

JavaScript: Einführung in React

Werkzeuge 12.06.2015 10:45 Uhr – Roberto Bez – 0 Kommentare

React ist ein von Facebook entwickeltes JavaScript-Framework, das von einer ständig wachsenden Community gepflegt wird. Wegen der Möglichkeit, React Views sowohl auf dem Client, als auch auf dem Server zu rendern, erfreut sich das Framework bei vielen Entwicklern großer Beliebtheit.

Obwohl bereits andere Frameworks den Weg eines isomorphen Renderings (Client/Server) gegangen sind, unterscheidet sich React von ihnen in einer grundlegenden Funktion: dem virtuellen Document Object Model (DOM). Die Views modifizieren nicht direkt den DOM des Browsers,


sondern halten ihn virtuell in den Komponenten vor und sind damit komplett von irgendwelchen Abhängigkeiten losgelöst. Mit Methoden wie `renderToString(<React View`

INHALTSVERZEICHNIS

1. JavaScript: Einführung in React
 2. Aufbau, Syntax und Eigenschaften
 3. Objektspezifikation, Events und Datenfluss
 4. React und Flux
 5. Aktionen und Fazit
- » [Auf einer Seite lesen](#)

Anzeige





Component>) lassen sich die React-View-Komponenten auf dem Server rendern.

Dass React funktioniert, haben einige große Unternehmen bereits gezeigt: Nicht nur Teile von Facebook wurden damit entwickelt, sondern auch Plattformen wie Instagram oder die Webversion von WhatsApp nutzen das Framework. Gerade für Seiten, die in Suchmaschinen eine Rolle spielen sollen, ist Search Engine Optimization, kurz SEO, ein KO-Kriterium. Daher ist zu beachten, dass das serverseitige Rendern von Webanwendungen, die mit Frameworks wie AngularJS entwickelt wurden, sehr aufwendig ist. Beispielsweise ist das Erstellen von Snapshots des HTML-Codes mit PhantomJS bei dynamischen Seiten nicht gerade einfach – und auch nicht besonders performant.

In React hingegen erzeugen alle Komponenten ihr eigenes virtuelles DOM. Dadurch lassen sie sich nicht nur auf dem Server rendern, sondern sie erleichtern auch das Unit-Testing. Wenn sich Komponenten beispielsweise durch Benutzerinteraktionen oder durch neue Antworten vom Server ändern, versucht das virtuelle DOM nicht die ganze Anwendung, sondern nur das kleinste betroffene Element neu zu rendern.

Ein Beispiel kann den Performance-Gewinn deutlicher erklären: Angenommen es gibt ein Model (siehe Model View Controller), das etwa ein Bürogebäude abbilden soll. Alle relevanten Eigenschaften des Gebäudes (Räume, Fenster ...) werden auf den aktuellen *State* des Objektes gespiegelt. Ein derartiges Verhalten legt React mit dem DOM an den Tag.

Werden im Gebäude mehrere Räume zugunsten eines Großraumbüros zusammengelegt, würde das Framework wie folgt damit umgehen: Erst identifiziert es alle stattfindenden Änderungen (Reconciliation) und anschließend aktualisiert es das DOM entsprechend. Wichtig ist, dass React kein neues Gebäude bauen, sondern nur die betroffenen Räume neu gestalten würde. Wird also ein DOM-Element verändert, sein Elternelement aber nicht, ist letzteres nicht betroffen.

React installieren

Das [React Starter Kit](#) gibt es als Download oder als JSFiddle Snipped zum direkten Loslegen. Clientseitig reicht es, die *react.js*-Datei einzubinden. Optional lässt sich noch *JSXTransformer.js* hinzufügen, um die in JavaScript XML (JSX) verfassten Komponenten parsen zu können (später dazu mehr). React-Code kann man anschließend in einem Script-Block mit *type="text/js"* entwickeln:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="build/react.js"></script>
    <script src="build/JSXTransformer.js"></script>
  </head>
  <body>
    <div id="example"></div>
    <script type="text/jsx">
      React.render(
        <h1>Hello, world!</h1>,
        document.getElementById('example')
      );
    </script>
  </body>
</html>
```

Der JSX-Code steht nicht im Code der Website, sondern ist in eine separate Datei ausgelagert, mit Tools wie Webpack oder Browserify zusammengestellt und einmalig eingebunden.

Ist Webpack und das dazugehörige npm-Modul *jsx-loader* installiert, reicht die folgende Datei *webpack.config.js* aus, um direkt loszulegen zu können:

```
'use strict';

var path = require('path'),
```

```
targetPath = path.join(__dirname, 'build', 'assets'),
config;

config = {
  resolve: {
    extensions: [ '', '.js', '.jsx' ]
  },
  entry: [
    './site.jsx'
  ],
  output: {
    path: targetPath,
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      { test: /\.jsx$/, loader: 'jsx-loader' },
    ]
  }
};

module.exports = config;
```

Das Bundle ist vor dem schließenden Body-Tag der HTML-Seite einzubinden:

```
...
<script type="text/javascript" src="build/assets/bundle.js"></scrip
</body>
```

Vorherige

1

2

3

4

5

Nächste

Kommentieren

MEHR ZUM THEMA

JAVASCRIPT REACT

TEILE DIESEN BEITRAG



Forum bei heise online:

JavaScript

<https://heise.de/-2689175>

Drucken

Anzeige

Webcast: So wird Ihr RZ zukunftssicher!

In 5 Schritten zur modernen IT-Infrastruktur

Besserer Schutz mit Multi-Faktor-Authentifizierung

So finden Sie das passende Notebook!

Sind Ihre Filialen Einfallstore für Cyberangriffe?

Wie Corona die Zukunft der Arbeit gestaltet

Corona & Phishing: Bieten Sie Hackern die Stirn!

Zero Trust: Null Vertrauen, aber voll zufrieden

Container- und Serverless-Umgebungen absichern

Risk-based Vulnerability Management – jetzt!

Anzeige

News

7-Tage-News

News-Archiv

Rubriken

Sprachen

Architektur/Methoden

Werkzeuge

Know-how

Standards

Literatur

Videos

Veranstaltungsberichte

Blogs

Der Dotnet-Doktor

the next big thing

Neuigkeiten von der Insel

Tales from the Web side

Continuous Architecture

Der Pragmatische Architekt

ÜberKreuz

Modernes C++

Podcasts

Mein Scrum ist kaputt

SoftwareArchitekTOUR

Techtiefen

Videos

Konferenzen

Termine

colspan

"Ich roll' dann mal aus"