Kostenloses eBook

LERNEN

Vue.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#vue.js

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit Vue.js

Bemerkungen

`Vue.js` ist ein schnell wachsendes Front-End-Framework für `JavaScript` , das von `Angular.js` , `Reactive.js` und `Rivets.js` inspiriert wurde und ein `Rivets.js` Design der Benutzeroberfläche, Manipulation und tiefe Reaktivität bietet.

Es wird als `MVVM` Muster-Framework, `Model-View View-Model` , das auf dem Konzept der *bidirektionalen Bindungsdaten* an Komponenten und Ansichten basiert. Es ist unglaublich schnell, übertrifft die Geschwindigkeiten anderer Top-Tier- `JS` Frameworks und ist sehr benutzerfreundlich für einfache Integration und Prototyping.

Versionen

Ausführung	Veröffentlichungsdatum
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

"Hallo Welt!" Programm

Um Vue.js zu verwenden , stellen Sie sicher, dass Sie die Skriptdatei in Ihrem HTML- Code enthalten. Fügen Sie Ihrem HTML-Code beispielsweise Folgendes hinzu.

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

Einfaches Beispiel

HTML-Vorlage

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

JavaScript

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

Sehen Sie sich eine Live-Demo dieses Beispiels an.

Vielleicht möchten Sie auch das von Vue.js erstellte Beispiel "Hello World" überprüfen.

Hallo Welt in Vue 2 (Der JSX-Weg)

JSX soll vom Browser nicht interpretiert werden. Es muss zuerst in Standard-Javascript übersetzt werden. Um JSX verwenden zu können, müssen Sie das Plugin für babel `babel-plugin-transform-vue-JSX` installieren

Führen Sie den folgenden Befehl aus:

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

und fügen Sie es wie `.babelrc` zu Ihrem `.babelrc` :

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

Beispielcode mit 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>
    )
  }
})
```

Mit JSX können Sie kurze HTML / XML-ähnliche Strukturen in dieselbe Datei schreiben wie JavaScript-Code.

Glückwunsch, du bist fertig :)

Benutzereingaben behandeln

Mit VueJS können auch Benutzereingaben problemlos verarbeitet werden. Durch die bidirektionale Bindung mit dem V-Modell können Daten ganz einfach geändert werden.

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!'
  }
})
```

Es ist sehr einfach, eine bidirektionale Bindung in VueJS mithilfe der `v-model` Direktive durchzuführen.

Sehen Sie sich hier ein Live-Beispiel an.

Erste Schritte mit Vue.js online lesen: https://riptutorial.com/de/vue-js/topic/1057/erste-schritte-mit-vue-js

Kapitel 2: Bedingtes Rendern

Syntax

- `<element v-if="condition"></element>` // v-if
- `<element v-if="condition"></element><element v-else="condition"></element>` // V-if | v-sonst
- `<template v-if="condition">...</template>` // Vorlage für v-if
- `<element v-show="condition"></element>` // V-Show

Bemerkungen

Es ist sehr wichtig, sich an den Unterschied zwischen `v-if` und `v-show` zu erinnern. Während ihre Verwendung fast identisch ist, ein Element gebunden `v-if` *nur in das DOM machen wird* , wenn es Bedingung ist `true` zum **ersten Mal.** Bei Verwendung der `v-show` - Richtlinie, ***werden*** *alle Elemente in das DOM gemacht* , sondern mit dem versteckten `display` Stil , wenn die Bedingung ist `false` !

Examples

Überblick

In Vue.js wird das bedingte Rendern durch Verwendung einer Reihe von Anweisungen für Elemente in der Vorlage erreicht.

`v-if`

Element zeigt normalerweise , wenn die Bedingung ist `true` . Wenn die Bedingung `false` , wird nur eine *teilweise* Kompilierung durchgeführt und das Element wird nicht in das DOM gerendert, bis die Bedingung `true` .

`v-else`

Akzeptiert keine Bedingung, sondern rendert das Element, wenn die `v-if` Bedingung des vorherigen Elements `false` . Kann nur nach einem Element mit der `v-if` Direktive verwendet werden.

`v-show`

Verhält sich ähnlich wie `v-if` das Element jedoch *immer* in das DOM gerendert wird, auch wenn die Bedingung `false` . Wenn die Bedingung `false` , setzt diese Direktive einfach den `display` des Elements auf `none` .

v-if / v-else

Angenommen, wir haben eine `Vue.js` Instanz definiert als:

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

Sie können jedes HTML-Element bedingt darstellen, indem Sie die V-if-Direktive einfügen. Das Element, das v-if enthält, wird nur gerendert, wenn die Bedingung als wahr ausgewertet wird:

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

Das Element `<h1>` wird in diesem Fall gerendert, da die Variable 'a' wahr ist. v-if kann mit einem beliebigen Ausdruck, einer berechneten Eigenschaft oder einer Funktion verwendet werden, die einen booleschen Wert ergibt:

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

Sie können ein `template` verwenden, um mehrere Elemente für eine einzelne Bedingung zusammenzufassen:

```
<!-- 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>
```

Bei Verwendung von `v-if` haben Sie auch die Möglichkeit, eine Zählerbedingung in die `v-else` Direktive zu integrieren. Der Inhalt des Elements wird nur angezeigt, wenn die Bedingung des vorherigen v-if falsch war. Beachten Sie, dass dies bedeutet, dass ein Element mit v-else unmittelbar nach einem Element mit v-if erscheinen muss.

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

Genau wie bei v-if können Sie mit v-else mehrere HTML-Elemente innerhalb einer `<template>` gruppieren:

```
<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-show

Die Verwendung der `v-show` Direktive ist fast identisch mit der von `v-if` . Der einzige Unterschied besteht darin, dass `v-show` die `<template>` -Syntax *nicht* unterstützt und es keine "alternative" Bedingung gibt.

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

Die grundlegende Verwendung ist wie folgt ...

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

Während `v-show` die `v-else` Direktive zur Definition von "alternativen" Bedingungen nicht unterstützt, kann dies durch Negieren der vorherigen Bedingung erreicht werden.

```
<!-- 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>
```

Bedingtes Rendern online lesen: https://riptutorial.com/de/vue-js/topic/3465/bedingtes-rendern

Kapitel 3: Benutzerdefinierte Filter

Syntax

- `Vue.filter(name, function(value){});` //Basic
- `Vue.filter(name, function(value, begin, end){});` // Basic mit Umbruchwerten
- `Vue.filter(name, function(value, input){});` // Dynamisch
- `Vue.filter(name, { read: function(value){}, write: function(value){} });` // Zweiwege

Parameter

Parameter	Einzelheiten
Name	String - gewünschter aufrufbarer Name des Filters
Wert	[Callback] Any - Wert der Daten, die in den Filter gelangen
Start	[Rückruf] Beliebiger Wert, der vor den übergebenen Daten liegt
Ende	[Rückruf] Beliebiger Wert, der nach den übergebenen Daten kommt
Eingang	[Rückruf] Alle - Benutzereingaben, die für dynamische Ergebnisse an die Vue-Instanz gebunden sind

Examples

Zweiwege-Filter

Mit einem `two-way filter` können wir einem einzelnen `filter` einen `read` *und* `write` zuweisen, der den Wert *derselben* Daten zwischen `view` und `model` ändert.

```
//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'
 */
```

Basic

Benutzerdefinierte Filter in `Vue.js` können einfach in einem einzigen Funktionsaufruf von `Vue.filter` .

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

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

Es ist `./filters` , alle benutzerdefinierten Filter in separaten Dateien zu speichern, z. Wenn Sie diesen Weg gehen, müssen Sie **das JS-Teil ersetzen** :

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

Sie können auch Ihre eigenen `begin` und `end` Wrapper definieren.

```
//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'
```

Benutzerdefinierte Filter online lesen: https://riptutorial.com/de/vue-js/topic/1878/benutzerdefinierte-filter

Kapitel 4: Benutzerdefinierte Richtlinien

Syntax

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

Parameter

Parameter	Einzelheiten
`id`	String - Die Direktive-ID, die ohne das `v-` Präfix verwendet wird. (Fügen Sie das `v-` Präfix hinzu, wenn Sie es verwenden)
`definition`	Object - Ein Definitionsobjekt kann mehrere Hook-Funktionen (alle optional) bereitstellen: `bind` , `update` und `unbind`

Examples

Grundlagen

Zusätzlich zu den Standardvorgaben, die im Kern ausgeliefert werden, können Sie mit Vue.js auch benutzerdefinierte Anweisungen registrieren. Benutzerdefinierte Anweisungen bieten einen Mechanismus zum Zuordnen von Datenänderungen zu beliebigem DOM-Verhalten.

Sie können eine globale benutzerdefinierte Direktive mit der Methode `Vue.directive(id, definition)` registrieren und eine Direktive-ID übergeben, auf die ein Definitionsobjekt folgt. Sie können auch eine lokale benutzerdefinierte Direktive registrieren, indem Sie sie in die `directives` Option einer Komponente aufnehmen.

Hook-Funktionen

- **bind** : Wird nur einmal aufgerufen, wenn die Direktive zuerst an das Element gebunden wird.
- **update** : wurde zum ersten Mal unmittelbar nach dem `bind` mit dem Anfangswert aufgerufen und dann erneut, wenn sich der Bindungswert ändert. Der neue Wert und der vorherige Wert werden als Argument bereitgestellt.
- **Unbind** : wird nur einmal aufgerufen, wenn die Direktive vom Element getrennt wird.

```
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()
    }
})
```

Nach der Registrierung können Sie es wie folgt in Vue.js-Vorlagen verwenden (denken Sie daran, das `v-` Präfix hinzuzufügen):

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

Wenn Sie nur die `update` benötigen, können Sie anstelle des Definitionsobjekts nur eine einzige Funktion übergeben:

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

Eigenschaften der Direktiveinstanz

Alle Hook - Funktionen werden in die eigentliche Richtlinie Objekt kopiert, die Sie in diesen Funktionen als ihre zugreifen können `this` Kontext. Das Direktive-Objekt macht einige nützliche Eigenschaften verfügbar:

- **el** : das Element, an das die Direktive gebunden ist.
- **vm** : Der Kontext ViewModel, dem diese Direktive gehört.
- **Ausdruck** : Der Ausdruck der Bindung, ohne Argumente und Filter.
- **arg** : das Argument, falls vorhanden.
- **name** : Der Name der Direktive ohne Präfix.
- **Modifikatoren** : ein Objekt, das ggf. Modifikatoren enthält.
- **Deskriptor** : Ein Objekt, das das Analyseergebnis der gesamten Direktive enthält.
- **params** : ein Objekt, das Parameterattribute enthält. Unten erklärt.

  Sie sollten alle diese Eigenschaften als schreibgeschützt behandeln und niemals ändern. Sie können dem Direktive-Objekt auch benutzerdefinierte Eigenschaften hinzufügen. Achten Sie jedoch darauf, vorhandene interne Eigenschaften nicht versehentlich zu überschreiben.

Ein Beispiel für eine benutzerdefinierte Direktive, die einige dieser Eigenschaften verwendet:

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!'
  }
})
```

Ergebnis

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

Objekt-Literal

Wenn Ihre Anweisung mehrere Werte erfordert, können Sie auch ein JavaScript-Objektliteral übergeben. Denken Sie daran, dass Anweisungen jeden gültigen JavaScript-Ausdruck annehmen können:

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!"
})
```

Wörtliche Modifikation

Wenn eine Anweisung mit dem literal-Modifizierer verwendet wird, wird der Attributwert als einfacher String interpretiert und direkt an die `update` . Die `update` wird auch nur einmal aufgerufen, da eine einfache Zeichenfolge nicht reaktiv sein kann.

HTML

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

JavaScript

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

Benutzerdefinierte Richtlinien online lesen: https://riptutorial.com/de/vue-js/topic/2368/benutzerdefinierte-richtlinien

Kapitel 5: Beobachter

Examples

Wie es funktioniert

Sie können die Dateneigenschaften einer beliebigen Vue-Instanz überwachen. Beim Ansehen einer Eigenschaft lösen Sie bei Änderung eine Methode aus:

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

Sie können den alten und den neuen Wert abrufen:

```
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'
    }
}
```

Wenn Sie verschachtelte Eigenschaften für ein Objekt überwachen müssen, müssen Sie die Eigenschaft `deep` :

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

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

Wann werden die Daten aktualisiert?

Wenn Sie den Watcher auslösen müssen, bevor Sie an einem Objekt neue Änderungen vornehmen, müssen Sie die `nextTick()` -Methode verwenden:

```
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'
        }
    }
}
```

Beobachter online lesen: https://riptutorial.com/de/vue-js/topic/7988/beobachter

Kapitel 6: Berechnete Eigenschaften

Bemerkungen

Daten vs. berechnete Eigenschaften

Der Hauptfall der Unterschiede zwischen den `data` und den `computed` Eigenschaften einer `Vue` Instanz hängt vom potenziellen Status oder der Wahrscheinlichkeit der Änderung der Daten ab. Bei der Entscheidung, welche Kategorie ein bestimmtes Objekt sein soll, können diese Fragen hilfreich sein:

- Ist das ein konstanter Wert? (**Daten**)
- Hat dies die Möglichkeit sich zu ändern? (**berechnet** oder **Daten**)
- Ist der Wert davon abhängig vom Wert anderer Daten? (**berechnet**)
- Benötigen Sie zusätzliche Daten oder Berechnungen, bevor sie verwendet werden können? (**berechnet**)
- Wird sich der Wert nur unter bestimmten Umständen ändern? (**Daten**)

Examples

Basisbeispiel

Vorlage

```
<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
    }
  }
})
```

Ergebnis

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

Hier haben wir eine berechnete Eigenschaft `b` . Die von uns bereitgestellte Funktion wird als Getter-Funktion für die Eigenschaft `vm.b` :

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

Der Wert von `vm.b` hängt immer vom Wert von `vm.a` .

Sie können an berechnete Eigenschaften in Vorlagen genau wie eine normale Eigenschaft Datenbinden. Vue ist sich bewusst, dass `vm.b` von `vm.b` abhängt. `vm.a` werden alle Bindungen aktualisiert, die von `vm.b` wenn sich `vm.a` ändert.

Berechnete Eigenschaften vs. Watch

Vorlage

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

Beispiel ansehen

```
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
})
```

Berechnetes Beispiel

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

Berechnete Setter

Berechnete Eigenschaften werden automatisch neu berechnet, wenn sich Daten ändern, von denen die Berechnung abhängt. Wenn Sie jedoch eine berechnete Eigenschaft manuell ändern müssen, können Sie in Vue dazu eine Setter-Methode erstellen:

Vorlage (aus dem Grundbeispiel oben):

```
<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
      }
    }
  }
```

Sie können jetzt entweder den Getter oder den Setter aufrufen:

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

`vm.b = 4` den Setter auf und setzt dies.a auf 3; Durch die Erweiterung wird vm.b mit 4 ausgewertet.

Berechnete Setter für v-Modell verwenden

Sie benötigen möglicherweise ein `v-model` für eine berechnete Eigenschaft. Normalerweise aktualisiert das V-Modell den berechneten Eigenschaftswert nicht.

Die Vorlage:

```
<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>
```

Styling:

```
#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;
}
```

Die Komponente:

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

```
    }
})
```

Geige Hier würden Sie sehen, wenn Sie auf das Optionsfeld klicken, hat dies keinerlei Verwendung. Ihr Status ist noch online.

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

Geige Und jetzt können Sie sehen, wie das Umschalten geschieht, wenn das Kontrollkästchen aktiviert / deaktiviert ist.

Berechnete Eigenschaften online lesen: https://riptutorial.com/de/vue-js/topic/2371/berechnete-eigenschaften

Kapitel 7: Datenbindung

Examples

Text

Die grundlegendste Form der Datenbindung ist die Textinterpolation mit der "Moustache" -Syntax (doppelte geschweifte Klammern):

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

Das Mustache-Tag wird durch den Wert der Eigenschaft `msg` des entsprechenden Datenobjekts ersetzt. Sie wird auch aktualisiert, wenn sich die `msg` -Eigenschaft des Datenobjekts ändert.

Sie können auch einmalige Interpolationen durchführen, die sich bei Datenänderungen nicht aktualisieren:

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

Rohes HTML

Der doppelte Schnurrbart interpretiert die Daten als Text, nicht als HTML. Um echtes HTML auszugeben, müssen Sie dreifache Schnurrbärte verwenden:

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

Der Inhalt wird als einfaches HTML eingefügt - Datenbindungen werden ignoriert. Wenn Sie Schablonenstücke wiederverwenden müssen, sollten Sie Partials verwenden.

Attribute

Schnurrbärte können auch in HTML-Attributen verwendet werden:

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

Beachten Sie, dass Attributinterpolationen in Vue.js-Anweisungen und speziellen Attributen nicht zulässig sind. Keine Sorge, Vue.js wird Warnungen auslösen, wenn Schnurrbärte an falschen Stellen verwendet werden.

Filter

Mit Vue.js können Sie optionale "Filter" an das Ende eines Ausdrucks anhängen, der mit dem "Pipe" -Symbol gekennzeichnet ist:

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

Hier "leiten" wir den Wert des `message` durch den eingebauten `capitalize` , der eigentlich nur eine JavaScript-Funktion ist, die den kapitalisierten Wert zurückgibt. Vue.js bietet eine Reihe von integrierten Filtern. Wir werden später darüber sprechen, wie Sie Ihre eigenen Filter schreiben.

Beachten Sie, dass die Pipe-Syntax nicht Teil der JavaScript-Syntax ist. Daher können Sie keine Filter in Ausdrücken mischen. Sie können sie nur am Ende eines Ausdrucks anhängen.

Filter können verkettet werden:

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

Filter können auch Argumente annehmen:

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

Die Filterfunktion erhält immer den Wert des Ausdrucks als erstes Argument. Zitierte Argumente werden als einfache Zeichenfolge interpretiert, während nicht zitierte Argumente als Ausdrücke ausgewertet werden. Hier wird der einfache String `'arg1'` als zweites Argument an den Filter übergeben, und der Wert des Ausdrucks `arg2` wird ausgewertet und als drittes Argument übergeben.

Datenbindung online lesen: https://riptutorial.com/de/vue-js/topic/1213/datenbindung

Kapitel 8: Die Array-Änderungswarnung ist zu beachten

Einführung

Wenn Sie versuchen, einen Wert eines Elements an einem bestimmten Index eines Arrays festzulegen, das mit der Datenoption initialisiert wurde, kann vue die Änderung nicht erkennen und löst keine Aktualisierung des Status aus. Um diesen Nachteil zu umgehen, sollten Sie entweder Vue. $ Set von vue oder die Methode Array.prototype.splice verwenden

Examples

Verwendung von Vue. $ Set

In Ihrer Methode oder einem beliebigen Lebenszyklus-Hook, der das Arrayelement am jeweiligen Index ändert

```
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');
        }
    }
})
```

Hier ist der Link zur Geige

Array.prototype.splice verwenden

Sie können dieselbe Änderung anstelle von `Vue.$set` mithilfe des `splice()` des Array-Prototyps

```
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');
        }
    }
})
```

Für verschachteltes Array

Wenn Sie ein verschachteltes Array haben, können Sie Folgendes tun

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

Hier ist der Link zum jsfiddle

Array von Objekten, die Arrays enthalten

```
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');
        }
    }
})
```

Hier ist der Link zur Geige

Die Array-Änderungswarnung ist zu beachten online lesen: https://riptutorial.com/de/vue-js/topic/10679/die-array-anderungswarnung-ist-zu-beachten

Kapitel 9: Dynamische Komponenten

Bemerkungen

`<component>` ist ein reserviertes Komponentenelement, nicht mit der Komponenteninstanz verwechselt werden.

`v-bind` ist eine Direktive. Direktiven wird mit *v-* vorangestellt, um anzuzeigen, dass es sich um von Vue bereitgestellte spezielle Attribute handelt.

Examples

Beispiel für einfache dynamische Komponenten

Wechseln Sie dynamisch zwischen mehreren Komponenten mit dem `<component>` -Element und übergeben Sie die Daten an *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>
```

Ausschnitt:

Live Demo

Seitennavigation mit Keep-Alive

Manchmal möchten Sie die ausgeschalteten Komponenten im Speicher behalten. Um dies zu ermöglichen, sollten Sie das Element `<keep-alive>` verwenden:

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;
}
```

Ausschnitt:

Live Demo

Dynamische Komponenten online lesen: https://riptutorial.com/de/vue-js/topic/7702/dynamische-komponenten

Kapitel 10: Einzelne Dateikomponenten

Einführung

Beschreiben, wie einzelne Dateikomponenten in einer .vue-Datei erstellt werden.

Insbesondere die Designentscheidungen, die getroffen werden können.

Examples

Beispiel für eine .vue-Komponentendatei

```
<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>
```

Einzelne Dateikomponenten online lesen: https://riptutorial.com/de/vue-js/topic/10118/einzelne-dateikomponenten

Kapitel 11: Ereignisbus

Einführung

Ereignisbusse sind eine nützliche Möglichkeit zur Kommunikation zwischen Komponenten, die nicht direkt zusammenhängen, dh keine Eltern-Kind-Beziehung haben.

Es ist nur eine leere `vue` Instanz, die zum `$emit` Events oder zum Abhören von `$on` die besagten Events verwendet werden kann.

Syntax

1. voreingestellter neuer Vue () exportieren

Bemerkungen

Verwenden Sie vuex, wenn Ihre Anwendung viele Komponenten enthält, die voneinander Daten erfordern.

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
    })
  }
})
```

Ereignisbus online lesen: https://riptutorial.com/de/vue-js/topic/9498/ereignisbus

Kapitel 12: Komponenten

Bemerkungen

In Komponente (n):

props ist ein Array von String-Literalen oder Objektreferenzen, die zum Übergeben von Daten von der übergeordneten Komponente verwendet werden. Es kann auch in Form eines Objekts vorliegen, wenn eine feinere Steuerung gewünscht wird, z. B. durch Angabe von Standardwerten, Art der akzeptierten Daten, ob dies erforderlich oder optional ist

data muss eine Funktion sein, die ein Objekt anstelle eines einfachen Objekts zurückgibt. Dies liegt daran, dass wir für jede Instanz der Komponente eigene Daten zur Wiederverwendbarkeit benötigen.

events ist ein Objekt, das Listener für Ereignisse enthält, auf die die Komponente durch Verhaltensänderungen reagieren kann

Methodenobjekt , das Funktionen enthält, die das der Komponente zugeordnete Verhalten definieren

Berechnete Eigenschaften sind wie Beobachter oder Observable. Wenn sich Abhängigkeiten ändern, werden die Eigenschaften automatisch neu berechnet und Änderungen werden sofort in DOM wiedergegeben, wenn DOM berechnete Eigenschaften verwendet

ready ist der Lebenszyklushaken einer Vue-Instanz

Examples

Komponentenbereich (nicht global)

Demo

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
  }
});
```

Was sind Komponenten und wie definiere ich Komponenten?

Komponenten in Vue sind wie Widgets. Sie erlauben uns, wiederverwendbare benutzerdefinierte Elemente mit dem gewünschten Verhalten zu schreiben.

Sie sind nichts anderes als Objekte, die alle Optionen enthalten können, die der Stamm oder eine Vue-Instanz enthalten kann, einschließlich einer zu rendernden HTML-Vorlage.

Komponenten bestehen aus:

- HTML-Markup: Die Vorlage der Komponente
- CSS-Stile: wie das HTML-Markup angezeigt wird
- JavaScript-Code: die Daten und das Verhalten

Diese können jeweils in eine separate Datei oder als einzelne Datei mit der Erweiterung `.vue` . Im Folgenden sind Beispiele aufgeführt, die beide Möglichkeiten zeigen:

.VUE - als einzelne Datei für die Komponente

```
<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>
```

Dateien trennen

hello-world.js - die JS-Datei für das Komponentenobjekt

```
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;
```

```
        }
    }
}
```

hallo-world.template.html

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

hallo-world.css

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

In diesen Beispielen wird die Es2015-Syntax verwendet. Daher wird Babel benötigt, um sie für ältere Browser nach es5 zu kompilieren.
Babel wird zusammen mit **Browserify + vueify** oder **Webpack + vue-loader** benötigt, um `hello-world.vue` zu kompilieren.

Nachdem wir nun die `hello-world` Komponente definiert haben, sollten wir sie bei Vue registrieren.

Dies kann auf zwei Arten erfolgen:

Registrieren Sie sich als globale Komponente
In der Datei `main.js` (Einstiegspunkt zur App) können wir jede Komponente global mit `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>'
});
```

Oder registrieren Sie es lokal in einer übergeordneten Komponente oder Stammkomponente

```
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
});
```

Globale Komponenten können überall in der Vue-Anwendung verwendet werden.

Lokale Komponenten sind nur für die Verwendung in der übergeordneten Komponente verfügbar, mit der sie registriert sind.

Fragmentkomponente
Möglicherweise erhalten Sie einen Konsolenfehler, der Ihnen mitteilt, dass Sie nichts tun können, da Ihre *Komponente* eine *Fragmentkomponente ist* . Um dieses Problem zu lösen, wickeln Sie Ihre Komponentenvorlage einfach in ein einzelnes Tag ein, beispielsweise ein `<div>` .

Lokale Registrierung von Komponenten

Eine Komponente kann entweder global oder lokal registriert werden (Bindung an eine andere bestimmte Komponente).

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

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

Diese neue Komponente () ist nur innerhalb des Gültigkeitsbereichs (der Vorlage) der übergeordneten Komponente verfügbar.

Inline-Registrierung

Sie können eine Komponente in einem Schritt erweitern und registrieren:

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

Auch wenn die Komponente lokal registriert ist:

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

Datenregistrierung in Komponenten

Wenn Sie beim Registrieren einer Komponente ein Objekt an die `data` -Eigenschaft übergeben, würden alle Instanzen der Komponente auf dieselben Daten zeigen. Um dies zu lösen, müssen wir `data` von einer Funktion zurückgeben.

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

Veranstaltungen

Eine Möglichkeit, wie Komponenten mit ihren Vorfahren / Nachkommen kommunizieren können, ist über benutzerdefinierte Kommunikationsereignisse. Alle Vue-Instanzen sind auch Emitter und implementieren eine benutzerdefinierte Ereignisschnittstelle, die die Kommunikation innerhalb eines Komponentenbaums erleichtert. Wir können Folgendes verwenden:

- `$on` : Hören Sie sich Ereignisse an, die von Vorfahren oder Nachkommen dieser Komponente ausgegeben werden.
- `$broadcast` : Gibt ein Ereignis aus, das sich nach unten an alle Nachkommen ausbreitet.
- `$dispatch` : Gibt ein Ereignis aus, das zuerst auf der Komponente selbst ausgelöst wird und sich dann an alle Vorfahren weiterleitet.
- `$emit` : Löst ein Ereignis für sich aus.

Beispielsweise möchten wir eine bestimmte Schaltflächenkomponente in einer Formularkomponente ausblenden, wenn das Formular übermittelt wird. Auf dem übergeordneten Element:

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

Auf dem untergeordneten Element:

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

Einige Dinge zu beachten:

- Wenn ein Ereignis eine Komponente findet, die es abhört und ausgelöst wird, wird es nicht weiter verbreitet, es sei denn, der Funktionsrückruf in dieser Komponente gibt `true` .
- `$dispatch()` immer zuerst bei der Komponente ausgelöst, die sie ausgegeben hat.
- Wir können beliebig viele Argumente an den Ereignishandler übergeben. `this.$broadcast('submit-form', this.formData, this.formStatus)` erlaubt uns den Zugriff auf diese Argumente wie `'submit-form': function (formData, formStatus) {}`

Komponenten online lesen: https://riptutorial.com/de/vue-js/topic/1775/komponenten

Kapitel 13: Kundenspezifische Komponenten mit V-Modell

Einführung

Oft müssen wir einige Komponenten erstellen, die einige Aktionen / Vorgänge an Daten ausführen, und dies in der übergeordneten Komponente. In den meisten `vuex` wäre `vuex` eine bessere Lösung, aber in Fällen, in denen das Verhalten der `vuex` Komponente nichts mit dem Status der Anwendung zu tun hat, z

Einzelne Speicher für jede Komponente zu haben, wenn sie sich gewöhnen, wird kompliziert.

Bemerkungen

Um ein `v-model` an einer Komponente zu haben, müssen Sie zwei Bedingungen erfüllen.

1. Es sollte eine Requisite namens "value" haben
2. Es sollte ein `input` mit dem von den übergeordneten Komponenten erwarteten Wert ausgeben.

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

ist nur syntaktischer Zucker für

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

Examples

v-Modell auf einer Zählerkomponente

Hier ist der `counter` eine untergeordnete Komponente, auf die von einer `demo` Komponente zugegriffen wird, die eine `v-model` Komponente ist.

```
// 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()
    }
  }
})
```

Diese untergeordnete Komponente gibt jedes Mal ein `result` aus, wenn die Methoden `sub()` oder `add()` aufgerufen werden.

```
// 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>
```

Da `v-model` , die auf der untergeordneten Komponente ist, eine Stütze mit dem Namen `value` wurde zur gleichen Zeit gesendet wird , gibt es ein Eingabeereignis auf dem `counter` , die den Wert der untergeordneten Komponente wiederum liefern.

Kundenspezifische Komponenten mit V-Modell online lesen: https://riptutorial.com/de/vue-js/topic/9353/kundenspezifische-komponenten-mit-v-modell

Kapitel 14: Lebenszyklus-Haken

Examples

Haken für Vue 1.x

- **`init`**

  Wird synchron aufgerufen, nachdem die Instanz initialisiert wurde und bevor eine erste Datenbeobachtung erfolgt.

- **`created`**

  Wird synchron aufgerufen, nachdem die Instanz erstellt wurde. Dies tritt vor dem `$el` Setup auf, aber nach `data observation` wurden `computed properties` , `watch/event callbacks` und `methods` .

- **`beforeCompile`**

  Unmittelbar vor dem Kompilieren der Vue-Instanz.

- **`compiled`**

  Sofort nach Abschluss der Kompilierung. Alle `directives` sind verknüpft, aber noch bevor `$el` verfügbar ist.

- **`ready`**

  Tritt nach der Kompilierung auf *und* `$el` ist abgeschlossen und die Instanz wird zum ersten Mal in das DOM eingefügt.

- **`attached`**

  Tritt auf, wenn `$el` durch eine `directive` an das DOM angehängt wird oder die Instanz `$appendTo()` .

- **`detached`**

  Wird aufgerufen, wenn `$el` von der DOM- oder Instanzmethode entfernt / getrennt wird.

- **`beforeDestroy`**

  Unmittelbar bevor die Vue-Instanz zerstört wird, ist sie dennoch voll funktionsfähig.

- **`destroyed`**

  Wird aufgerufen, nachdem eine Instanz zerstört wurde. Alle `bindings` und `directives` wurden

bereits ungebunden und untergeordnete Instanzen wurden ebenfalls zerstört.

Verwendung in einer Instanz

Da *alle* Lebenszyklus-Hooks in `Vue.js` nur `functions` , können Sie *alle* direkt in der Instanzdeklaration platzieren.

```
//JS
new Vue({

    el: '#example',

    data: {
        ...
    },

    methods: {
        ...
    },

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

    ready: function() {
        ...
    }

});
```

Häufige Fallstricke: Zugriff auf DOM über den `ready ()` - Haken

Eine häufige Verwendung des `ready()` Hooks ist der Zugriff auf das DOM, z. B. um ein Javascript-Plugin zu initiieren, die Dimensionen eines Elements zu ermitteln usw.

Das Problem

Aufgrund des asynchronen DOM-Aktualisierungsmechanismus von Vue kann nicht garantiert werden, dass das DOM beim Aufruf des `ready()` -Hooks vollständig aktualisiert wurde. Dies führt normalerweise zu einem Fehler, da das Element nicht definiert ist.

Die Lösung

In dieser Situation kann die `$nextTick()` hilfreich sein. Diese Methode verschiebt die Ausführung der bereitgestellten Rückruffunktion bis nach dem nächsten Tick, was bedeutet, dass sie ausgelöst wird, wenn garantiert ist, dass alle DOM-Aktualisierungen abgeschlossen sind.

Beispiel:

```
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.
    })
  }
}
```

Lebenszyklus-Haken online lesen: https://riptutorial.com/de/vue-js/topic/1852/lebenszyklus-haken

Kapitel 15: Listen-Rendering

Examples

Grundlegende Verwendung

Eine Liste kann mit der `v-for` Direktive gerendert werden. Die Syntax erfordert, dass Sie das Quellenarray angeben, für das eine Iteration durchgeführt werden soll, sowie einen Alias, der zum Verweisen auf jedes Element in der Iteration verwendet wird. Im folgende Beispiel verwenden wir `items` als Quelle Array und `item` als Alias für jedes Element.

HTML

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

Skript

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

Sie können eine Arbeits Demo sehen hier .

Nur HTML-Elemente rendern

In diesem Beispiel werden fünf `<li>` -Tags gerendert

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

Schweine-Countdown-Liste

```
<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/

Iteration über ein Objekt

`v-for` kann zum Durchlaufen von Objektschlüsseln (und Werten) verwendet werden:

HTML:

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

Skript:

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

Listen-Rendering online lesen: https://riptutorial.com/de/vue-js/topic/1972/listen-rendering

Kapitel 16: Mixins

Examples

Global Mixin

Sie können ein Mixin auch global anwenden. Vorsicht walten lassen! Wenn Sie ein Mixin global anwenden, wirkt sich dies auf **jede** später erstellte Vue-Instanz aus. Bei korrekter Verwendung kann dies verwendet werden, um Verarbeitungslogik für benutzerdefinierte Optionen einzufügen:

```
// 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!"
```

> Verwenden Sie globale Mixins sparsam und sorgfältig, da sie sich auf jede einzelne Vue-Instanz auswirkt, einschließlich der Komponenten von Drittanbietern. In den meisten Fällen sollten Sie es nur für die Handhabung benutzerdefinierter Optionen verwenden, wie im obigen Beispiel gezeigt.

Zusammenführungsstrategien für benutzerdefinierte Optionen

Beim Zusammenführen von benutzerdefinierten Optionen wird die Standardstrategie verwendet, bei der der vorhandene Wert einfach überschrieben wird. Wenn Sie möchten, dass eine benutzerdefinierte Option mithilfe einer benutzerdefinierten Logik zusammengeführt wird, müssen Sie eine Funktion an `Vue.config.optionMergeStrategies` :

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

Für die meisten objektbasierten Optionen können Sie einfach dieselbe Strategie verwenden, die von `methods` verwendet wird:

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

Grundlagen

Mixins sind eine flexible Möglichkeit, wiederverwendbare Funktionen für Vue-Komponenten zu verteilen. Ein Mixin-Objekt kann beliebige Komponentenoptionen enthalten. Wenn eine Komponente ein Mixin verwendet, werden alle Optionen im Mixin in die eigenen Optionen der Komponente „gemischt“.

```
// 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!"
```

Option Zusammenführen

Wenn ein Mixin und die Komponente selbst überlappende Optionen enthalten, werden sie mit geeigneten Strategien "zusammengeführt". Beispielsweise werden Hook-Funktionen mit demselben Namen in einem Array zusammengeführt, sodass alle aufgerufen werden. Außerdem werden Mixin-Hooks **vor** den eigenen Hooks der Komponente aufgerufen:

```
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"
```

Optionen, die Objektwerte erwarten, z. B. `methods` , `components` und `directives` , werden in dasselbe Objekt eingefügt. Die Optionen der Komponente haben Vorrang, wenn sich in diesen Objekten widersprüchliche Schlüssel befinden:

```
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"
```

Beachten Sie, dass in `Vue.extend()` die gleichen Zusammenführungsstrategien verwendet werden.

Mixins online lesen: https://riptutorial.com/de/vue-js/topic/2562/mixins

Kapitel 17: Modifikatoren

Einführung

Es gibt einige häufig verwendete Vorgänge wie `event.preventDefault()` oder `event.stopPropagation()` in Event- `event.stopPropagation()` . Obwohl dies innerhalb von Methoden problemlos möglich ist, wäre es besser, wenn es sich bei den Methoden um reine Datenlogik handeln kann, anstatt sich mit DOM-Ereignisdetails zu befassen.

Examples

Event Modifiers

Vue stellt Ereignismodifizierer für `v-on` bereit, indem Direktive-Postfixes aufgerufen werden, die durch einen Punkt gekennzeichnet sind.

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

Zum Beispiel:

```
<!-- 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>
```

Schlüsselmodifikatoren

Beim Abhören von Tastaturereignissen müssen wir häufig nach gängigen Tastencodes suchen. Das Erinnern an alle keyCodes ist umständlich, daher bietet Vue Aliasnamen für die am häufigsten verwendeten Schlüssel:

- `.enter`
- `.tab`
- `.delete` (erfasst die Tasten "Löschen" und "Rücktaste")
- `.esc`
- `.space`
- `.up`

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

Zum Beispiel:

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

Eingangsmodifikatoren

- `.trim`

Wenn Sie Benutzereingaben automatisch getrimmt werden möchten, können Sie den hinzufügen `trim` Modifikator zu Ihrem `v-model` verwaltet Eingänge:

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

- `.number`

Wenn Sie möchten, dass die Benutzereingabe automatisch als Zahl typisiert wird, können Sie wie folgt vorgehen:

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

- `.lazy`

Im Allgemeinen synchronisiert `v-model` die Eingabe mit den Daten nach jedem Eingabeereignis. Sie können jedoch den Modifikator " `lazy` hinzufügen, statt nach Änderungsereignissen zu synchronisieren:

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

Modifikatoren online lesen: https://riptutorial.com/de/vue-js/topic/8612/modifikatoren

Kapitel 18: Plugins

Einführung

Vue-Plugins fügen globale Funktionen wie globale Methoden, Anweisungen, Übergänge, Filter, Instanzmethoden und Objekte hinzu und fügen einige Komponentenoptionen mithilfe von Mixins ein

Syntax

- MyPlugin.install = Funktion (Vue, Optionen) {}

Parameter

Name	Beschreibung
Vue	Vue Konstruktor, eingespritzt von Vue
Optionen	Zusätzliche Optionen bei Bedarf

Bemerkungen

In den meisten Fällen müssen Sie Vue explizit mitteilen, dass Sie ein Plugin verwenden

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

Optionen übergeben

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

Examples

Einfacher Logger

```
//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
    }
  }
}
```

Bevor Sie Ihre Vue-Instanz aufrufen, müssen Sie Ihr Plugin registrieren

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

Vue.use(Logger);

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

Jetzt können Sie `this.$log` für jede untergeordnete Komponente aufrufen

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

Plugins online lesen: https://riptutorial.com/de/vue-js/topic/8726/plugins

Kapitel 19: Polyfill "Webpack" Vorlage

Parameter

Dateien oder Pakete	Befehl oder Konfiguration zum Ändern
Babel-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']`

Bemerkungen

Die oben beschriebenen Konfigurationen, das Beispiel, das eine nicht standardisierte Funktion verwendet, funktionieren mit dem "Internet Explorer" und der `npm test` wird bestanden.

Examples

Verwendung von Funktionen zum Polyfill (ex: find)

```
<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>
```

Polyfill "Webpack" Vorlage online lesen: https://riptutorial.com/de/vue-js/topic/9174/polyfill--webpack--vorlage

Kapitel 20: Requisiten

Bemerkungen

camelCase <=> Kebab-Fall

Denken Sie bei der Definition der Namen Ihrer `props` immer daran, dass die Namen von HTML-Attributen nicht zwischen Groß- und Kleinschreibung unterscheiden. Das heißt, wenn Sie in Ihrer Komponentendefinition eine `prop` im Kamelfall definieren ...

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

... Sie müssen es in Ihrer HTML-Komponente als my-prop bezeichnen.

Examples

Übergabe von Daten an Eltern mit Requisiten

In Vue.js verfügt jede Komponenteninstanz über ***einen eigenen isolierten Bereich*** . Wenn also eine übergeordnete Komponente über eine untergeordnete Komponente verfügt, hat die untergeordnete Komponente ihren eigenen isolierten Bereich und die übergeordnete Komponente ihren eigenen isolierten Bereich.

Bei mittelgroßen bis großen Anwendungen werden durch das Befolgen der Best Practices-Konventionen während der Entwicklungsphase und danach während der Wartung zahlreiche Kopfschmerzen vermieden. Es ist eines der folgenden Dinge, die vermieden werden, dass ***Elterndaten direkt von der untergeordneten Komponente referenziert / mutiert werden*** . Wie verweisen wir dann auf die übergeordneten Daten innerhalb einer untergeordneten Komponente?

Alle in einer untergeordneten Komponente erforderlichen übergeordneten Daten sollten als `props` vom übergeordneten `props` an das untergeordnete Element weitergegeben werden.

Use Case: Angenommen , wir eine User - Datenbank mit zwei Tabellen haben `users` und `addresses` mit den folgenden Feldern:
`users` Tabelle

Name	Telefon	Email
John Mclane	(1) 234 5678 9012	john@dirhard.com
James Bond	(44) 777 0007 0077	bond@mi6.com

`addresses` Tabelle

Block	Straße	Stadt
Nakatomi-Türme	Broadway	New York
Mi6 Haus	Buckingham Road	London

und wir möchten drei Komponenten haben, um entsprechende Benutzerinformationen überall in unserer App anzuzeigen

Benutzerkomponente.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:''
        }
    },
}
```

contact-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>
```

Wir zeigen die `phone` und `email` Daten an. `email` handelt es sich um Eigenschaften von `user-component` in `contact-details` die keine Telefon- oder E-Mail-Daten enthalten.

Daten als Requisiten übergeben

Innerhalb der `user-component.js` in der ***template-*** Eigenschaft, in der wir die Komponente `<contact-details>` enthalten, übergeben wir das ***Telefon*** und die ***E-Mail-*** Daten von `<user-component>` (übergeordnete Komponente) an `<contact-details>` (untergeordnete Komponente) durch dynamisches Binden an die **Requisiten** - `:phone="phone"` und `:email="email` ist identisch mit `v-bind:phone="phone"` und `v-bind:email="email"`

Requisiten - Dynamische Bindung

Da wir die Requisiten dynamisch binden, wird jede Änderung in ***Telefon*** oder ***E-Mail*** innerhalb der übergeordneten Komponente, dh `<user-component>` , sofort in der untergeordneten Komponente, dh `<contact-details>` , widergespiegelt.

Requisiten - als Literale

Wenn wir jedoch die Werte von ***phone*** und ***email*** als String-Literal-Werte übergeben hätten, z. B. `phone="(44) 777 0007 0077" email="bond@mi6.com"` , würde dies keine Datenänderungen widerspiegeln, die im übergeordneten `phone="(44) 777 0007 0077" email="bond@mi6.com"` Komponente.

One-Way-Bindung

Standardmäßig ist die Richtung der Änderungen von oben nach unten, dh jede Änderung an dynamisch gebundenen Requisiten in der übergeordneten Komponente wird an die untergeordnete Komponente weitergegeben, aber jede Änderung an den Prop-Werten in einer untergeordneten Komponente wird nicht an die übergeordnete Komponente weitergegeben.

Zum Beispiel: Wenn wir innerhalb der `<contact-details>` die E-Mail von `bond@mi6.com` in `jamesbond@mi6.com` , enthalten die übergeordneten Daten, dh die Telefondateneigenschaft in `<user-component>` , weiterhin den Wert `bond@mi6.com` .

Wenn wir jedoch den Wert von email von `bond@mi6.com` in `jamesbond@mi6.co` in der übergeordneten Komponente (in unserem Anwendungsfall `<user-component>` ändern, dann wird der Wert von email in der `jamesbond@mi6.co` Komponente (`<contact-details>` In unserem Anwendungsfall wird automatisch in `jamesbond@mi6.com` Änderung im übergeordneten `jamesbond@mi6.com` wird sofort an das `jamesbond@mi6.com` .

Zwei-Wege-Bindung

Wenn Sie eine `:email.sync="email"` Bindung `:email.sync="email"` Bindung explizit angeben `:email.sync="email"` statt `:email="email"` . Wenn wir nun den Wert von prop in der untergeordneten Komponente ändern, wird die Änderung auch in der übergeordneten Komponente angezeigt.

In einer mittelgroßen bis großen App ist es sehr schwierig, den übergeordneten Status des untergeordneten Status zu ändern, insbesondere während des Debugging. - **Seien Sie vorsichtig** .

In Vue.js 2.0 ist keine .sync-Option verfügbar. **Die wechselseitige Bindung für Requisiten wird in Vue.js 2.0 nicht mehr unterstützt** .

Einmalige Bindung

Es ist auch möglich, die explizite **einmalige** Bindung als `:email.once="email` zu definieren `:email.once="email` , sie ähnelt mehr oder weniger der Übergabe eines Literalwerts, da nachfolgende Änderungen des übergeordneten Eigenschaftswerts nicht an das `:email.once="email` .

VORBEHALT

Wenn **Object oder Array** als Requisite übergeben wird, werden sie **IMMER DURCH REFERENZEN EINGESCHLOSSEN. Dies** bedeutet, unabhängig vom explizit definierten Bindungstyp `:email.sync="email"` oder `:email="email"` oder `:email.once="email"` Wenn E-Mail ein Objekt oder ein Array im übergeordneten Element ist, wirkt sich die Änderung des Prop-Werts innerhalb der untergeordneten Komponente unabhängig vom Bindungstyp auf den Wert im übergeordneten Element aus.

Requisiten als Array

In der Datei `contact-details.js` haben wir `props:['phone', 'email']` definiert `props:['phone', 'email']` als Array. `contact-details.js` ist in Ordnung, wenn wir keine fein abgestimmte Steuerung mit Requisiten wünschen.

Requisiten als Objekt

Wenn wir eine feinere Kontrolle über Requisiten wünschen, wie

- wenn wir definieren wollen, welche Werte als Requisite akzeptabel sind
- Was sollte ein Standardwert für die Requisite sein
- ob ein Wert MUSS (erforderlich) für die Requisite übergeben werden muss oder optional ist

dann müssen wir die `address.js` verwenden, um die Requisiten zu definieren, wie wir es in `address.js` getan `address.js` .

Wenn wir wiederverwendbare Komponenten erstellen, die auch von anderen Entwicklern im Team verwendet werden können, ist es eine gute Praxis, Requisiten als Objekte zu definieren, sodass jeder, der die Komponente verwendet, eine klare Vorstellung davon hat, welche Art von Daten verwendet werden und ob es ist obligatorisch oder optional.

Es wird auch als ***Requisitenvalidierung bezeichnet*** . Der **Typ** kann einer der folgenden systemeigenen Konstruktoren sein:

- String
- Nummer
- Boolean
- Array
- Objekt
- Funktion
- oder ein benutzerdefinierter Konstruktor

Einige Beispiele für die Validierung von Requisiten aus 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 vs Kebab-Fall

Bei HTML-Attributen wird nicht zwischen `addresstype` und `addressType` unterschieden. `addresstype` `addressType` , dass zwischen `addresstype` und `addressType` nicht unterschieden werden kann. Wenn Sie Namen von camelCase-Attributen als Attribute verwenden, müssen Sie die entsprechenden Kebab-Großbuchstaben verwenden:
`addressType` sollte im HTML-Attribut als `address-type` geschrieben werden.

Dynamische Requisiten

So wie Sie Daten aus einer Sicht an das Modell binden können, können Sie Requisiten auch mit derselben V-Bind-Direktive binden, um Informationen von übergeordneten an untergeordnete Komponenten zu übergeben.

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>
```

Ergebnis

```
hello world
```

Props übergeben, während Sie Vue JSX verwenden

Wir haben eine übergeordnete Komponente: Beim Importieren einer untergeordneten Komponente werden Requisiten über ein Attribut übergeben. Hier ist das Attribut 'src' und wir übergeben auch das '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} />
        );
    }
};
```

Und eine Kindkomponente, wo wir Requisiten übergeben müssen. Wir müssen angeben, welche Requisiten wir passieren.

ChildComponent.js:

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

Requisiten online lesen: https://riptutorial.com/de/vue-js/topic/3080/requisiten

Kapitel 21: Schlüssel

Bemerkungen

Wichtig! Slots nach Render garantieren keine Reihenfolge der Positionen für Slots. Slot, der erste, hat nach dem Rendern möglicherweise eine andere Position.

Examples

Einzelne Slots verwenden

Einzelne Steckplätze werden verwendet, wenn eine untergeordnete Komponente nur einen `slot` in ihrer Vorlage definiert. Die obige `page` verwendet einen einzelnen Slot zum Verteilen von Inhalt.

Nachfolgend finden Sie ein Beispiel für die Vorlage der `page` , die einen einzelnen Steckplatz verwendet:

```
<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>
```

Um zu zeigen, wie der Slot funktioniert, können wir eine Seite wie folgt einrichten.

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

Das Endergebnis wird sein:

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

Wenn wir nichts zwischen die `page` stattdessen `<page></page>` , würden wir stattdessen das folgende Ergebnis liefern, da zwischen den `slot` Tags in der `page` Standardinhalt vorhanden ist.

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

Was sind Slots?

Slots bieten eine bequeme Möglichkeit, Inhalte von einer übergeordneten Komponente an eine untergeordnete Komponente zu verteilen. Dieser Inhalt kann aus Text, HTML oder anderen Komponenten bestehen.

Es kann manchmal hilfreich sein, sich Slots als ein Mittel vorzustellen, mit dem Inhalt direkt in die Vorlage einer untergeordneten Komponente eingefügt werden kann.

Slots sind besonders nützlich, wenn die Komponentenzusammensetzung unter der übergeordneten Komponente nicht immer gleich ist.

Nehmen wir das folgende Beispiel, wo wir eine `page` . Der Inhalt der Seite kann sich abhängig davon ändern, ob auf dieser Seite ein Artikel, ein Blogbeitrag oder ein Formular angezeigt wird.

Artikel

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

Blogeintrag

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

Bilden

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

Beachten Sie, wie sich der Inhalt der `page` ändern kann. Wenn wir keine Slots verwenden würden, wäre dies schwieriger, da der innere Teil der Vorlage repariert würde.

Denken Sie daran: *"Alles in der übergeordneten Vorlage wird im übergeordneten Bereich kompiliert. Alles in der untergeordneten Vorlage wird im untergeordneten Bereich kompiliert."*

Benannte Slots verwenden

Benannte Slots funktionieren ähnlich wie einzelne Slots. Sie können stattdessen Inhalte in verschiedenen Bereichen innerhalb Ihrer untergeordneten Komponentenvorlage verteilen.

Nehmen Sie die `page` aus dem vorherigen Beispiel, ändern Sie die Vorlage jedoch wie folgt:

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

Bei Verwendung der `page` können wir nun über das `slot` Attribut bestimmen, wo der Inhalt platziert wird:

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

Die resultierende Seite wird sein:

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

Wenn ein `slot` ohne definiert ist `name` - Attribut dann alle Inhalte , die innerhalb Komponente Tags platziert ist kein Spezifizierungs `slot` Attribut wird in diesem Schlitz angeordnet werden.

Siehe das Beispiel für die Mehrfacheinfügung in den offiziellen Dokumenten von Vue.js.

Slots in Vue JSX mit 'babel-plugin-transform-vue-jsx' verwenden

Wenn Sie VueJS2 verwenden und JSX zusammen damit verwenden möchten. In diesem Fall den

Schlitz zu verwenden, die Lösung mit Beispiel ist below.We verwenden hat `this.$slots.default` Es ist fast wie `this.props.children` in JS React.

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>
        );
    }
};
```

Schlüssel online lesen: https://riptutorial.com/de/vue-js/topic/4484/schlussel

Kapitel 22: Veranstaltungen

Examples

Ereignissyntax

So senden Sie ein Ereignis: `vm.$emit('new-message');`

Um ein Ereignis `vm.$on('new-message');` : `vm.$on('new-message');`

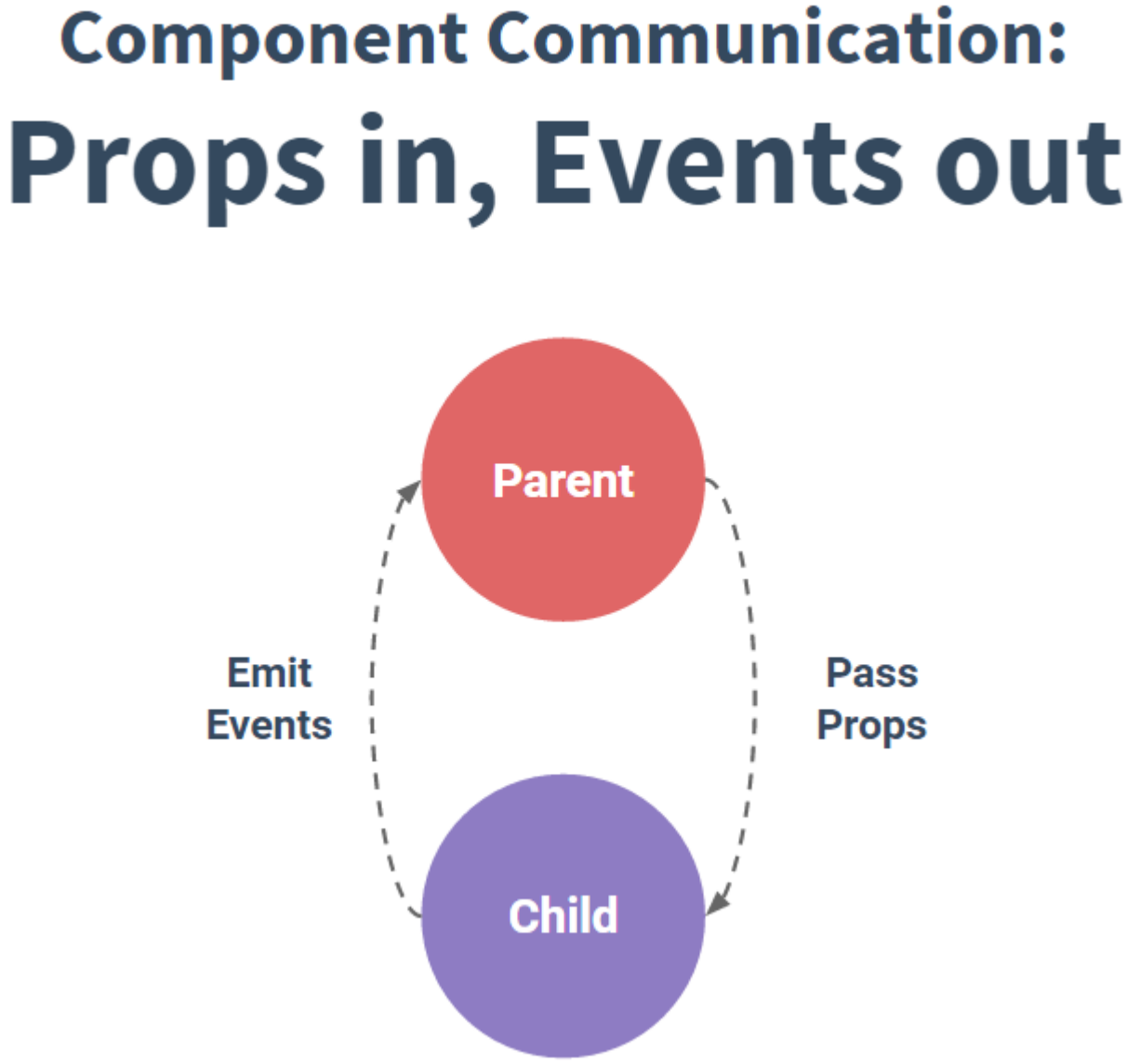
So senden Sie ein Ereignis für alle Komponenten **nach unten:** `vm.$broadcast('new-message');` . `vm.$broadcast('new-message');`

Um ein Ereignis für alle Komponenten **bis** senden: `vm.$dispatch('new-message');`

Hinweis: `$broadcast` und `$dispatch` werden in Vue2 nicht weiter unterstützt. (siehe Vue2-Funktionen)

Wann sollte ich Events nutzen?

Das folgende Bild zeigt, wie die Komponentenkommunikation funktionieren sollte. Das Bild stammt von The Progressive Framework Folien von Evan You (Entwickler von VueJS).



Hier ist ein Beispiel, wie es funktioniert:

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
  }
});
```

Das obige Beispiel kann verbessert werden!

Das obige Beispiel zeigt, wie die Komponentenkommunikation funktioniert. Im Falle einer benutzerdefinierten Eingabekomponente sollten Sie jedoch `v-model` , um die übergeordnete Variable mit dem eingegebenen Wert zu synchronisieren.

DEMO Vue1

DEMO Vue2

In Vue1 sollten Sie `.sync` für die Requisite verwenden, die an die `<message-box>` -Komponente gesendet wurde. Dies weist VueJS an, den Wert in der untergeordneten Komponente mit dem übergeordneten zu synchronisieren.

> **Denken Sie daran:** Jede Komponenteninstanz hat ihren eigenen isolierten Bereich.

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>
```

In Vue2 gibt es ein spezielles `'input'` -Ereignis, das Sie `$emit` . Mit diesem Ereignis können Sie ein `v-model` direkt in die `<message-box>` -Komponente einfügen. Das Beispiel sieht wie folgt aus:

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
  }
});
```

Beachten Sie, wie schnell die Eingabe aktualisiert wird.

Wie gehe ich mit der Abwertung von $ dispatch und $ broadcast um? (Busereignismuster)

Möglicherweise haben Sie erkannt, dass `$emit` emit für die Komponente gilt, die das Ereignis auslöst. Dies ist ein Problem, wenn Sie in der Komponentenstruktur zwischen weit voneinander entfernten Komponenten kommunizieren möchten.

Hinweis: In Vue1 können Sie `$dispatch` oder `$broadcast` , nicht jedoch in Vue2. Der Grund ist, dass es nicht gut skaliert. Es ist ein beliebtes `bus` Muster , dies zu verwalten:

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
  }
});
```

Sie müssen nur verstehen , dass jede `Vue()` kann beispielsweise `$emit` und zu fangen (`$on`) ein Ereignis. Wir erklären , nur einen globalen `Vue` Instanz Anruf `bus` und dann jede Komponente mit diesen Variablen kann daraus emittieren und fangen Ereignisse. So stellen Sie sicher , dass die Komponente Anbindung an die Autobahnen `bus` variabel.

Veranstaltungen online lesen: https://riptutorial.com/de/vue-js/topic/5941/veranstaltungen

Kapitel 23: Verwenden Sie "this" in Vue

Einführung

Einer der häufigsten Fehler , die wir in Vue Code auf Stackoverflow zu finden , ist der Missbrauch `this` . Die häufigsten Fehler betreffen im Allgemeinen zwei Bereiche: Sie verwenden `this` in Rückrufen für Versprechen oder andere asynchrone Funktionen und verwenden Pfeilfunktionen, um Methoden, berechnete Eigenschaften usw. zu definieren.

Examples

FALSCH! Verwenden Sie "this" in einem Rückruf in einer Vue-Methode.

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

FALSCH! Verwenden Sie "dies" in einem Versprechen.

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

RECHT! Verwenden Sie einen Verschluss, um "dies" zu erfassen

Sie können die korrekte Erfassung `this` unter Verwendung eines Verschlusses .

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

RECHT! Verwenden Sie bind.

Sie können die Rückruffunktion binden .

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

RECHT! Verwenden Sie eine Pfeilfunktion.

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

Vorsicht! Pfeil - Funktionen sind eine Syntax eingeführt in EcmaScript 2015. Es wird noch nicht unterstützt , aber *alle* modernen Browser, so dass es nur verwenden , wenn Sie Targeting ein Browser , den Sie *wissen* , unterstützt, oder wenn Sie Ihren Javascript kompilieren bis ES5 Syntax mit so etwas wie babel .

FALSCH! Verwenden einer Pfeilfunktion zum Definieren einer Methode, die auf "this" verweist

```
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"
  }
})
```

RECHT! Definieren Sie Methoden mit der typischen Funktionssyntax

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

Alternativ, wenn Sie einen Javascript-Compiler oder einen Browser verwenden, der Ecmascript 2015 unterstützt

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

Verwenden Sie "this" in Vue online lesen: https://riptutorial.com/de/vue-js/topic/9350/verwenden-sie--this--in-vue

Kapitel 24: VueJS + Redux mit Vua-Redux (beste Lösung)

Examples

Wie benutze ich Vua-Redux?

Vua Redux von NPM installieren:

Installiere durch:

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

Initialisieren:

==========

// 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;
```

Verwenden Sie in Ihrer Komponente:

// 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);
```

VueJS + Redux mit Vua-Redux (beste Lösung) online lesen: https://riptutorial.com/de/vue-js/topic/7396/vuejs-plus-redux-mit-vua-redux--beste-losung-

Kapitel 25: vue-router

Einführung

vue-router ist die offiziell unterstützte Routing-Bibliothek für vue.js.

Syntax

- `<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

Grundlegendes Routing

Die einfachste Möglichkeit, den vue-router einzurichten, ist die Verwendung der über CDN bereitgestellten Version.

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')
```

vue-router online lesen: https://riptutorial.com/de/vue-js/topic/9654/vue-router

Kapitel 26: Vuex

Einführung

Vuex ist eine Zustandsverwaltungsstruktur + Bibliothek für Vue.js-Anwendungen. Es dient als zentraler Speicher für alle Komponenten in einer Anwendung. Regeln sorgen dafür, dass der Zustand nur in vorhersehbarer Weise mutiert werden kann. Es ist auch in die Erweiterung der offiziellen Entwicklertools von Vue integriert und bietet erweiterte Funktionen wie Null-Konfigurations-Debugging für Zeitreisen und Export / Import von Status-Snapshots.

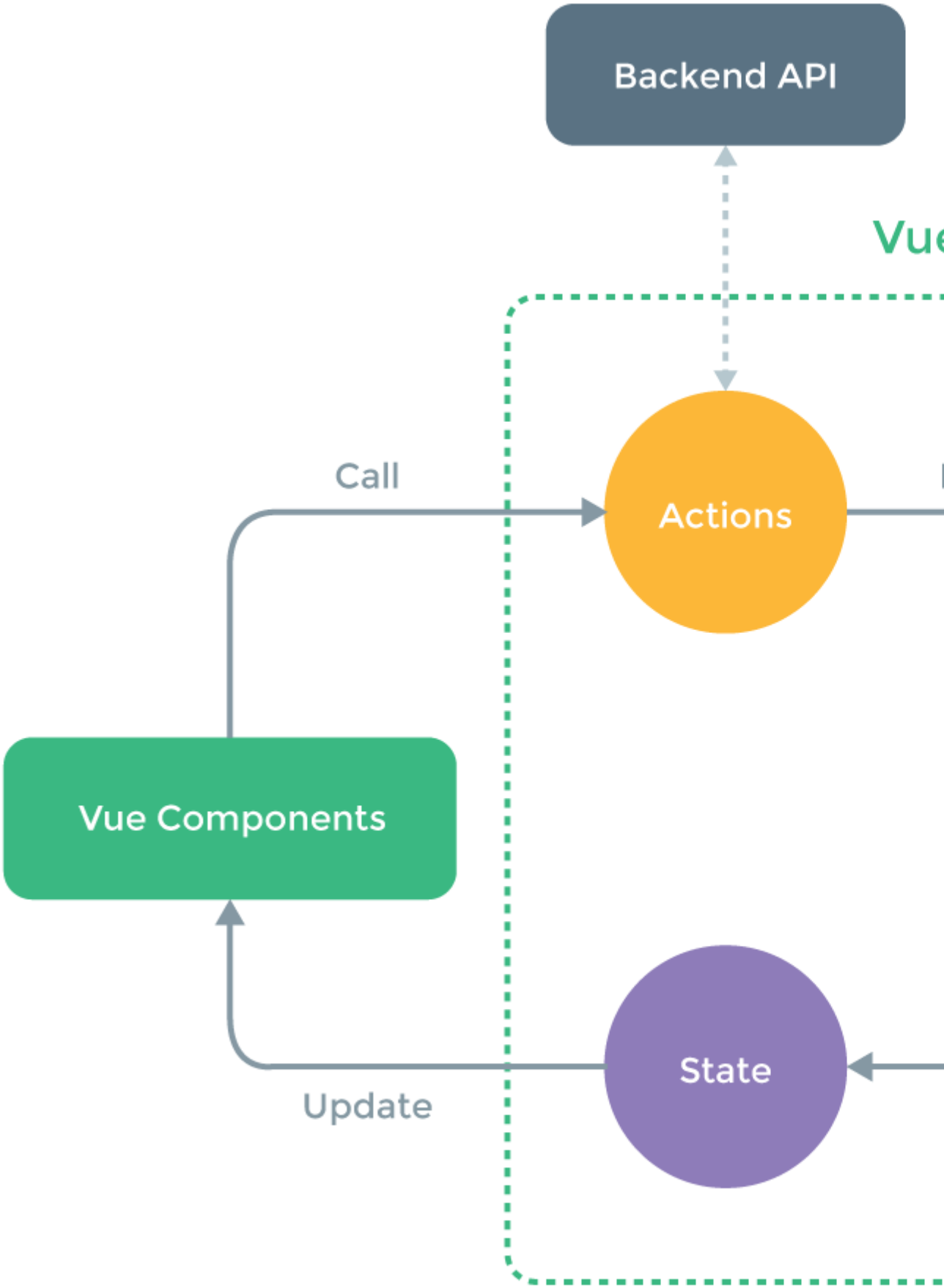
Examples

Was ist Vuex?

Vuex ist ein offizielles Plugin für Vue.js, das einen zentralisierten Datastore zur Verwendung in Ihrer Anwendung bietet. Sie wird stark von der Flux-Anwendungsarchitektur beeinflusst, die einen unidirektionalen Datenfluss bietet, was zu einem einfacheren Anwendungsdesign und Argumentation führt.

In einer Vuex-Anwendung enthält der Datastore den gesamten Status der gemeinsam **genutzten Anwendung** . Dieser Zustand wird durch **Mutationen** geändert, die als Reaktion auf eine **Aktion ausgeführt werden,** die ein Mutationsereignis über den **Dispatcher** aufruft.

Ein Beispiel für den Datenfluss in einer Vuex-Anwendung ist in der folgenden Abbildung dargestellt.


Backend API
Vue
Call
Actions
Vue Components
State
Update

Diagramm, das unter der MIT- Lizenz verwendet wird, ursprünglich aus dem offiziellen Vuex GitHub-Repo .

Einzelne Vue.js-Anwendungskomponenten können auf das Speicherobjekt zugreifen, um Daten über **Getter** abzurufen, bei denen es sich um reine Funktionen handelt, die eine Nur-Lese-Kopie der gewünschten Daten zurückgeben.

Die Komponenten können **Aktionen** haben , die Funktionen sind , die Änderungen an der Komponente eigene Kopie der Daten durchführen, den **Dispatcher** verwenden , um ein Mutationsereignis zu versenden. Dieses Ereignis wird dann vom Datastore verarbeitet, der den Status bei Bedarf aktualisiert.

Änderungen werden dann automatisch in der gesamten Anwendung übernommen, da alle Komponenten über ihre Getter reaktiv an das Geschäft gebunden werden.

Ein Beispiel, das die Verwendung von Vuex in einem Vue-Projekt veranschaulicht.

```
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())
    }
  }
})
```

Und die HTML-Vorlage für das gleiche:

```
<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. Hier enthält der **Zustand** die **lastClickTime-** Eigenschaft, die als null initialisiert wurde. Diese Einrichtung von Standardwerten ist wichtig , die Eigenschaften **reaktiv** zu halten. **Eigenschaften, die im Zustand nicht erwähnt** werden, sind verfügbar, aber die danach vorgenommenen Änderungen sind **nicht** mit Hilfe von Gettern **zugänglich** .

2. Der verwendete Getter stellt eine berechnete Eigenschaft bereit, die jedes Mal aktualisiert wird, wenn eine Mutation den Wert der Eigenschaft state aktualisiert.

3. **Nur Mutationen** dürfen den Zustand und seine Eigenschaften ändern, das heißt, dies geschieht **nur synchron** .

4. Eine Aktion kann für asynchrone Aktualisierungen verwendet werden, bei denen der API-Aufruf (der hier durch den zufallsgesteuerten setTimeout verspottet wird) in der Aktion ausgeführt werden kann, und nachdem die Antwort abgerufen wurde, eine Mutation ausgeführt werden kann, um die Änderung des Status vorzunehmen .

Warum Vuex verwenden?

Wenn Sie große Anwendungen wie Single-Page-Apps (SPAs) erstellen, die in der Regel aus vielen wiederverwendbaren Komponenten bestehen, können sie sehr schwer zu erstellen und zu warten sein. Die gemeinsame Nutzung von Daten und Zuständen zwischen diesen Komponenten kann auch schnell brechen und das Debugging und die Wartung schwierig werden.

Durch die Verwendung eines zentralen Anwendungsdatenspeichers kann der gesamte Anwendungsstatus an einer Stelle dargestellt werden, wodurch die Anwendung organisierter wird.

Durch die Verwendung eines unidirektionalen Datenflusses, durch Mutationen und durch das Festlegen des Zugriffs auf Komponentendaten auf die erforderlichen Daten wird es viel einfacher, die Komponentenrolle und die Auswirkungen auf den Anwendungsstatus zu beurteilen.

VueJS-Komponenten sind separate Entitäten und können Daten nicht einfach untereinander austauschen. Um Daten ohne Vuex gemeinsam zu nutzen, müssen wir ein Ereignis mit Daten `emit` und dann das Ereignis mit `on` überwachen.

Komponente 1

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

Komponente 2

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

Wenn vuex installiert ist, können wir einfach von jeder Komponente aus auf die Daten zugreifen, ohne dass Ereignisse abgehört werden müssen.

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

Wir können Daten auch synchron mit *Mutatoren ändern* , asynchrone *Aktionen verwenden* und Daten mit *Getter*- Funktionen *abrufen* .

Getterfunktionen funktionieren möglicherweise als global berechnete Funktionen. Wir können auf sie von Komponenten zugreifen:

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

Aktionen sind globale Methoden. Wir können sie aus Komponenten versenden:

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

Mutationen sind die einzige Möglichkeit, Daten in vuex zu ändern. Wir können Änderungen festschreiben:

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

Vuex-Code würde so aussehen

```
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
  }
}
```

Wie installiere ich Vuex?

In den meisten Fällen verwenden Sie Vuex in größeren komponentenbasierten Anwendungen, in denen Sie wahrscheinlich einen Modul-Bundler wie Webpack oder Browserify in Verbindung mit Vueify verwenden, wenn Sie einzelne Dateien verwenden.

In diesem Fall ist der einfachste Weg, um Vuex zu erhalten, über NPM. Führen Sie den folgenden Befehl aus, um Vuex zu installieren, und speichern Sie es in Ihren Anwendungsabhängigkeiten.

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

Stellen Sie sicher, dass Sie Vuex mit Ihrem Vue-Setup verknüpfen, indem Sie die folgende Zeile nach der Anweisung von Request `require('vue')` einfügen.

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

Vuex ist auch als CDN erhältlich. Sie können die neueste Version von cdnjs hier herunterladen.

Automatisch unzulässige Benachrichtigungen

In diesem Beispiel wird ein Vuex-Modul dynamisch registriert, um benutzerdefinierte Benachrichtigungen zu speichern, die automatisch verworfen werden können

notifications.js

vuex store auflösen und einige Konstanten definieren

```
//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';
```

Setzen Sie unser Modul in den Ausgangszustand

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

Setze unsere Modul-Getter

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

Setzen Sie unser Modul Aktionen

```
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);
  }
}
```

Setze unsere Modulmutationen

```
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);
  }
}
```

Registrieren Sie unser Modul mit definiertem Zustand, Eingaben, Aktionen und Mutationen

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

Verwendungszweck

componentA.vue

Diese Komponente zeigt alle Benachrichtigungen als Warnmeldungen des Bootstraps in der oberen rechten Ecke des Bildschirms an. Außerdem können Sie jede Benachrichtigung manuell abbrechen.

```
<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>
```

Snippet zum Hinzufügen einer Benachrichtigung in einer anderen Komponente

```
//payload could be anything, this example content matches with componentA.vue
this.$store.dispatch('Add', {
  title = 'Hello',
  text = 'World',
  type = 'info',
  duration = 15000
});
```

Vuex online lesen: https://riptutorial.com/de/vue-js/topic/3430/vuex

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit 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	Bedingtes Rendern	jaredsk, m_callens, Nirazul, user6939352
3	Benutzerdefinierte Filter	Finrod, M U, m_callens
4	Benutzerdefinierte Richtlinien	Mat J, Ogie Sado
5	Beobachter	El_Matella
6	Berechnete Eigenschaften	Amresh Venugopal, cl3m, jaredsk, m_callens, Theo, Yerko Palma
7	Datenbindung	gurghet, Jilson Thomas
8	Die Array-Änderungswarnung ist zu beachten	Vamsi Krishna
9	Dynamische Komponenten	Med, Ru Chern Chong
10	Einzelne Dateikomponenten	jordiburgos, Ru Chern Chong
11	Ereignisbus	Amresh Venugopal
12	Komponenten	Donkarnash, Elfayer, Hector Lorenzo, Jeff, m_callens, phaberest, RedRiderX, user6939352
13	Kundenspezifische Komponenten mit V-Modell	Amresh Venugopal
14	Lebenszyklus-Haken	Linus Borg, m_callens, PatrickSteele, xtreak
15	Listen-Rendering	chuanxd, gurghet, Mahmoud, Theo

16	Mixins	Ogie Sado
17	Modifikatoren	sept08
18	Plugins	AldoRomo88
19	Polyfill "Webpack" Vorlage	Stefano Nepa
20	Requisiten	asemahle, Donkarnash, FlatLander, m_callens, rap-2-h, Shuvo Habib
21	Schlüssel	Daniel Waghorn, Elfayer, Shuvo Habib, Slava
22	Veranstaltungen	Elfayer
23	Verwenden Sie "this" in Vue	Bert
24	VueJS + Redux mit Vua-Redux (beste Lösung)	Aniko Litvanyi, FlatLander, Shuvo Habib, Stefano Nepa
25	vue-router	AJ Gregory
26	Vuex	AldoRomo88, Amresh Venugopal, Daniel Waghorn, Matej Vrzala M4, Ru Chern Chong