

Hooks, Concurrent Rendering, Suspense API

Alles neu in React?

Slides: <https://nils.buzz/ejs2019-react>

NILS HARTMANN

nils@nilshartmann.net

Freiberuflischer Entwickler, Architekt, Trainer aus Hamburg

Java
JavaScript, TypeScript
React
GraphQL

Trainings & Workshops

<https://nilshartmann.net/react-workshops>

HTTPS://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", and "Coffee".
- Content Area:** A list of messages from various users (Harry, Peter, Maja, Sue) and system notifications (Anonymous joins/leaves). A message from "Anonymous-620" states: "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." A "Login" button is present.
- Footer:** "Please login to post messages" and a "Login" button.

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", and "Coffee".
- Content Area:** The same list of messages and notifications as the left screenshot. A message from "Anonymous-620" is highlighted with a yellow background: "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." A "Login" button is present.
- Right Sidebar:** Displays a user profile for "Klaus" with the message "You're logged in as Klaus".
- Bottom Right:** Buttons for "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

HINTERGRUND

Hooks: State, Context etc auch in Funktionskomponenten

Motivation:

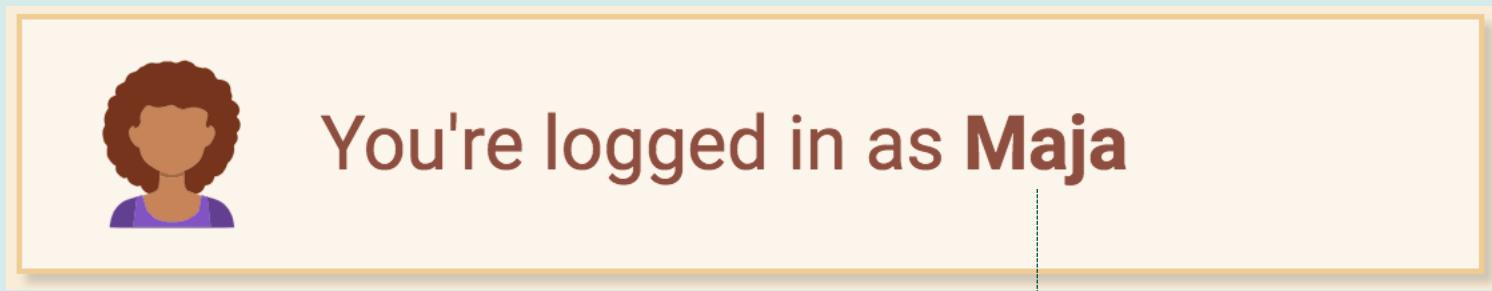
- Bessere Wiederverwendbarkeit von Code
- Logik in Klassen nicht immer einfach verständlich (insb Lifecycles)
 - Durch Concurrent Rendering noch problematischer

Hooks sind reguläre Funktionen, aber...

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

USECONTEXT HOOK

useContext: Vereinfachter Zugriff auf den Context



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";

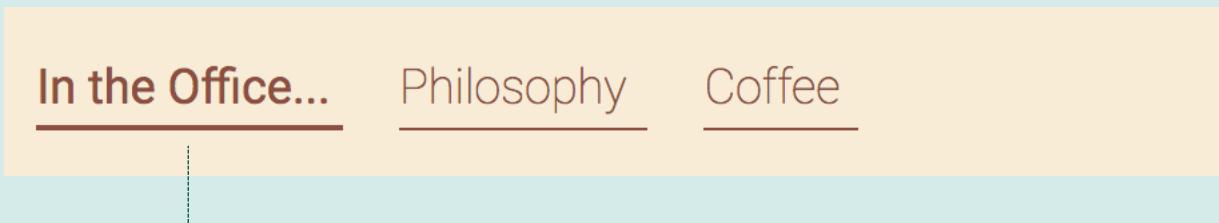
function CurrentUserProfile(props) {
  const chatValue = useContext(ChatContext);
  const themeValue = useContext(ThemeContext);

  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

USERREDUCER HOOK

useReducer: Redux für Komponenten?

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

Actions sind einfache JavaScript-Objekte

```
const action = {
  type: "SET_USER", ----- Type
  username: "..." ----- Payload
}
```

```
const action = {
  type: "SET_PASSWORD",
  password: "..."
}
```

```
const action = {
  type: "RESET"
}
```

USERREDUCER HOOK

useReducer: Redux für Komponenten?

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

```
function loginReducer(oldState, action) {  
  switch (action.type) {  
    case "SET_USER":  
      // ...  
    }  
  }  
}
```

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};  
  }  
}  
}
```

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 throw new Error("Invalid action!");  
  }  
}
```

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: Ein globaler Reducer für Session-Informationen

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

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Über Context werden State und Dispatch nach unten gereicht

Beispiel: Provider-Komponente für dispatcher

```
const AuthDispatcherCtx = React.createContext(null);  
  
function AuthProvider(props) {
```

```
}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Über Context werden State und Dispatch nach unten gereicht

Beispiel: Provider-Komponente für dispatcher

```
const AuthDispatcherCtx = React.createContext(null);

function AuthProvider(props) {
  const [authState, dispatch] = useReducer(authenticationReducer);

}

}
```

USERREDUCER HOOK

useContext & useReducer: Für globalen App State?

- Reducer für globalen App State
- Über Context werden State und Dispatch nach unten gereicht

Beispiel: Provider-Komponente für dispatcher

```
const AuthDispatcherCtx = React.createContext(null);

function AuthProvider(props) {
  const [authState, dispatch] = useReducer(authenticationReducer);
  return (
    <AuthDispatcherCtx.Provider
      value={{authState, dispatch}}>
      {props.children}
    </AuthDispatcherCtx.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
- (Empfehlung: nur für State, der sich nicht oft ändert)

Beispiel: Verwendung innerhalb der Anwendung

```
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
- (Empfehlung: nur für State, der sich nicht oft ändert)

Beispiel: Verwendung innerhalb der Anwendung – Actions auslösen

```
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)

Beispiel: Verwendung innerhalb der Anwendung – State verwenden

```
function Avatar(props) {  
  const {authState} = useContext(AuthDispatcher);  
  
  return (  
    <h1>Hello, {authState.user}</h1>  
  );  
}
```

USERREDUCER HOOK

CustomHooks: Für globalen App State?

- Es können eigene Hooks definiert werden, die auch Hooks verwenden dürfen
- Damit wäre auch "fachliche" API möglich, um auf dispatch zu verzichten

USERREDUCER HOOK

CustomHooks: Für globalen App State?

- Es können eigene Hooks definiert werden, die auch Hooks verwenden dürfen
- Damit wäre auch "fachliche" API möglich, um auf dispatch zu verzichten

Beispiel: Custom Hook

```
function useAuth(props) {  
  const {dispatch, authState} = useContext(AuthDispatcher);  
  
  const logout = () => dispatch({type: "LOGOUT"});  
  const login = () => dispatch({type: "LOGOUT", ...});  
  
  return {  
    logout, login, authState  
  }  
}
```

USERREDUCER HOOK

CustomHooks: Für globalen App State?

- Es können eigene Hooks definiert werden, die auch Hooks verwenden dürfen
- Damit wäre auch "fachliche" API möglich, um auf dispatch zu verzichten

Beispiel: Verwendung Custom Hook (statt dispatch)

```
function Logout(props) {  
  const {logout} = useAuth();  
  
  return <button onClick={logout}>Logout</button>  
}
```

USERREDUCER HOOK

CustomHooks: Für globalen App State?

- Es können eigene Hooks definiert werden, die auch Hooks verwenden dürfen
- Damit wäre auch "fachliche" API möglich, um auf dispatch zu verzichten

Beispiel: Verwendung Custom Hook (statt state)

```
function Avatar(props) {  
  const {authState} = useAuth();  
  
  return (  
    <h1>Hello, {authState.user}</h1>  
  );  
}
```

REDUX HOOKS

Redux bietet Hooks API an (seit v7.1.0)

- useSelector wird verwendet, um aus dem Store Daten zu lesen
 - Ersetzt mapStateToProps

```
function Inbox(props) {  
  
  const messageCount = useSelector(store => store.messages.length);  
  const user = useSelector(store => store.user);  
  
  return (  
    <h1>Hello {user}, you have {messageCount} messages</h1>  
  );  
}
```

REDUX HOOKS

Redux bietet Hooks API an (seit v7.1.0)

- useSelector wird verwendet, um aus dem Store Daten zu lesen
 - Ersetzt mapStateToProps

REDUX HOOKS

Redux bietet Hooks API an (seit v7.1.0)

- useSelector wird verwendet, um aus dem Store Daten zu lesen
 - Ersetzt mapStateToProps
- useDispatch liefert dispatch-Methode zum Versenden von Actions zurück

```
function Logout(props) {  
  const dispatch = useDispatch();  
  
}  
}
```

REDUX HOOKS

Redux bietet Hooks API an (seit v7.1.0)

- useSelector wird verwendet, um aus dem Store Daten zu lesen
 - Ersetzt mapStateToProps
- useDispatch liefert dispatch-Methode zum Versenden von Actions zurück

```
function Logout(props) {  
  
  const dispatch = useDispatch();  
  const logout = () => dispatch({ type: "LOGOUT" })  
  
}
```

REDUX HOOKS

Redux bietet Hooks API an (seit v7.1.0)

- useSelector wird verwendet, um aus dem Store Daten zu lesen
 - Ersetzt mapStateToProps
- useDispatch liefert dispatch-Methode zum Versenden von Actions zurück

```
function Logout(props) {  
  
  const dispatch = useDispatch();  
  const logout = () => dispatch({ type: "LOGOUT" })  
  
  return (  
    <button onClick={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  
    ("Dependencies")  
  );  
  
  return <div><h1>Chat</h1>...</div>  
}
```

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);
```

```
}
```

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} />
  </>;
}
```

NEUE DEVTOOLS

Neue Dev Tools unterstützen auch Hooks

- Beispiel: AddMessage

The screenshot shows the Chrome DevTools interface with the Components tab selected. The left pane displays the component tree, and the right pane provides details about the selected component, `<AddMessage>`.

Component Tree (Left):

- <Tab key="1">
- <Tab key="2">
- <TabPanel key="0">
 - <ChatroomPanel>
 - <MessageContainer>
 - <ChatroomPanelFooter>
 - <AddMessage> (highlighted with a blue selection bar)
 - <Avatar>
- <Sidebar>
- <Footer>

Component Details (Right):

- props**
 - `userId: u5`
 - `onMsgSend: onMsgSend()`
- hooks**
 - `State: "abc"`
- rendered by**
 - `<ChatroomPanelFooter>`
 - `<ChatroomPanel>`
 - `<ChatPage>`
 - `<App>`

NEUE DEVTOOLS

useMemo: Kann verwendet werden, um teure Operationen zu "cachen"

- Komponentenfunktion wird bei jedem neu-rendern ausgeführt
- Funktion in useMemo aber nur, wenn sich **Abhängigkeiten** ändern

```
function AddMessage(props) {  
  
  const [state, setState] = React.useState("");  
  
  const avatar = React.useMemo(  
    () => <Avatar userId={props.userId} />,  
    [props.userId]  
  )  
  
  return <div>  
    {avatar}  
    <input value={state} onChange={...} />  
  </div>;  
}
```

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

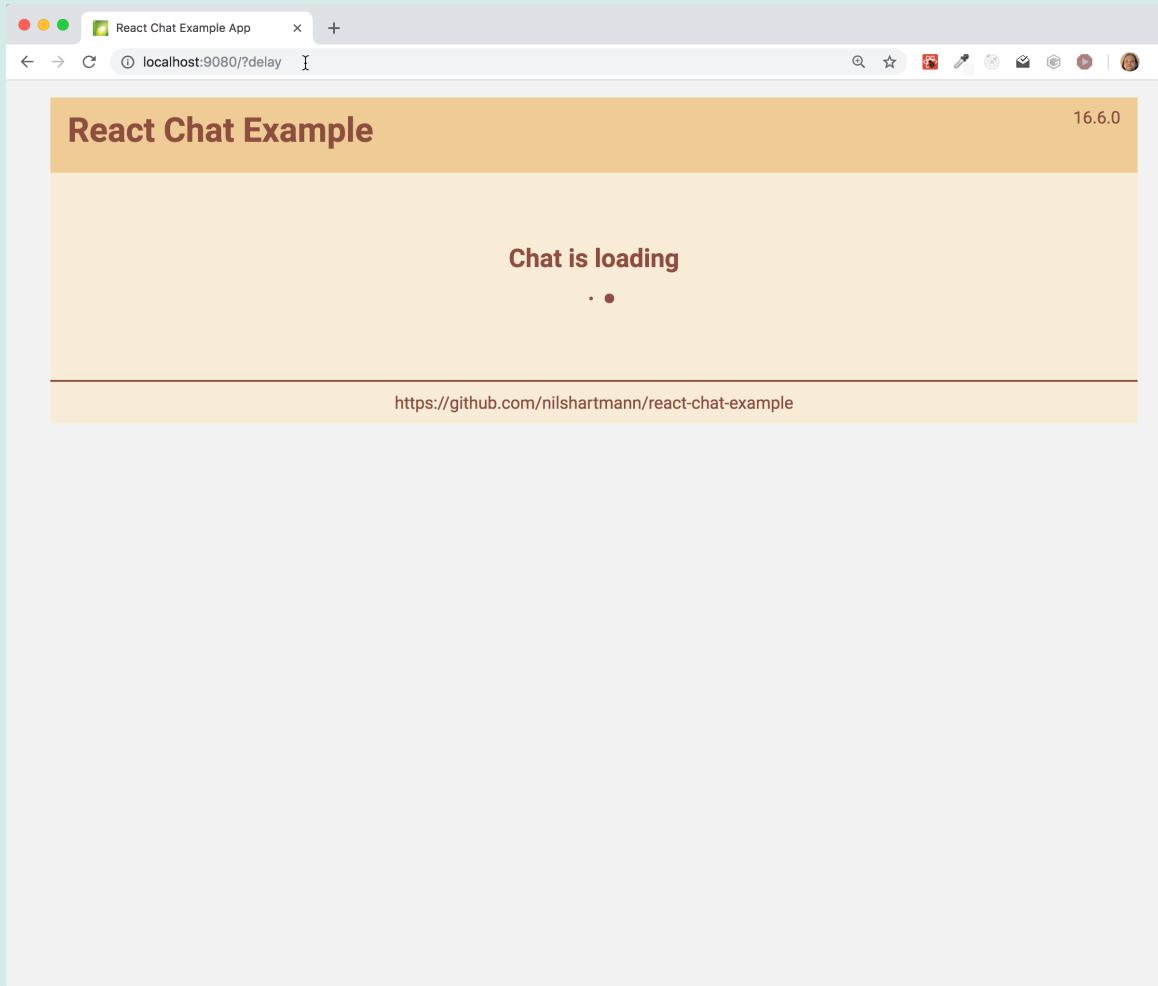
Suspense: React kann das Rendern von Komponenten unterbrechen, während (asynchron) Daten geladen werden [16.6]

- Funktioniert aktuell (nur) für Code Splitting

DEMO: LAZY UND SUSPENSE

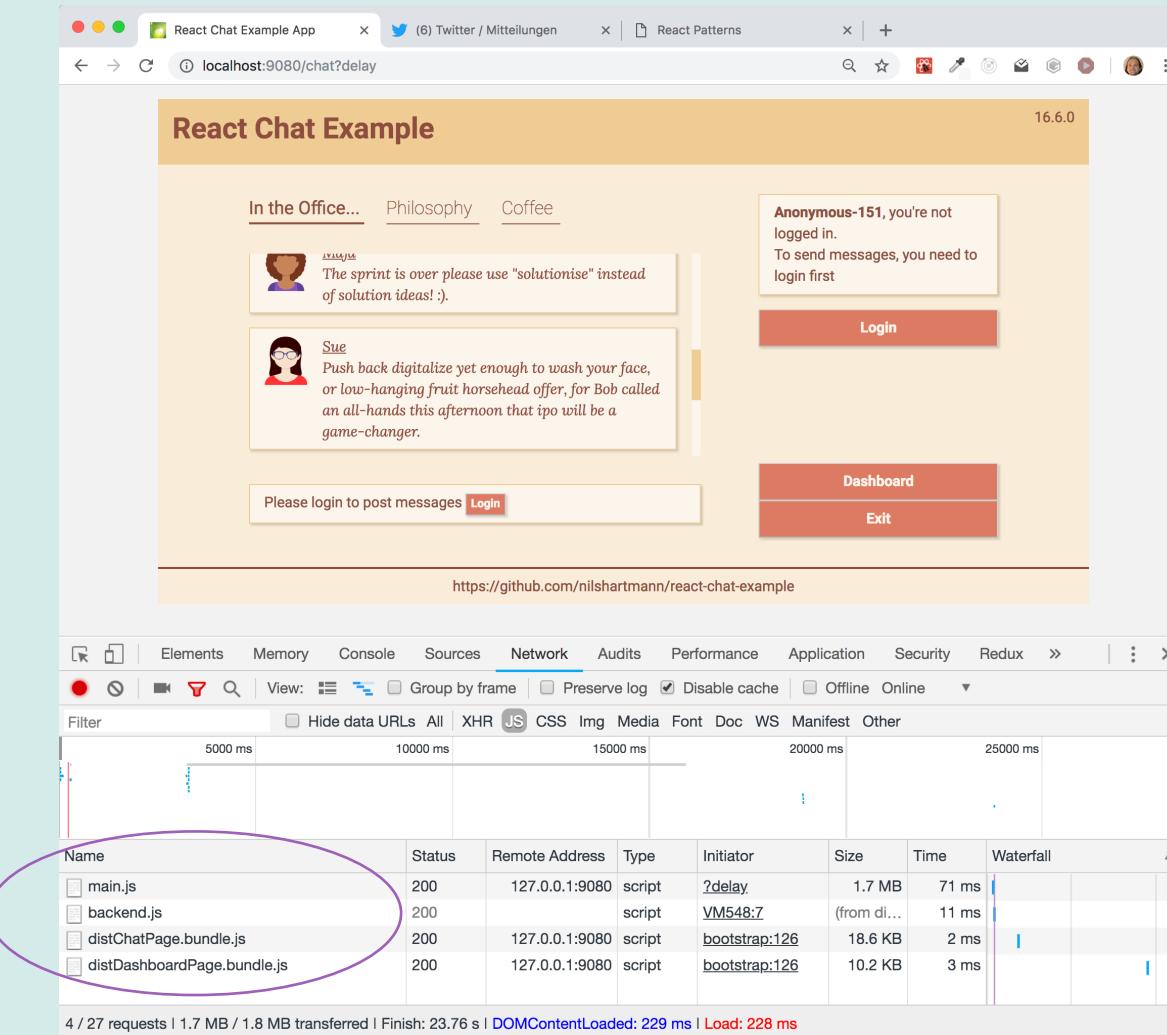
- **Demo: Fallback Komponente**

<http://localhost:9081/?delay>



DEMO: LAZY UND SUSPENSE

- **Netzwerk Requests**
<http://localhost:9081/?delay>



React Chat Example App

localhost:9081/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>

Network

main.js

backend.js

distChatPage.bundle.js

distDashboardPage.bundle.js

5000 ms 10000 ms 15000 ms 20000 ms 25000 ms

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>

ASYNCHRONES DATEN LADEN

- **"Klassisches" Daten laden**

- In componentDidMount Daten das Laden anstoßen
- In der Zwischenzeit Loading Indicator anzeigen
- (Mit Hooks andere API, aber gleiches Konzept)

```
class LogsView extends React.Component {  
  state = {};  
  
  async componentDidMount() {  
    const response = await fetch("/api/logs");  
    const logs = await response.json();  
    this.setState({ logs })  
  }  
  
  render() {  
    if (!this.state.logs) { return <h1>Loading...</h1> }  
    return <div> // render logs </div>;  
  }  
}
```

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>;  
}
```

DATEN LADEN MIT SUSPENSE - 2

- **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
- Komponente wird irgendwo im Tree mit **Suspense** umschlossen

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

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 Rendering

AUSBLICK

CONCURRENT RENDERING

Time Slicing: Bessere Nutzung von CPU Zeiten

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

CONCURRENT RENDERING

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 RENDERING

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

```
const container = document.getElementById("...");  
const root = ReactDOM.unstable_createRoot(container);  
  
root.render(  
  <App />,  
  container  
);
```

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

ZUSAMMENFASSUNG – SUSPENSE & CONCURRENT RENDERING

Ab React 16.x

- Hooks (stabil seit 16.8)
 - Funktionale Komponenten auch mit State, Lifecycle etc
- Suspense
 - Kann das Rendern eines Teils der Hierarchie unterbrechen und später fortsetzen
 - Funktioniert stabil für Lazy Loading von Komponenten (ab 16.6)
 - Zum Laden von Daten noch **unstable**
- Concurrent Mode (**unstable**)
 - Erlaubt es React, verschiedene Render Vorgänge unterschiedlich zu priorisieren
- Cache API (**unstable**)
 - Neue Möglichkeit, Daten für React zu laden
 - Sieht synchron aus, blockiert aber (trotzdem) nicht

vielen Dank!

Slides: <https://nils.buzz/ejs2019-react>

Source Code: <https://nils.buzz/ejs2019-react-chat>