propE: {
          twoWay: true
       },
       // custom validator function
       propF: {
          validator: function (value) {
             return value > 10
          }
       },
       // coerce function (new in 1.0.12)
       // cast the value before setting it on the component
       propG: {
          coerce: function (val) {
            return val + '' // cast the value to string
          }
       },
       propH: {
          coerce: function (val) {
            return JSON.parse(val) // cast the value to Object
          }
       }
    }
});
```

camelCase против кебаба

HTML-атрибуты нечувствительны к регистру, что означает, что он не может различать `addresstype` и `addressType` , поэтому при использовании имен propel camelCase в качестве атрибутов нам необходимо использовать их эквиваленты кебаба (дефисграничные):

`addressType` должен быть записан как `address-type` в атрибуте HTML.

Динамические реквизиты

Так же, как вы можете привязывать данные к представлению к модели, вы также можете связывать реквизиты с помощью той же директивы v-bind для передачи информации от родительского к дочерним компонентам.

JS

```
new Vue({
    el: '#example',
    data: {
        msg: 'hello world'
    }
});

Vue.component('child', {
    props: ['myMessage'],
    template: '<span>{{ myMessage }}</span>
});
```

HTML

```
<div id="example">
    <input v-model="msg" />
    <child v-bind:my-message="msg"></child>
    <!-- Shorthand ... <child :my-message="msg"></child> -->
</div>
```

Результат

```
hello world
```

Передача реквизитов при использовании Vue JSX

У нас есть родительский компонент: импортируя в него дочерний компонент, мы передадим реквизит через атрибут. Здесь атрибут «src», и мы также передаем «src».

ParentComponent.js

```
import ChildComponent from './ChildComponent';
export default {
    render(h, {props}) {
        const src = 'https://cdn-images-1.medium.com/max/800/1*AxRXW2j8qmGJixIYg7n6uw.jpeg';
        return (
```

```
            <ChildComponent src={src} />
        );
    }
};
```

И дочерний компонент, где нам нужно пройти реквизит. Нам нужно указать, какие реквизиты мы проходим.

ChildComponent.js:

```
export default {
    props: ['src'],
    render(h, {props}) {
        return (
            <a href = {props.src} download = "myimage" >
                Click this link
            </a>
        );
    }
};
```

Прочитайте Реквизит онлайн: https://riptutorial.com/ru/vue-js/topic/3080/реквизит

глава 22: Связывание данных

Examples

Текст

Наиболее простой формой привязки данных является интерполяция текста с использованием синтаксиса «Усы» (двойные фигурные скобки):

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

Тег усы будет заменен значением свойства `msg` на соответствующем объекте данных. Он также будет обновляться всякий раз, когда изменяется свойство `msg` объекта объекта.

Вы также можете выполнять одноразовые интерполяции, которые не обновляются при изменении данных:

```
<span>This will never change: {{* msg }}</span>
```

Сырой HTML

Двойные усы интерпретируют данные как обычный текст, а не HTML. Для вывода реального HTML вам нужно будет использовать тройные усы:

```
<div>{{{ raw_html }}}</div>
```

Содержимое вставляется в виде простого HTML - привязки данных игнорируются. Если вам нужно повторно использовать шаблоны, вы должны использовать частичные.

Атрибуты

Усы также можно использовать внутри атрибутов HTML:

```
<div id="item-{{ id }}"></div>
```

Обратите внимание, что интерполяции атрибутов запрещены в директивах Vue.js и специальных атрибутах. Не беспокойтесь, Vue.js поднимет вам предупреждения, когда усы будут использоваться в неправильных местах.

фильтры

Vue.js позволяет добавлять дополнительные «фильтры» в конец выражения, обозначаемого символом «pipe»:

```
{{ message | capitalize }}
```

Здесь мы «труба» значение `message` выражения через встроенные `capitalize` фильтр, который на самом деле просто функция JavaScript , которая возвращает значение капитализированного. Vue.js предоставляет ряд встроенных фильтров, и мы поговорим о том, как писать собственные фильтры позже.

Обратите внимание, что синтаксис канала не является частью синтаксиса JavaScript, поэтому вы не можете смешивать фильтры внутри выражений; вы можете добавлять их только в конце выражения.

Фильтры могут быть связаны:

```
{{ message | filterA | filterB }}
```

Фильтры также могут принимать аргументы:

```
{{ message | filterA 'arg1' arg2 }}
```

Функция фильтра всегда принимает значение выражения в качестве первого аргумента. Котируемые аргументы интерпретируются как простая строка, а не цитируемые - будут вычисляться как выражения. Здесь простая строка `'arg1'` будет передана в фильтр в качестве второго аргумента, а значение выражения `arg2` будет оценено и передано в качестве третьего аргумента.

Прочитайте Связывание данных онлайн: https://riptutorial.com/ru/vue-js/topic/1213/связывание-данных

глава 23: События

Examples

Синтаксис событий

Чтобы отправить событие: `vm.$emit('new-message');`

Чтобы поймать событие: `vm.$on('new-message');`

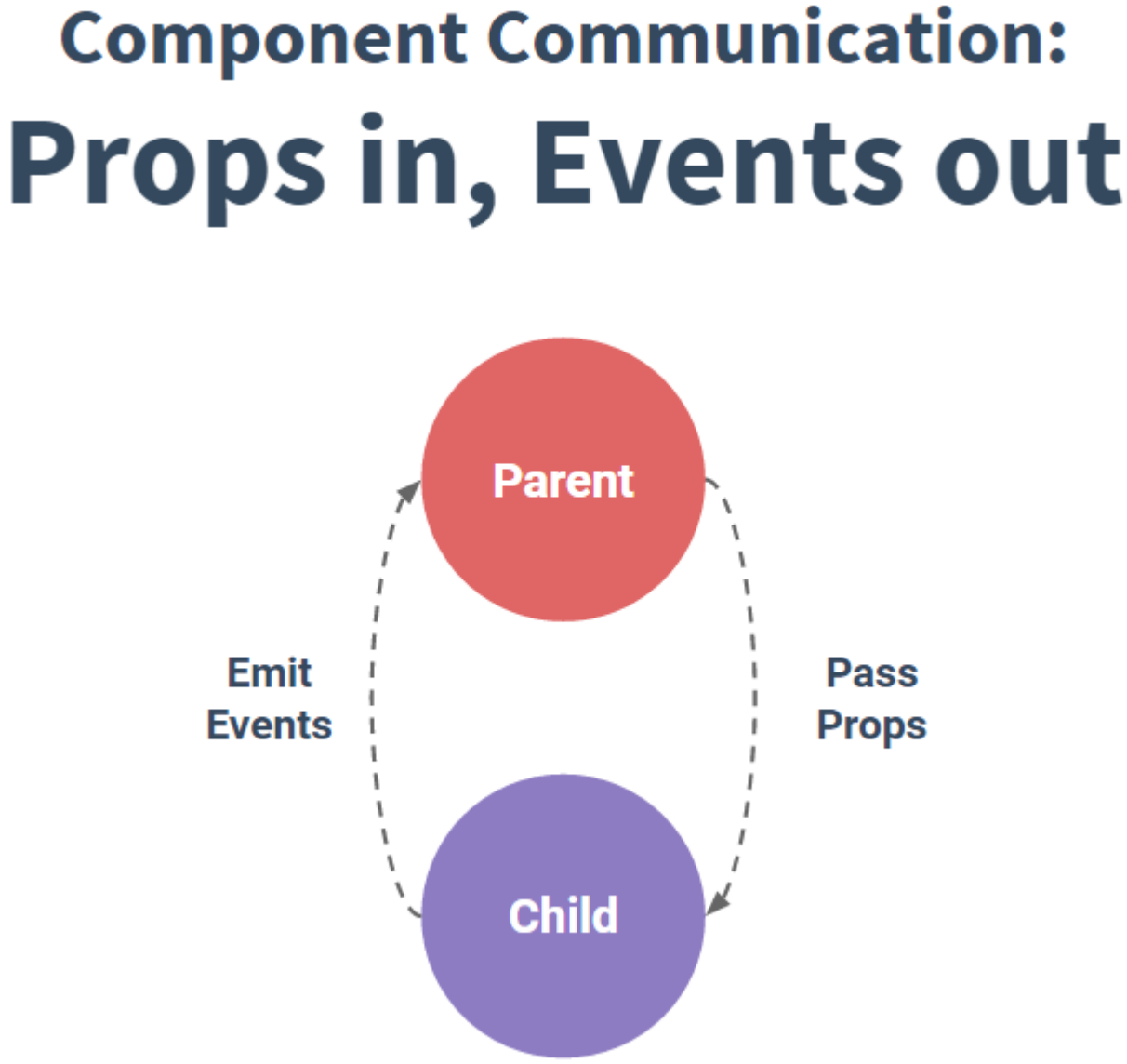
Для того, чтобы отправить событие для всех компонентов **вниз:** `vm.$broadcast('new-message'); . vm.$broadcast('new-message');`

Чтобы отправить событие всем компонентам **вверх** : `vm.$dispatch('new-message');`

Примечание: `$broadcast` и `$dispatch` устарели в Vue2. (см. функции Vue2)

Когда следует использовать события?

На следующем рисунке показано, как взаимодействие компонентов должно работать. Изображение происходит от слайдов Progressive Framework от Evan You (разработчика VueJS).



Вот пример того, как это работает:

DEMO

HTML

```
<script type="x-template" id="message-box">
    <input type="text" v-model="msg" @keyup="$emit('new-message', msg)" />
</script>

<message-box :msg="message" @new-message="updateMessage"></message-box>
<div>You typed: {{message}}</div>
```

JS

```
var messageBox = {
  template: '#message-box',
  props: ['msg']
};

new Vue({
  el: 'body',
```

```
  data: {
    message: ''
  },
  methods: {
    updateMessage: function(msg) {
      this.message = msg;
    }
  },
  components: {
    'message-box': messageBox
  }
});
```

Приведенный выше пример можно улучшить!

В приведенном выше примере показано, как работает компонентная связь. Но в случае пользовательского компонента ввода, чтобы синхронизировать родительскую переменную с введенным значением, мы можем использовать `v-model` .

DEMO Vue1

DEMO Vue2

В Vue1 вы должны использовать `.sync` на опоре, отправленном в компонент `<message-box>` . Это указывает VueJS синхронизировать значение в дочернем компоненте с родительским.

> **Помните:** каждый экземпляр компонента имеет свою изолированную область.

HTML Vue1

```
<script type="x-template" id="message-box">
  <input v-model="value" />
</script>

<div id="app">
  <message-box :value.sync="message"></message-box>
  <div>You typed: {{message}}</div>
</div>
```

В Vue2 есть специальное событие `'input'` которое вы можете `$emit` . Использование этого события позволяет поместить `v-model` непосредственно в компонент `<message-box>` . Пример будет выглядеть следующим образом:

HTML Vue2

```
<script type="x-template" id="message-box">
  <input :value="value" @input="$emit('input', $event.target.value)" />
</script>

<div id="app">
  <message-box v-model="message"></message-box>
```

```
  <div>You typed: {{message}}</div>
</div>
```

JS Vue 1 & 2

```
var messageBox = {
  template: '#message-box',
  props: ['value']
};

new Vue({
  el: '#app',
  data: {
    message: ''
  },
  components: {
    'message-box': messageBox
  }
});
```

Обратите внимание, как быстрее обновляется вход.

Как бороться с устареванием $ dispatch и $ broadcast? (шаблон события шины)

Возможно, вы поняли, что `$emit` привязан к компоненту, который испускает событие. Это проблема, когда вы хотите общаться между компонентами, расположенными далеко друг от друга в дереве компонентов.

Примечание. В Vue1 вы можете использовать `$dispatch` или `$broadcast` , но не в Vue2. Причина в том, что он плохо масштабируется. Существует популярный шаблон `bus` для управления этим:

DEMO

HTML

```
<script type="x-template" id="sender">
  <button @click="bus.$emit('new-event')">Click me to send an event !</button>
</script>

<script type="x-template" id="receiver">
  <div>I received {{numberOfEvents}} event{{numberOfEvents == 1 ? '' : 's'}}</div>
</script>

<sender></sender>
<receiver></receiver>
```

JS

```
var bus = new Vue();
```

```
var senderComponent = {
  template: '#sender',
  data() {
    return {
      bus: bus
    }
  }
};

var receiverComponent = {
  template: '#receiver',
  data() {
    return {
      numberOfEvents: 0
    }
  },
  ready() {
    var self = this;

    bus.$on('new-event', function() {
      ++self.numberOfEvents;
    });
  }
};

new Vue({
  el: 'body',
  components: {
    'sender': senderComponent,
    'receiver': receiverComponent
  }
});
```

Вам просто нужно понять, что любой экземпляр `Vue()` может `$emit` и catch (`$on`) событие. Мы просто объявляем глобальную `bus` вызовов экземпляра `Vue` а затем любой компонент с этой переменной может испускать и извлекать из нее события. Просто убедитесь, что компонент имеет доступ к переменной `bus` .

Прочитайте События онлайн: https://riptutorial.com/ru/vue-js/topic/5941/события

глава 24: Список рендеринга

Examples

Основное использование

Список может отображаться с помощью директивы `v-for` . Синтаксис требует, чтобы вы указывали исходный массив для итерации и псевдоним, который будет использоваться для ссылки на каждый элемент в итерации. В следующем примере мы используем `items` как исходный массив, а `item` как псевдоним для каждого элемента.

HTML

```
<div id="app">
  <h1>My List</h1>
  <table>
    <tr v-for="item in items">
      <td>{{item}}</td>
    </tr>
  </table>
</div>
```

скрипт

```
new Vue({
  el: '#app',
  data: {
    items: ['item 1', 'item 2', 'item 3']
  }
})
```

Вы можете просмотреть рабочую демку здесь .

Отображать только элементы HTML

В этом примере будет отображаться пять тегов `<li>`

```
<ul id="render-sample">
  <li v-for="n in 5">
    Hello Loop
  </li>
</ul>
```

Список обратного отсчета свиней

```
<ul>
  <li v-for="n in 10">{{11 - n}} pigs are tanning at the beach. One got fried, and
</ul>
```

https://jsfiddle.net/gurghet/3jeyka22/

Итерация по объекту

`v-for` может использоваться для итерации над объектными ключами (и значениями):

HTML:

```
<div v-for="(value, key) in object">
  {{ key }} : {{ value }}
</div>
```

Автор *сценария:*

```
new Vue({
  el: '#repeat-object',
  data: {
    object: {
      FirstName: 'John',
      LastName: 'Doe',
      Age: 30
    }
  }
})
```

Прочитайте Список рендеринга онлайн: https://riptutorial.com/ru/vue-js/topic/1972/список-рендеринга

глава 25: Условный рендеринг

Синтаксис

- `<element v-if="condition"></element>` // v-if
- `<element v-if="condition"></element><element v-else="condition"></element>` // v-if | v-то еще
- `<template v-if="condition">...</template>` // templated v-if
- `<element v-show="condition"></element>` // v-show

замечания

Очень важно помнить разницу между `v-if` и `v-show` . Хотя их использование почти идентично, элемент, связанный с `v-if` *будет отображаться только в DOM,* когда это условие является `true` в **первый раз** . При использовании `v-show` директивы, *все элементы отображаются в DOM* , но скрыты с помощью `display` стиля , если условие `false` !

Examples

обзор

В Vue.js условный рендеринг достигается с помощью набора директив по элементам в шаблоне.

`v-if`

Элемент отображается нормально, когда условие `true` . Когда условие `false` , происходит только *частичная* компиляция, и элемент не отображается в DOM, пока условие не станет `true` .

`v-else`

Не принимает условие, а отображает элемент, `v-if` условие `v-if` предыдущего элемента `false` . Может использоваться только после элемента с директивой `v-if` .

`v-show`

Поведение аналогично `v-if` , однако, элемент *всегда* будет отображаться в DOM, даже если условие `false` . Если условие `false` , эта директива просто установит стиль `display` элемента на `none` .

v-if / v-else

Предполагая, что у нас есть экземпляр `Vue.js` определенный как:

```
var vm = new Vue({
    el: '#example',
    data: {
        a: true,
        b: false
    }
});
```

Вы можете условно отобразить любой элемент html, включив в него директиву v-if; элемент, содержащий v-if, будет отображаться только если условие имеет значение true:

```
<!-- will render 'The condition is true' into the DOM -->
<div id="example">
    <h1 v-if="a">The condition is true</h1>
</div>
```

Элемент `<h1>` будет отображать в этом случае, потому что переменная 'a' истинна. v-if может использоваться с любым выражением, вычисленным свойством или функцией, которая вычисляет логическое значение:

```
<div v-if="0 === 1">                  false; won't render</div>
<div v-if="typeof(5) === 'number'">   true; will render</div>
```

Вы можете использовать элемент `template` для группировки нескольких элементов для одного условия:

```
<!-- in this case, nothing will be rendered except for the containing 'div' -->
<div id="example">
    <template v-if="b">
        <h1>Heading</h1>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </template>
</div>
```

При использовании `v-if` вас также есть возможность интегрировать условие счетчика с директивой `v-else` . Содержимое, содержащееся внутри элемента, будет отображаться только в том случае, если условие предыдущего v-if было ложным. Обратите внимание, что это означает, что элемент с v-else должен появляться сразу после элемента с v-if.

```
<!-- will render only 'ELSE' -->
<div id="example">
    <h1 v-if="b">IF</h1>
    <h1 v-else="a">ELSE</h1>
</div>
```

Как и с v-if, с помощью v-else вы можете группировать несколько элементов html вместе в `<template>` :

```
<div v-if="'a' === 'b'"> This will never be rendered. </div>
<template v-else>
```

```
    <ul>
      <li> You can also use templates with v-else. </li>
      <li> All of the content within the template </li>
      <li> will be rendered. </li>
    </ul>
</template>
```

v-шоу

Использование директивы `v-show` почти идентично использованию `v-if` . Единственное отличие состоит в том, что `v-show` *не* поддерживает синтаксис `<template>` , и нет «альтернативного» условия.

```
var vm = new Vue({
    el: '#example',
    data: {
        a: true
    }
});
```

Основное использование заключается в следующем ...

```
<!-- will render 'Condition met' -->
<div id="example">
    <h1 v-show="a">Condition met</h1>
</div>
```

Хотя `v-show` не поддерживает директиву `v-else` для определения «альтернативных» условий, это может быть достигнуто путем отрицания предыдущего ...

```
<!-- will render 'This is shown' -->
<div id="example">
    <h1 v-show="!a">This is hidden</h1>
    <h1 v-show="a">This is shown</h1>
</div>
```

Прочитайте Условный рендеринг онлайн: https://riptutorial.com/ru/vue-js/topic/3465/условный-рендеринг

глава 26: Шаблон "webpack" для Polyfill

параметры

Файлы или пакеты	Команда или конфигурация для изменения
столпотворение-polyfill	`npm i -save babel-polyfill`
karma.conf.js	`files: ['../../node_modules/babel-polyfill/dist/polyfill.js','./index.js'],`
webpack.base.conf.js	`app: ['babel-polyfill', './src/main.js']`

замечания

Конфигурации, описанные выше, пример, использующий нестандартную функцию, будет работать в «Internet explorer», и `npm test` пройдет.

Examples

Использование функций для полифонирования (например: поиск)

```
<template>
  <div class="hello">
    <p>{{ filtered() }}</p>
  </div>
</template>

<script>
export default {
  name: 'hello',
  data () {
    return {
      list: ['toto', 'titi', 'tata', 'tete']
    }
  },
  methods: {
    filtered () {
      return this.list.find((el) => el === 'tata')
    }
  }
}
</script>
```

Прочитайте Шаблон "webpack" для Polyfill онлайн: https://riptutorial.com/ru/vue-js/topic/9174/шаблон--webpack--для-polyfill

кредиты

S. No	Главы	Contributors
1	Начало работы с Vue.js	Community, Erick Petrucelli, ironcladgeek, J. Bruni, James, Lambda Ninja, m_callens, MotKohn, rap-2-h, Ru Chern Chong, Sankalp Singha, Shog9, Shuvo Habib, user1012181, user6939352, Yerko Palma
2	VueJS + Redux с Vua-Redux (лучшее решение)	Aniko Litvanyi, FlatLander, Shuvo Habib, Stefano Nepa
3	Vuex	AldoRomo88, Amresh Venugopal, Daniel Waghorn, Matej Vrzala M4, Ru Chern Chong
4	Автобус событий	Amresh Venugopal
5	Вычислимые свойства	Amresh Venugopal, cl3m, jaredsk, m_callens, Theo, Yerko Palma
6	вя-маршрутизатор	AJ Gregory
7	Динамические компоненты	Med, Ru Chern Chong
8	Зрителей	El_Matella
9	игровые автоматы	Daniel Waghorn, Elfayer, Shuvo Habib, Slava
10	Использование « this» в Vue	Bert
11	Компоненты	Donkarnash, Elfayer, Hector Lorenzo, Jeff, m_callens, phaberest, RedRiderX, user6939352
12	Компоненты VUE для одного файла	jordiburgos, Ru Chern Chong
13	Крючки жизненного цикла	Linus Borg, m_callens, PatrickSteele, xtreak
14	Модификаторы	sept08
15	Плагины	AldoRomo88

16	Пользовательские директивы	Mat J, Ogie Sado
17	Пользовательские компоненты с v-образной моделью	Amresh Venugopal
18	Пользовательские фильтры	Finrod, M U, m_callens
19	Предостережения об изменении размера массива	Vamsi Krishna
20	Примеси	Ogie Sado
21	Реквизит	asemahle, Donkarnash, FlatLander, m_callens, rap-2-h, Shuvo Habib
22	Связывание данных	gurghet, Jilson Thomas
23	События	Elfayer
24	Список рендеринга	chuanxd, gurghet, Mahmoud, Theo
25	Условный рендеринг	jaredsk, m_callens, Nirazul, user6939352
26	Шаблон "webpack" для Polyfill	Stefano Nepa