KAPITEL 1

Einführung

Vue.js konzentriert sich auf die Gestaltung von Benutzeroberflächen. Dabei wird ganz bewusst kein umfassendes Gesamtpaket angeboten. Stattdessen beschränkt sich der Kern der Vue-Bibliothek bewusst auf den View-Layer. Daher stammt auch der Name, der (im Englischen »vju:« ausgesprochen) ganz ähnlich klingt wie »View«. Durch diesen schlanken Ansatz kann Vue auch in bestehende Projekte leicht integriert werden.

Weitere Funktionalitäten können nach Wunsch durch die inkrementelle Hinzugabe weiterer Bibliotheken und Komponenten sowie die Nutzung moderner Tools ergänzt werden. Für den Anfang genügt jedoch die Referenzierung von Vue über einen CDN-Anbieter wie *jsDelivr*.

Bei der in den Codebeispielen referenzierten Version von Vue handelt es sich um einen Entwickler-Build. Er ist nicht so performant wie die Version für den produktiven Einsatz, enthält aber umfassendere Fehler- und Warnmeldungen für die Konsole. Für den Einsatz in produktiven Umgebungen sollte ein minifizierter Produktiv-Build von Vue (beispielsweise via *https://cdn.jsdelivr.net/npm/vue/dist/vue.min.js*) referenziert werden.

Sehen wir uns nun einfach ein erstes Beispiel an:

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

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

 <script>
  var app = new Vue({
   el: '#app',
   data: {
    message: 'Hello Vue!'
   }
  })
 </script>
</body>
<html>
```

Dieser Code erzeugt im Browser die folgende Ausgabe:

```
Hello Vue!
```

Hiermit haben wir bereits eine erste Vue-Anwendung geschrieben. Was im ersten Moment lediglich wie das Rendern eines Strings aussieht, ist in Wirklichkeit viel mehr. Das DOM der HTML-Seite und die Daten von Vue sind nun miteinander verknüpft. Durch diese Abhängigkeit werden Änderungen an den Daten sofort im Browser sichtbar. Dieses deklarative Rendering ist eine der Kernfunktionen von Vue.

Speichern Sie den obigen Code also einfach in einer Datei mit der Endung *.html*, öffnen Sie diese Datei in Ihrem Browser und starten Sie anschließend die JavaScript-Konsole (in Chrome ein Rechtsklick auf eine beliebige Stelle und *Untersuchen* wählen). Wenn Sie nun den Befehl **`app.message = "Hello again!"`** in der Konsole eingeben, erscheint der neue Text direkt im Fenster des Browsers.

KAPITEL 2

Die Vue-Instanz

Die *Root-Instanz* ist das Herzstück jeder Vue-Anwendung. Im Beispiel aus dem vorherigen Kapitel stellt sie etwa das Datenattribut `message` bereit, dessen Wert mit der Template-Syntax im DOM gerendert wird. Eine typische Vue-Anwendung beginnt immer mit einer Vue-Instanz, die via `new Vue` erstellt wird.

```
var vm = new Vue({
 ...
})
```

Vue ist teilweise vom MVVM-Pattern (*Model-View-ViewModel*) inspiriert, weshalb für die Instanziierung oft der Variablenname `vm` (für Viewmodel) genutzt wird.

Bei der Instanziierung wird immer ein Objekt mit Optionen übergeben. Einen Großteil dieser Optionen werden wir in diesem Teil des Buchs besprechen. Eine komplette Auflistung findet sich in Teil III, »Vue-API«.

Von der Root-Instanz ausgehend kann sich eine Baumstruktur ergeben, die sich aus verschachtelten HTML-Elementen und weiteren Komponenten von Vue zusammensetzt. Diese wiederverwendbaren *Components* werden im zweiten Teil dieses Buchs besprochen. Vorab sei jedoch gesagt: Auch Components sind Vue-Instanzen und arbeiten bis auf einige wenige Ausnahmen mit dem gleichen Optionenobjekt wie die Root-Instanz der Vue-Anwendung.

Daten und Funktionen

Immer wenn eine neue Vue-Instanz erzeugt wird, werden zunächst alle Attribute aus dem Objekt `data` des übergebenen Optionenobjekts in das Reaktivsystem von Vue übertragen. Ändert sich später eines dieser Attribute, wird auch der View-Layer aktualisiert.

```
<div id="app">
 {{ firstName }} {{ lastName }} shot first!
</div>

var vm = new Vue({
 el: '#app',
 data: {
  firstName: "Han",
  lastName "Solo"
 }
})
```

Ändern sich in diesem Beispiel die Werte von `firstName` oder `lastName`, wird auch der View-Layer aktualisiert. Dabei ist es wichtig, zu verstehen, dass diese reaktive Funktionalität nur für die Attribute des Objekts `data` gilt, die bereits bei der Instanziierung existieren.

Die nachträgliche Zuweisung von **`vm.sidekick = "Chewbacca"`** über die JavaScript-Konsole des Browsers würde zwar das neue Attribut anlegen, Änderungen an `sidekick` hätten aber keine automatische Aktualisierung des Views zur Folge.

Falls die Funktionalität einer Vue-Anwendung also bestimmte Attribute voraussetzt, die aber erst später mit tatsächlichen Werten gefüllt werden, sollten diese Attribute mit entsprechenden Initialwerten direkt bei der Erstellung der Vue-Instanz definiert werden:

```
data: {
 message: '',
 maxTries: 0,
 showToolbar: false,
 users: [],
 error: null
}
```

Auf alle Attribute aus dem Objekt `data` kann über die Instanzvariable `vm` zugegriffen werden, etwa `vm.firstName`. Darüber hinaus bie-

tet sie Zugriff auf einige besondere Attribute und Funktionen der Instanz. Diese verwenden das Dollarzeichen `$` als Präfix und werden im dritten Teil dieses Buchs erläutert.

Lifecycle-Hooks

Jede Vue-Instanz durchläuft bei ihrer Initialisierung mehrere Schritte, etwa die Verarbeitung der Datenattribute, die Kompilierung des Templates oder die Verknüpfung mit dem DOM. Dabei ruft Vue eine Reihe von Funktionen auf, die sogenannten *Lifecycle-Hooks*. Sie ermöglichen es, die einzelnen Schritte während der Instanziierung mit eigenem Code zu erweitern.

```
var vm = new Vue({
 data: {
  foo: 'bar'
 },
 created: function () {
  console.log('foo is: ' + this.foo)
 }
})
```

In diesem Beispiel wird der Text `foo is bar` in der Konsole des Browsers ausgegeben, sobald die Vue-Instanz erzeugt wurde.

Neben `created` gibt es noch einige weitere Hooks wie `mounted`, `updated` und `destroyed`, die an unterschiedlichen Punkten während des Lebenszyklus der Vue-Instanz aufgerufen werden. Beim Aufruf dieser Funktionen verweist der Kontext für die Variable `this` jeweils auf die aufrufende Vue-Instanz.

Falls Sie mit der aktuellen ES7-Syntax arbeiten, sollten Sie bei der Definition von Lifecycle-Hooks unbedingt auf die Verwendung der sogenannten Arrow Functions verzichten. Da *Arrow Functions* immer die nächsthöhere Hierarchieebene als Kontext definieren, zeigt die Variable `this` nicht wie erwartet auf die Vue-Instanz. Dies führt zu Fehlern wie `Uncaught TypeError: Cannot read property of undefined`.

KAPITEL 3

Templates

Das Objekt `data` verwaltet die Daten der Vue-Instanz. Aber wie werden diese Daten an der richtigen Stelle im Browser dargestellt? Hierfür benutzt Vue eine spezielle Template-Syntax, die auf HTML basiert. Mit ihr können einzelne Datenattribute deklarativ an das DOM gebunden werden.

Wie genau das Template für eine Vue-Instanz definiert wird, haben wir bereits im ersten Kapitel dieses Buchs gesehen. Die für das Template relevanten Angaben sind hier fett markiert:

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

<script>
 var vm = new Vue({
  el: '#app',
  data: {
   myMessage: 'Hello Vue!'
  }
 })
</script>
```

Innerhalb des HTML der Anwendung wurde eine eindeutige ID – hier `"app"` – für ein HTML-Element festgelegt. Gleichzeitig wird diese ID im Attribut `el` des Optionenobjekts der Vue-Instanz referenziert. In diesem Codebeispiel wird nun der Div-Container mit der ID `"app"` mit allen untergeordneten HTML-Elementen als Template für die Vue-Instanz verwendet.

Interpolationen

Die simpelste und gebräuchlichste Form für die Referenzierung von Daten ist die Textinterpolation mit der »Double-Mustache-Syntax« mit je zwei geschweiften öffnenden und schließenden Klammern {{ }}:

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

Das Mustache-Tag wird beim Rendering durch den tatsächlichen Wert des Datenattributs `myMessage` ersetzt. Eine Aktualisierung erfolgt, sobald sich der Wert von `message` ändert.

Daten, die über die Double-Mustache-Syntax in das Template eingefügt werden, werden immer als Text interpretiert. Um richtiges HTML auszugeben, dient die sogenannte Direktive `v-html`:

```
<div id="app">
 <span v-html="rawHtml"></span>
</div>
```

Durch die Nutzung der Direktive `v-html` wird der Inhalt des Elements `span` durch den Wert des Datenattributs `rawHtml` ersetzt und dabei als HTML interpretiert.

Die Double-Mustache-Syntax {{ }} wird für HTML-Attribute nicht unterstützt. Stattdessen ist hier die Direktive `v-bind` zu verwenden:

```
<a v-bind:href="targetLink">Click here</a>
```

Bei booleschen HTML-Attributen wie `selected` oder `disabled`, die durch ihre bloße Existenz den Wert `true` implizieren, funktioniert `v-bind` (genau wie bei den Attributen `class` und `style`, die wir später noch gesondert betrachten werden) etwas anders:

```
<button v-bind:disabled="isDisabled">Submit</button>
```

Ist der Wert des Datenattributs `isDisabled` entweder `null`, `undefined` oder `false`, wird das HTML-Attribut `disabled` nicht an das gerenderte Element `<button>` angefügt.

Bei allen obigen Beispielen wurden immer nur direkte Werte aus dem Datenobjekt der Vue-Instanz über den Namen ihres jeweiligen Attributs in das Template eingefügt. Vue unterstützt bei der Nutzung von {{ }} oder `v-bind` aber auch JavaScript-Ausdrücke:

```
{{ number + 1 }}

{{ hasPermission ? 'Yes' : 'No' }}

{{ myMessage.toLowerCase().split('').join(' ') }}

<div v-bind:id="'list-' + id"></div>
```

Alle Ausdrücke werden als JavaScript interpretiert. Dabei bezieht sich der Kontext immer auf die dem Template zugeordnete Vue-Instanz. Jede Interpolation darf allerdings immer nur einen Ausdruck enthalten.

Alle Ausdrücke in Templates werden gekapselt. Daher haben sie bei ihrer Auswertung nur auf einige ausgewählte globale Klassen wie etwa `Math` oder `Date` Zugriff. Falls Sie in Ihrem Projekt beispielsweise eigene globale Variablen definiert haben, können diese innerhalb solcher Ausdrücke nicht verwendet werden.

Direktiven

Direktiven sind Attribute für HTML-Elemente, die mit dem Präfix `v-` versehen sind und von Vue verwendet werden. Zwei dieser Direktiven haben wir mit `v-bind` und `v-html` bereits kennengelernt.

Mit Ausnahme der Direktive `v-for` sollten alle Direktiven einen einzelnen JavaScript-Ausdruck beinhalten. Die Aufgabe einer jeden Direktive ist die Anwendung bestimmter Seiteneffekte auf das DOM, sobald sich die in den jeweiligen Ausdrücken verwendeten Werte ändern. Erinnern wir uns an dieses Beispiel:

```
<span v-html="myRawHtml"></span>
```

Hier wird die Direktive `v-html` genutzt, um das Datenattribut `myRawHtml` als HTML in das Template einzufügen. Sollte sich der Wert von `myRawHtml` ändern, würde `v-html` automatisch den HTML-Inhalt von `<span>` aktualisieren.

Argumente

Einige Direktiven können Argumente erhalten, die über einen Doppelpunkt vom Namen der Direktive getrennt werden. Mit `v-bind` haben wir bereits solch eine Direktive kennengelernt:

```
<a v-bind:href="url">Click me!</a>
```

Hier wird die Direktive `v-bind` mit dem Argument `href` verwendet, um das Ergebnis des auszuwertenden Ausdrucks (das ist hier der Wert des Datenattributs `url`) an das HTML-Attribut `href` für das Anchor-Tag zu knüpfen. Ein anderes Beispiel für eine Direktive mit Argumenten ist `v-on`. Mit ihr können Listener für DOM-Events registriert werden:

```
<a v-on:click="doSomething">Click me!</a>
```

Das Thema Events wird in Kapitel 8 ausführlich behandelt.

Modifizierer

Modifizierer sind Postfixes für Direktiven, die über einen Punkt vom Namen der Direktive getrennt werden. Sie können als eine Art Optionsparameter verstanden werden, die das Standardverhalten einer Direktive beeinflussen.

Beispielsweise kann der Modifizierer `.prevent` genutzt werden, um der Direktive `v-on` mitzuteilen, dass vor der Ausführung des Callbacks für das soeben ausgelöste Event `event.preventDefault()` ausgeführt werden soll:

```
<form v-on:submit.prevent="onSubmit">
 ...
</form>
```

Shorthands

Durch das Präfix `v-` lassen sich Direktiven leicht von normalen HTML-Attributen unterscheiden. Für besonders häufig genutzte Direktiven kann das auf Dauer aber etwas nervig werden, deshalb stellt Vue für `v-bind` und `v-on` Kurzschreibweisen bereit.

Diese Kürzel sehen auf den ersten Blick zwar etwas ungewöhnlich aus, tatsächlich sind die für die Kurzschreibweisen genutzten Zeichen aber valide Zeichen für die Benennung von HTML-Attributen und sollten daher in keinem Browser Probleme verursachen.

Die Kurzschreibweise von `v-bind` verwendet den Doppelpunkt:

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

<!-- Shorthand -->
<a :href="url"> ... </a>
```

Für die Kurzschreibweise von `v-on` wird das `@`-Zeichen genutzt:

```
<!-- Vollständige Schreibweise -->
<a v-on:click="doSomething"> ... </a>

<!-- Shorthand -->
<a @click="doSomething"> ... </a>
```