

NILS HARTMANN
<https://nilshartmann.net>

2019

Moderne React

Slides: <https://bit.ly/md-dev-days-react-2019>

NILS HARTMANN

<https://nilshartmann.net>

Freiberuflischer Entwickler, Architekt, Trainer aus Hamburg

Java
JavaScript, TypeScript
React
GraphQL

nils@nilshartmann.net

React 16.x

Blick voraus

We plan to split the rollout of new React features into the following milestones:

- React 16.6 with Suspense for Code Splitting (*already shipped*)
- A minor 16.x release with React Hooks (~Q1 2019) ----- **16.8 (aktuelle Version)**
- A minor 16.x release with Concurrent Mode (~Q2 2019)
- A minor 16.x release with Suspense for Data Fetching (~mid 2019)

<https://reactjs.org/blog/2018/11/27/react-16-roadmap.html>

Weiterhin nur Minor-Versionen (!)

The image displays two side-by-side screenshots of a web application titled "React Chat Example".

Left Screenshot (React 16.6.0):

- Header:** "React Chat Example" and "16.6.0".
- Navigation:** "In the Office..." (highlighted), "Philosophy", "Coffee".
- Content Area:**
 - Anonymous-620:** "I also believe it's important for every member to be involved and invested in our company and this is one way to do so. Curate."
 - Harry:** "Pushback."
 - Peter:** "Guerrilla marketing we don't want to boil the ocean we need to leverage our synergies touch base"
 - Maja:** "The sprint is over please use "solutionise" instead of solution ideas! ;).
 - Sue:** "Push back digitalize yet enough to wash your face, or low-hanging fruit horsehead offer, for Bob called an all-hands this afternoon that ipo will be a game-changer."
 - Anonymous-620 joined in the Office...**
 - Anonymous-621 joined in the Office...**
 - Anonymous-617 left in the Office...**
 - Klaus:** "Anonymous-621 logged in as Klaus"
- Bottom:** "Please login to post messages" and a red "Login" button.
- Footer:** "https://github.com/nilshartmann/react-chat-example"

Right Screenshot (React 16.7.0-alpha.0):

- Header:** "React Chat Example" and "React 16.7.0-alpha.0".
- Navigation:** "In the Office..." (highlighted), "Philosophy", "Coffee".
- Content Area:**
 - Anonymous-620:** "I also believe it's important for every member to be involved and invested in our company and this is one way to do so. Curate."
 - Harry:** "Pushback."
 - Peter:** "Guerrilla marketing we don't want to boil the ocean we need to leverage our synergies touch base"
 - Maja:** "The sprint is over please use "solutionise" instead of solution ideas! ;).
 - Sue:** "Push back digitalize yet enough to wash your face, or low-hanging fruit horsehead offer, for Bob called an all-hands this afternoon that ipo will be a game-changer."
 - Anonymous-620 joined in the Office...**
 - Anonymous-621 joined in the Office...**
 - Anonymous-617 left in the Office...**
 - Klaus:** "You" and "Anonymous-621 logged in as Klaus"
- Bottom:** "Add Message" input field and a red "Send" button, along with three red buttons labeled "Dashboard (Effects)", "Dashboard (Suspense)", and "Exit".
- Footer:** "https://github.com/nilshartmann/react-chat-example"

<https://github.com/nilshartmann/react-chat-example>

EIN BEISPIEL...

16.8

Hooks

FUNCTIONS EVERYWHERE

HINTERGRUND

Hooks: State, Context etc auch in Funktionskomponenten

Motivation:

- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
 - Durch Concurrent Rendering noch problematischer

HINTERGRUND

Hooks: State, Context etc auch in Funktionskomponenten

Motivation:

- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
 - Durch Concurrent Rendering noch problematischer

Hooks sind reguläre Funktionen

Hooks: State, Context etc auch in Funktionskomponenten

Motivation:

- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
 - Durch Concurrent Rendering noch problematischer

Hooks sind reguläre Funktionen, aber...

- **müssen** mit "use" beginnen
- **müssen** am Anfang einer Komponente stehen
- nur in Funktionskomponenten

USECONTEXT HOOK

useContext: Vereinfachter Zugriff auf den Context



You're logged in as **Maja**

User-Objekt liegt in einem
Context

USECONTEXT HOOK

useContext: Vereinfachter Zugriff auf den Context

- **Bisher**: Zugriff über Consumer-Komponente und Render Properties

```
function CurrentUserProfile(props) {  
  
  return (  
    <ChatContext.Consumer>  
      {chatValue => {  
        return <div>  
          <Avatar userId={chatValue.user.id} />  
          You're logged in as {chatValue.user.name} />  
        </div>;  
      }}  
    </ChatContext.Consumer>  
  );  
}
```

USECONTEXT HOOK

useContext: Vereinfachter Zugriff auf den Context

- **Bisher**: Zugriff über Consumer-Komponente und Render Properties
- Unübersichtlich bei mehreren Kontexten

```
function CurrentUserProfile(props) {  
  
  return (  
    <ChatContext.Consumer>  
      {chatValue => {  
        return <ThemeContext.Consumer>  
          { themeValue => {  
            return <div className={themeValue.name}>  
              <Avatar userId={chatValue.user.id} />  
              You're logged in as {chatValue.user.name} />  
            </div>;  
          }  
        }  
      }  
    </ThemeContext.Consumer>  
  }  
}</ChatContext.Consumer>  
);
```

USECONTEXT HOOK

useContext: Vereinfachter Zugriff auf den Context

- Hook: "normale" Funktion, Komponente wird gerendert, wenn Context sich ändert

```
import { useContext } from "react";

function CurrentUserProfile(props) {
  const chatValue = useContext(ChatContext);
  const themeValue = useContext(ThemeContext);

}
```

USECONTEXT HOOK

useContext: Vereinfachter Zugriff auf den Context

- Hook: "normale" Funktion, Komponente wird gerendert, wenn Context sich ändert

```
import { useContext } from "react";

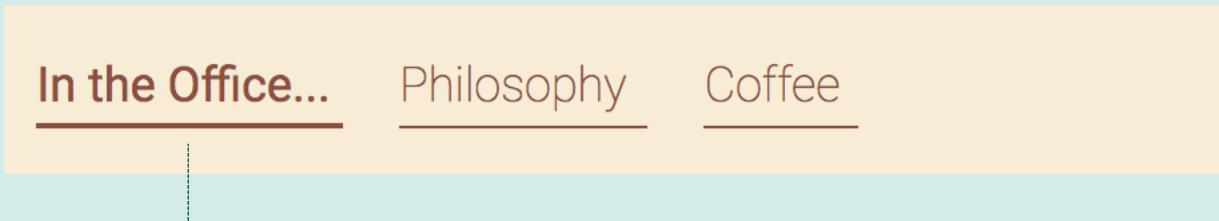
function CurrentUserProfile(props) {
  const chatValue = useContext(ChatContext);
  const themeValue = useContext(ThemeContext);

  return (
    <div className={themeValue.name}>
      <Avatar userId={chatValue.user.id} />
      You're logged in as {chatValue.user.name} />
    </div>
  );
}
```

USESTATE HOOK

useState: State in Funktionskomponenten

Beispiel: Tab Bar



Zustand: welche Tab ist geöffnet?

USESTATE HOOK

Vorher: Setzen von State in Funktionen nicht möglich

- Setzen und Lesen nicht einheitlich (this.setState vs this.state.x)
- setState "seltsame" Semantik
- this-Problematik

```
class Tabs extends React.Component {  
  state = { activeTabId: 0 }  
  render() {  
    return (  
      <div>  
        {props.tabs.map(tab => {  
          return <Tab  
            classname={tab.id === activeTabId ? "active" : ""}  
            onClick={() => this.setState({activeTabId: tab.id})}  
          />  
        })}  
      </div>);  
  }  
}
```

USESTATE HOOK

useState: State erzeugen

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
}  
  
|  
Aktueller State  
|  
Setter  
|  
initialer Wert  
|  
}  
}
```

USESTATE HOOK

useState: Aktuellen State verwenden

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
  
  return (  
    <div>  
      {props.tabs.map(tab => {  
        return <Tab  
          classname={tab.id === activeTabId ? "active" : ""}  
        />  
      })}  
    </div>  
  );  
}
```

Zugreifen auf State

USESTATE HOOK

useState: State verändern

```
function Tabs(props) {  
  const [activeTabId, setActiveTabId] = React.useState(0);  
  
  return (  
    <div>  
      {props.tabs.map(tab => {  
        return <Tab  
          classname={tab.id === activeTabId ? "active" : ""}  
          onClick={() => setActiveTabId(tab.id)}  
        />  
      })}  
    </div>  
  );  
}
```

**Setzen von State
(kein Objekt mehr!)**

USESTATE HOOK

useState: Mehrere States in einer Komponente möglich

- Kein "mergen" von State mehr!

```
function LoginForm(props) {  
  const [username, setUsername] = React.useState("klaus");  
  const [password, setPassword] = React.useState("");  
  
  return (<>  
    <input value={username}  
          onChange={e => setUsername(e.target.value)} />  
  
    <input value={password}  
          onChange={e => setPassword(e.target.value)} />  
  </>);  
}
```

USERREDUCER HOOK

useReducer: Redux für Komponenten?

- Für komplexen State mit viel Logik zur Veränderung

USERREDUCER HOOK

useReducer: Redux für Komponenten?

Schritt 1: Reducer-Funktion (state, action) => newState

```
function loginReducer(oldState, action) {  
  switch (action.type) {  
    case "SET_USER":  
      return {...oldState, username: action.username };  
  
    case "SET_PASSWORD":  
      return {...oldState, password: action.password };  
  
    case "RESET":  
      return { user: "", password: "" };  
  
    default:  
      return oldState;  
  }  
}
```

USERREDUCER HOOK

useReducer: Redux für Komponenten?

Schritt 2: Verwendung

```
function loginReducer() { ... }

function LoginForm(props) {
  const [state, dispatch] = React.useReducer(loginReducer);

  return (>
    </>);
}
```

USERREDUCER HOOK

useReducer: Redux für Komponenten?

Schritt 2a: Zugriff auf den State

```
function loginReducer() { ... }

function LoginForm(props) {
  const [state, dispatch] = React.useReducer(loginReducer);

  return (<>
    <input value={state.username}>
    />

    <input value={state.password}>
    />

    <button />;
  </>);
}
```

USERREDUCER HOOK

useReducer: Redux für Komponenten?

Schritt 2b: Verändern des States über Actions

```
function loginReducer() { ... }

function LoginForm(props) {
  const [state, dispatch] = React.useReducer(loginReducer);

  return (<>
    <input value={state.username}
      onChange={e =>
        dispatch({type: "SET_USER", username: e.target.value})} />

    <input value={state.password}
      onChange={e =>
        dispatch({type: "SET_PASSWORD", password: e.target.value})}/>

    <button onClick={() => dispatch({type: "CLEAR"})};
  </>);
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State

Beispiel: globaler Reducer für Session-Informationen

```
function authenticationReducer(state, action) {  
  switch (action.type) {  
    case "LOGIN_SUCCESS":  
      return { sessionToken: action.sessionToken }  
    case "LOGIN_FAILED":  
      return { sessionToken: null, error: action.error }  
    case "LOGOUT":  
      return { sessionToken: null, error: null }  
    default:  
      return null;  
  }  
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

Beispiel: Provider-Komponente für dispatcher

```
function authenticationReducer(state, action) { ... }
```

```
function ChatApp(props) {  
  const [state, dispatch] = useReducer(authenticationReducer);
```

```
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

Beispiel: Provider-Komponente für dispatcher

```
function authenticationReducer(state, action) { ... }

const AuthDispatcher = React.createContext(null);

function ChatApp(props) {
  const [state, dispatch] = useReducer(authenticationReducer);

}

}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

Beispiel: Provider-Komponente für dispatcher

```
function authenticationReducer(state, action) { ... }

const AuthDispatcher = React.createContext(null);

function ChatApp(props) {
  const [state, dispatch] = useReducer(authenticationReducer);
  return (
    <AuthDispatcher.Provider value={dispatch}>
      </AuthDispatcher.Provider>
    );
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht

Beispiel: Provider-Komponente für dispatcher

```
function authenticationReducer(state, action) { ... }

const AuthDispatcher = React.createContext(null);

function ChatApp(props) {
  const [state, dispatch] = useReducer(authenticationReducer);
  return (
    <AuthDispatcher.Provider value={dispatch}>
      {state.sessionToken ? props.children : "please login"}
    </AuthDispatcher.Provider>
  );
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht
- Unterkomponenten können dispatch-Funktion verwenden

```
function LogoutButton(props) {  
  const dispatch = useContext(AuthDispatcher) ;  
  
}  
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht
- Unterkomponenten können dispatch-Funktion verwenden

```
function LogoutButton(props) {  
  const dispatch = useContext(AuthDispatcher);  
  
  return (  
    <button onClick={() => dispatch({type: "LOGOUT"})}>  
      Logout  
    </button>  
  );  
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Dispatch (und ggf. State) wird über Context nach unten gereicht
- Unterkomponenten können dispatch-Funktion verwenden
- Empfehlung: nur für State, der sich nicht oft ändert

```
function LogoutButton(props) {  
  const dispatch = useContext(AuthDispatcher) ;  
  
  return (  
    <button onClick={() => dispatch({type: "LOGOUT"})}>  
      Logout  
    </button>  
  );  
}
```

ARBEITEN MIT SEITENEFFEKTEN

Server-Zugriffe, Subscriptions etc sind Seiteneffekte

ARBEITEN MIT SEITENEFFEKTEN

Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

ARBEITEN MIT SEITENEFFEKTEN

Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  componentWillUnmount() {  
    this.disconnectFromApi()  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

ARBEITEN MIT SEITENEFFEKTEN

Server-Zugriffe, Subscriptions etc sind Seiteneffekte

- Bislang nur in Klassen-Komponenten

```
class ChatPage extends React.Component {  
  componentDidMount() {  
    this.disconnectFromApi = ChatApi.subscribe(this.props.apiKey);  
  }  
  
  componentWillUnmount() {  
    this.disconnectFromApi()  
  }  
  
  componentDidUpdate(prevProps) {  
    if (prevProps.apiKey !== this.props.apiKey) {  
      ChatApi.subscribe(this.props.apiKey);  
    }  
  }  
  
  render() { return <div><h1>Chat</h1>...</div> }  
}
```

Nur ausführen, wenn Properties sich geändert haben

ARBEITEN MIT SEITENEFFEKTEN

useEffect: Seiteneffekte in Funktionskomponenten

```
function ChatPage(props) {  
  React.useEffect(  
    () => { ----- Ersetzt componentDidMount & componentDidUpdate  
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
  
    },  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

ARBEITEN MIT SEITENEFFEKTEN

useEffect: Seiteneffekte in Funktionskomponenten

Aufräumen in Rückgabe-Funktion

```
function ChatPage(props) {  
  React.useEffect(  
    () => {  
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
      return () => disconnectFromApi();  
    },  
    |  
    Ersetzt componentWillMount  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

ARBEITEN MIT SEITENEFFEKTEN

useEffect: Seiteneffekte in Funktionskomponenten

Bedingte Ausführung

Ohne 2. Parameter wird Hook u.U. in Endlosschleife ausgeführt

```
function ChatPage(props) {  
  React.useEffect(  
    () => {  
      const disconnectFromApi = ChatApi.subscribe(props.apiKey);  
      return () => disconnectFromApi();  
    },  
    [props.apiKey] ----- Ersetzt Property-Vergleich in componentDidUpdate  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) {  
  const [value, setValue] = React.useState(initialValue);  
  
  function onEnterHandler(e) {  
    const keyCode = e.which || e.keyCode;  
    if (keyCode === 13) {  
      onEnter(value);  
    }  
  }  
  
  return {  
    value,  
    onChange: e => setValue(e.target.value),  
    onKeyPress: onEnterHandler  
  };  
}
```

CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) { ... }
```

```
// Verwendung:
```

```
function LoginDialog(props) {
  const usernameInput = useFormInput("", ChatApi.login);

  return <form>
    <input {...usernameInput} />
  </form>
}
```

CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Handler für Input-Felder

```
function useFormInput(initialValue, onEnter) { ... }
```

// Verwendung:

```
function LoginDialog(props) {
  const usernameInput = useFormInput("", ChatApi.login);
  const passwordInput = useFormInput("", ChatApi.login);

  return <form>
    <input {...usernameInput} />
    <input {...passwordInput} />
  </form>
}
```

CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook", zum Laden von Daten

```
function useApi(path, initData) {
```

```
}
```

CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook", zum Laden von Daten
- Alle Hooks können in Custom Hooks verwendet werden

```
function useApi(path, initialValue) {
  const [data, setData] = React.useState(initialValue);

  React.useEffect(() => {
    async function readData() {
      const response = await fetch(`http://localhost:9000/${path}`);
      const data = await response.json();
      setData(data);
    }

    readData();
  }, [path]);

  return data;
}
```

CUSTOM HOOKS

Eigene Hooks sind möglich und können wiederverwendet werden

- Beispiel: Generischer "fetch hook"

```
function useApi(path, initialValue) { ... }

// Verwendung
function Dashboard(props) {
  const logs = useApi("/logs", []);
  const users = useApi("/users", []);

  return <>
    <LogViewer logs={logs} />
    <UsersViewer users={users} />
  </>;
}
```

HOOKS

- **Müssen wir jetzt alle Hooks verwenden?** 😱
- **Was ist mit unseren Klassen?** 😱

HOOKS

- **Müssen wir jetzt alle Hooks verwenden?** 😱
- **Was ist mit unseren Klassen?** 😱
- **Zunächst:**
 - Hooks sind "opt-in"
 - Hooks sind abwärtskompatibel
 - Eingeführt in Minor-Version (!)

HOOKS

- Müssen wir jetzt alle Hooks verwenden? 😱
- Was ist mit unseren Klassen? 😱
- ...also: keine Panik! React bleibt stabil! 😊

Finally, there is no rush to migrate to Hooks. We recommend avoiding any “big rewrites”, especially for existing, complex class components. It takes a bit of a mindshift to start “thinking in Hooks”. In our experience, it’s best to practice using Hooks in new and non-critical components first, and ensure that everybody on your team feels comfortable with them. After you give Hooks a try, please feel free to send us feedback, positive or negative.

We intend for Hooks to cover all existing use cases for classes, but **we will keep supporting class components for the foreseeable future**. At Facebook, we have tens of thousands of components written as classes, and we have absolutely no plans to rewrite them. Instead, we are starting to use Hooks in the new code side by side with classes.

16.6

Suspense

RENDERN UNTERBRECHEN

SUSPENSE

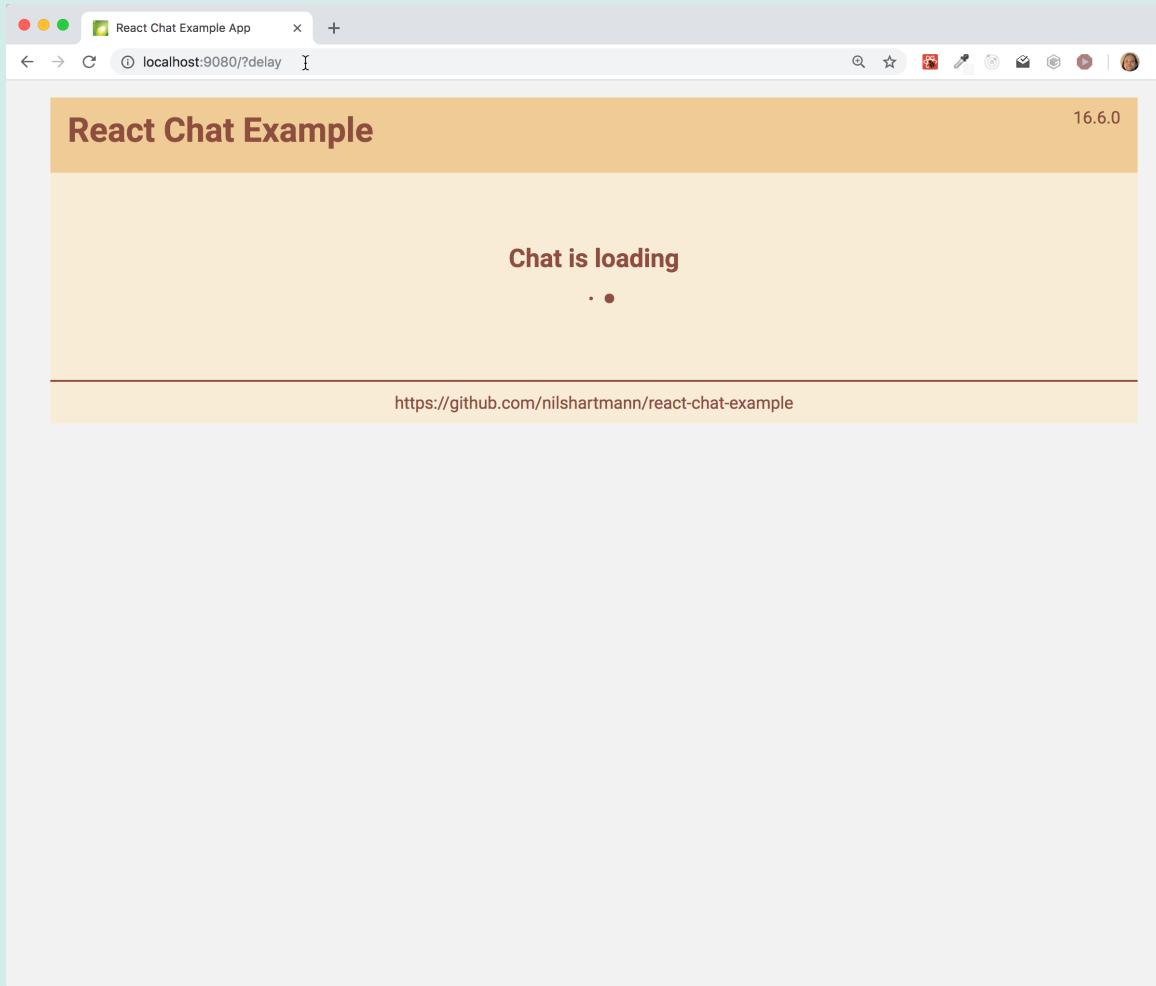
Suspense: React kann das Rendern von Komponenten unterbrechen, während (asynchron) Daten geladen werden [16.6]

- Funktioniert aktuell (nur) für Code Splitting

DEMO: LAZY UND SUSPENSE

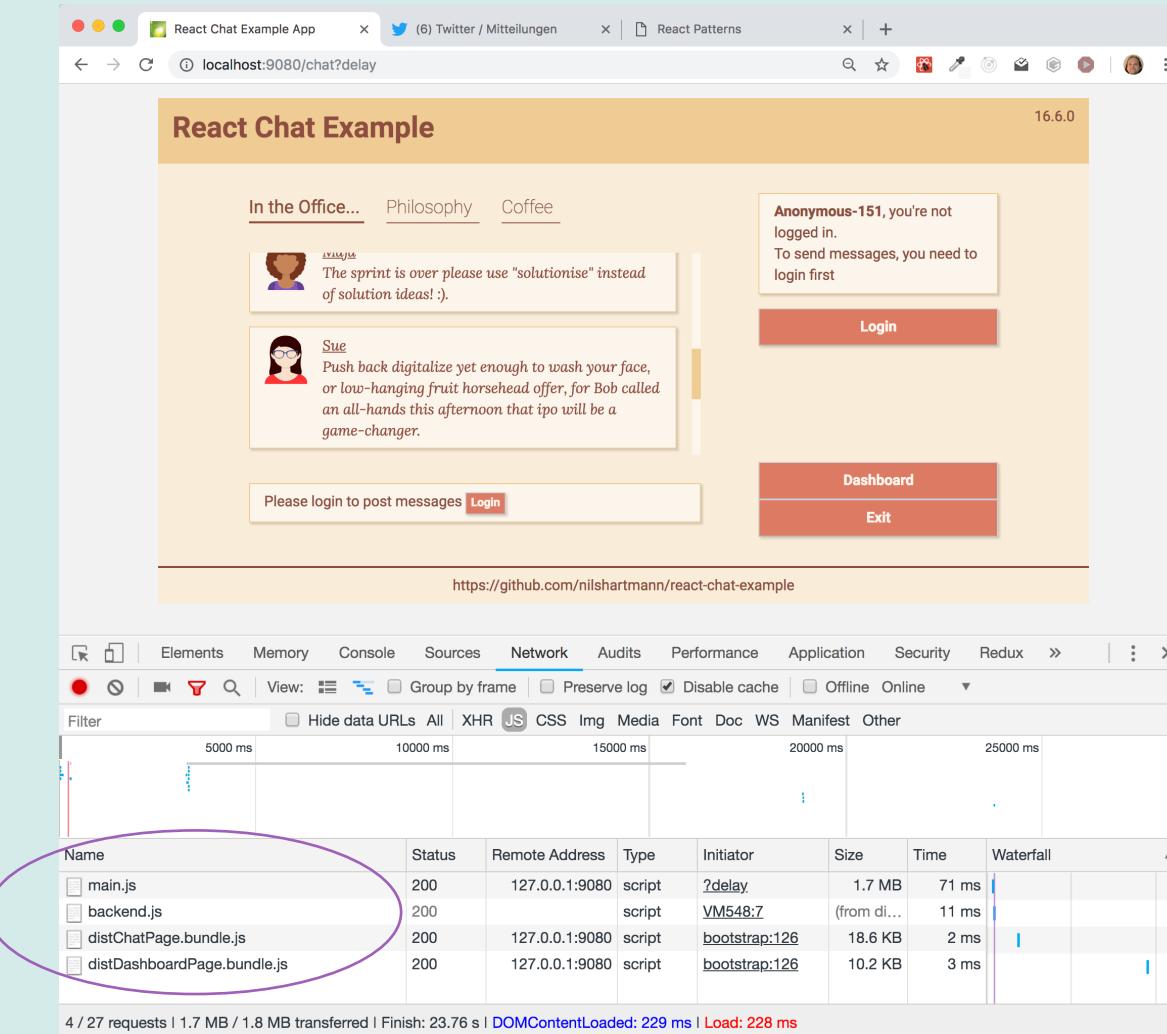
- **Demo: Fallback Komponente**

<http://localhost:9080/?delay>



DEMO: LAZY UND SUSPENSE

- **Netzwerk Requests**
<http://localhost:9080/?delay>



React Chat Example App (6 Twitter / Mitteilungen) | React Patterns

localhost:9080/chat?delay

React Chat Example

16.6.0

In the Office... Philosophy Coffee

AYAMAJM The sprint is over please use "solutionise" instead of solution ideas! .

Sue Push back digitalize yet enough to wash your face, or low-hanging fruit horsehead offer, for Bob called an all-hands this afternoon that ipo will be a game-changer.

Please login to post messages [Login](#)

Anonymous-151, you're not logged in.
To send messages, you need to login first

[Login](#)

[Dashboard](#)

[Exit](#)

<https://github.com/nilshartmann/react-chat-example>

Elements Memory Console Sources Network Audits Performance Application Security Redux

View: Group by frame Preserve log Disable cache Offline Online

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Name	Status	Remote Address	Type	Initiator	Size	Time	Waterfall
main.js	200	127.0.0.1:9080	script	?delay	1.7 MB	71 ms	
backend.js	200	127.0.0.1:9080	script	VM548:7	(from di...	11 ms	
distChatPage.bundle.js	200	127.0.0.1:9080	script	bootstrap:126	18.6 KB	2 ms	
distDashboardPage.bundle.js	200	127.0.0.1:9080	script	bootstrap:126	10.2 KB	3 ms	

4 / 27 requests | 1.7 MB / 1.8 MB transferred | Finish: 23.76 s | DOMContentLoaded: 229 ms | Load: 228 ms

SUSPENSE

React.lazy: Code splitting with Suspense [16.6]

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));  
class App {  
  render() {  
    return <>  
      <ChatPage />  
      // more pages...  
    </>  
  }  
}
```

Dynamic Import

SUSPENSE

React.Suspense: Zeigt Fallback Komponente an [16.6]

- Bis Komponente geladen ist, wird Spinner o.ä. angezeigt

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));

class App {
  render() {
    return <>
      <React.Suspense fallback={<h1>Loading...</h1>}>
        <ChatPage />
        // more pages...
      </React.Suspense>
    <>
  }
}
```

SUSPENSE

Ausblick [16.x ~mid 2019]: Suspense for Data Fetching

- *Alle gezeigten Beispiele verwenden unstable API!!*

BEISPIEL: DATEN LADEN MIT SUSPENSE

- REST Aufrufe mit fetch

/api/logs

Admin Dashboard

Close

Logs

```
[Anonymous-361] client disconnected  
[Anonymous-365] Assigned User id 'Anonymous-365'  
[Anonymous-365] Client registered  
[Anonymous-365] join chatroom with id 'r1'  
[Anonymous-366] Assigned User id 'Anonymous-366'  
[Anonymous-366] Client registered  
[Anonymous-366] join chatroom with id 'r1'  
[Anonymous-364] client disconnected  
[Anonymous-367] Assigned User id 'Anonymous-367'  
[Anonymous-367] Client registered
```

/api/users

User

Id	Name
u1	Susi
u2	Klaus
u3	Harry
u4	Peter
u5	Maja
u6	Sue
u7	Olivia
u8	Cathy

<https://github.com/nilshartmann/react-chat-example>

DATEN LADEN MIT SUSPENSE - 1

- **Daten laden mit Suspense**

- Beim Rendern wird eine Funktion aufgerufen die Daten liefert – oder auch nicht, dann wird Rendern **pausiert**
- Sobald die Funktion (später) Daten liefert, wird die Komponente gerendert
- Daten werden aus react-cache kommen (unstabile API zurzeit)

```
function LogsView() {  
  const logs = LogsResource.read(); // kehrt nur mit Daten zurück  
  
  return <div> ...geladene logs hier anzeigen... </div>;  
}
```

AUSBLICK: SUSPENSE AUF DEM SERVER

Suspense for Server Rendering

We started designing a new server renderer that supports Suspense (including waiting for asynchronous data on the server without double rendering) and progressively loading and hydrating page content in chunks for best user experience. You can watch an overview of its early prototype in [this talk](#). The new server renderer is going to be our major focus in 2019, but it's too early to say anything about its release schedule. Its development, as always, will happen on GitHub.

<https://reactjs.org/blog/2018/11/27/react-16-roadmap.html#suspense-for-server-rendering>

16.x unstable!

Concurrent React

AUSBLICK

CONCURRENT REACT

Time Slicing: Bessere Nutzung von CPU Zeiten

- Rendern ist eine "non-blocking" Operation
 - Es kann **immer** auf User-Interaktionen reagiert werden

CONCURRENT REACT

Time Slicing: Bessere Nutzung von CPU Zeiten

- Rendern ist eine "non-blocking" Operation
 - Es kann **immer** auf User-Interaktionen reagiert werden
- Komponenten können vor-gerendert werden, ohne sofort sichtbar zu sein
 - Ohne Nachteile für sichtbare Komponenten (Performance)

CONCURRENT REACT

Time Slicing: Bessere Nutzung von CPU Zeiten

- Rendern ist eine "non-blocking" Operation
 - Es kann **immer** auf User-Interaktionen reagiert werden
- Komponenten können vor-gerendert werden, ohne sofort sichtbar zu sein
 - Ohne Nachteile für sichtbare Komponenten (Performance)

Suspense: Besseres Umgehen mit IO

- Einheitliche API für das Arbeiten mit asynchronen Daten
- Pausieren des Renderns von **einem Teil** der Komponenten

CONCURRENT MODE

Concurrent Mode [16.7]

- Concurrent Mode muss explizit eingeschaltet werden
- Geht auf jeder Ebene in der Anwendung
 - Sehr gut für Migration, falls es Probleme gibt

```
ReactDOM.render(  
  <React.StrictMode>  
    <React.unstable_ConcurrentMode>  
      <ErrorHandler>  
        <App />  
      </ErrorHandler>  
    </React.unstable_ConcurrentMode>  
  </React.StrictMode>,  
  getDocumentById("...")  
) ;
```

SUSPENSE MIT CONCURRENT MODE

Scheduler: Erlaubt es, Aktionen zu priorisieren

- "Unwichtige" Aktionen niedriger priorisieren (z.B. Grafik aktualisieren)
- Wichtige Aktionen (z.B. User-Interaktion) bleiben dadurch flüssig

```
import {  
  unstable_LowPriority,  
  unstable_runWithPriority,  
  unstable_scheduleCallback  
} from "scheduler";  
  
function deferLoadUnimportantData(url) {  
  unstable_runWithPriority(unstable_LowPriority, function() {  
    unstable_scheduleCallback(function() {  
      loadUnimportantData(url);  
    });  
  });  
}  
}
```

SUSPENSE MIT CONCURRENT MODE

Suspense: Flickern verhindern mit Concurrent Mode

- `maxDuration` legt eine Zeit fest, bis `fallback` gerendert wird
- Bis dahin wird bestehende Komponente angezeigt

```
const ChatPage = React.lazy(() => import("./chat/ChatPage"));

class App {
  render() {
    return <>
      <React.Suspense maxDuration={100} fallback={<h1>...</h1>}>
        <ChatPage />
        // more pages...
      </React.Suspense>
    <>
  }
}
```

ZUSAMMENFASSUNG – SUSPENSE & CONCURRENT RENDERING

- **Ab React 16.x**

- Suspense
 - Kann das Rendern eines Teils der Hierarchie unterbrechen und später fortsetzen
 - Funktioniert heute für Lazy Loading von Komponenten
- Concurrent Mode
 - Erlaubt es React, verschiedene Render Vorgänge unterschiedlich zu priorisieren
- Cache API
 - Neue Möglichkeit, Daten für React zu laden
 - Sieht synchron aus, blockiert aber (trotzdem) nicht
 - API noch unstabil

vielen Dank!

Slides: <https://bit.ly/md-dev-days-react-2019>