

(Einführung in) React Hooks

React Barcamp 2020

Tim Kraut

AWESOME! Software



Komponenten in React

Deklarative Programmierung

- React: „So soll DOM aussehen, mach mal!“
- Vue & Angular 🥰

Template-Logik

- Bedingungen, Schleifen etc.
- React: JavaScript um „HTML“ erweitern
→ **JSX**
- Vue & Angular 🤨: HTML um „JavaScript“ erweitern
→ eigene Mikrosyntax/DSL

JSX

```
<MyButton onClick={onBuyClick} title="Kaufen" />
```

React-Anwendung

- Mindestens 1 Komponente
- Komponenten implementieren als **Funktion** oder als **Klasse**

Implementierung als Funktion

```
function MyButton({ onClick, title }) {
```

```
...
```

```
  return <button onClick={onClick}>{title}</button>  
}
```

Implementierung als Klasse

```
class MyButton extends Component {
```

```
...
```

```
render() {
```

```
  const {onClick, title} = this.props
```

```
  return <button onClick={onClick}>{title}</button>
```

```
}
```

```
}
```




Gibt
2 Arten...??

Bis React 16.8 (Februar 2019)

- „Einfache“ Komponente? ➔ Funktion
- State oder Lifecycle-Methoden? ➔ Klassen

Lifecycle-Methoden

- `componentDidMount()`
- `componentDidUpdate()`
- `componentWillUnmount()`
- ...

Probleme mit Klassen-Komponenten

- Mehrere Lifecycle-Methoden pro Feature
- Funktionale Natur von React
- Wiederverwendbare Logik mit State



Probleme mit Klassen-Komponenten

Mehrere Lifecycle-Methoden pro Feature



Beispiel: Daten herunterladen

<UserProfilePage **userId**={42} />

```
class UserProfilePage extends Component {  
  componentDidMount() {  
    this.fetchUser(this.props.userId)  
  }  
  
  componentDidUpdate(prevProps) {  
    if (prevProps.userId !== this.props.userId) {  
      this.fetchUser(this.props.userId)  
    }  
  }  
  
  fetchUser = userId => { ... }  
  
  render() { ... }  
}
```



Reaktive Komponente

- **Abhängig von Props?**
componentDidMount() + componentDidUpdate()
- Analog: **componentDidMount() + componentWillUnmount()** zum „**Aufräumen**“
➔ Timer, laufende Netzwerk-Requests, Event-Handler etc.

Problem 1: Fehleranfällig

- 2 Lifecycle-Methoden benötigt
→ Bugs/Memory Leaks

Problem 2: Code-Länge

- Duplizierter Code (2 Lifecycle-Methoden)
- Manuelles Überprüfen von Prop-Änderungen

Länge Codezeilen im Vergleich

- Klassen links, Funktionen mit Hooks rechts



Quelle: [Amelia Wattenberger: Thinking in Hooks](#)

Problem 3: Kohäsion

- Logik über mehrere Lifecycle-Methoden verteilt

Kohäsion

„In der objektorientierten Programmierung beschreibt Kohäsion, wie gut eine Programmeinheit eine **logische Aufgabe** oder Einheit abbildet.“

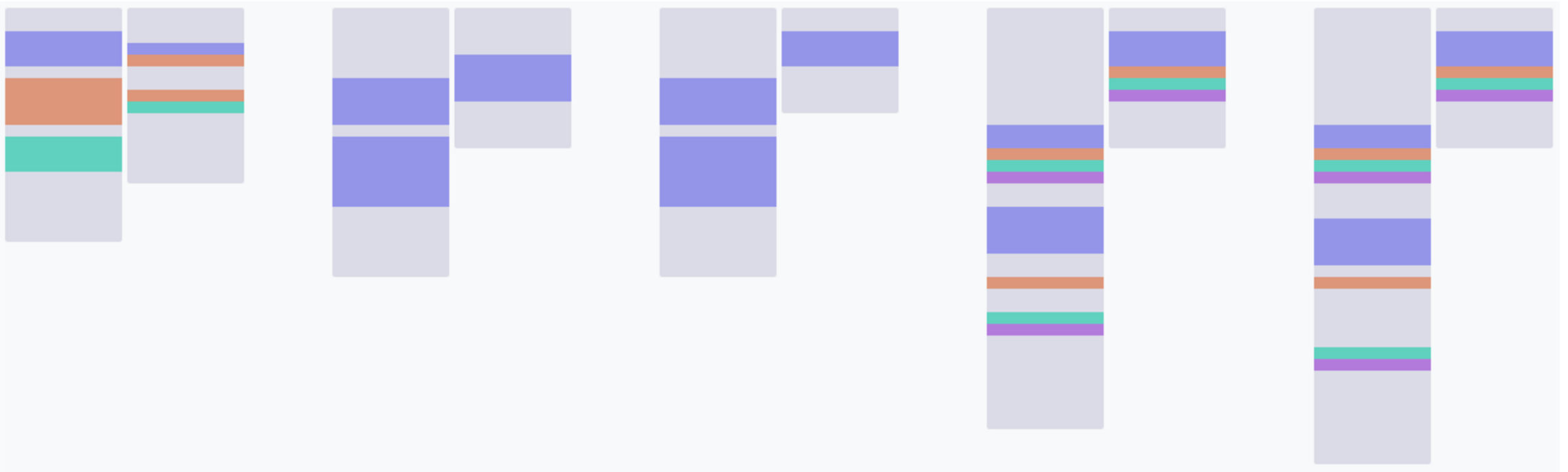
Wikipedia: Kohäsion (Informatik)

Kohäsion

- Was **logisch** zusammen gehört, sollte auch **optisch** zusammen stehen

Kohäsion im Vergleich

- Klassen links, Funktionen mit Hooks rechts



Quelle: [Amelia Wattenberger: Thinking in Hooks](#)

Probleme mit Klassen-Komponenten

Funktionale Natur von React



Erstellen

```
function MyButton({ onClick, title }) {  
  return <button onClick={onClick}>{title}</button>  
}
```

Verwenden

```
<MyButton onClick={onBuyClick} title="Kaufen" />
```

Konzeptionell*

MyButton({ onClick: onBuyClick, title: 'Kaufen' }) ➔ „Markup“

* Technisch: [Dan Abramov – React as a UI Runtime](#)



Konzept hinter Komponenten in React

- **Einfache JavaScript-Funktion**

`square(3) = 9`

- **Abstrakte Funktion**

$f(x) = g$ bzw. $f(p) = h$

- **React-Komponente**

`Komponente(Props) = Markup`

Problem 4:

Passt **DAS** zu
Klassen??



Probleme mit Klassen-Komponenten

Wiederverwendbare Logik mit State



Beispiel: Daten-Fetching mit Spinner

	isLoading	data	error
Initial	false	null	null
Request läuft	true	null	null
Erfolg	false	[{ ... }, { ... }, { ... }]	null
Fehler	false	null	„woah... alles kaputt!!!11!!elf!“

In welchem **Zustand** ist Daten-Fetching jeweils?

- In **State** innerhalb von **Klassen-Komponente** speichern

State aus Klasse wiederverwendbar?

- Komplizierte Patterns: **Higher-Order Components**, Render Props/**Function as a child**, ...
- Oder: Externe Bibliothek (z.B. Redux)

Problem 5: State wiederverwendbar

- Wiederverwendbares Verhalten mit State erfordert **komplizierte Patterns** oder eine **externe Bibliothek**

React Hooks

- State mit `useState()`
- Seiteneffekte mit `useEffect()`
- Custom Hooks
- Rules of Hooks & ESLint-Plugin-React-Hooks
- Weitere Hooks



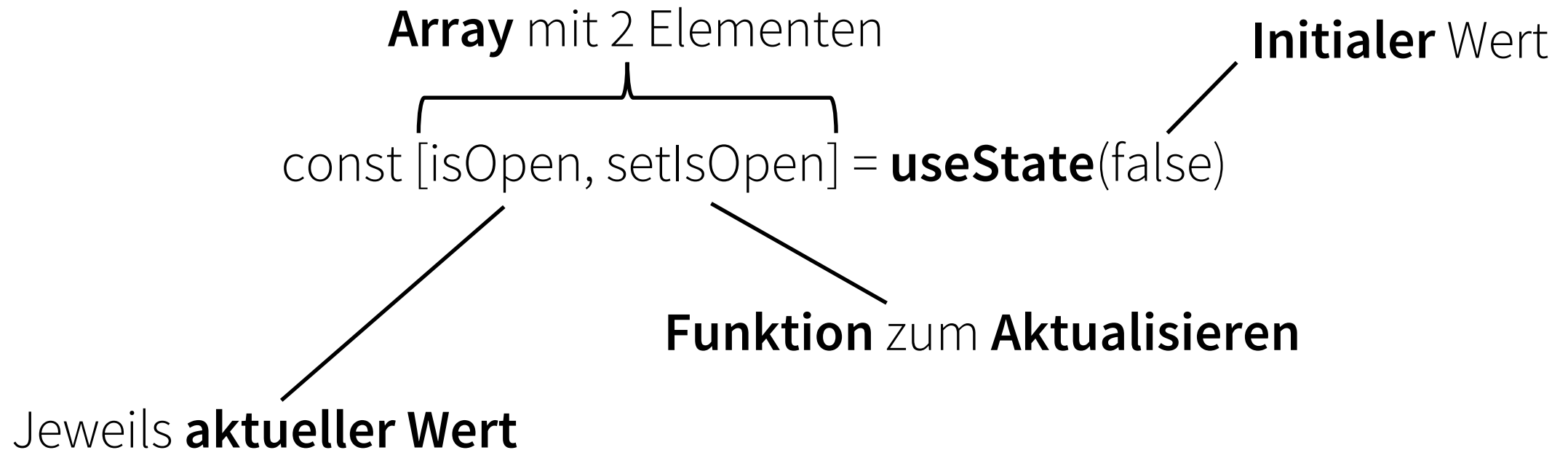
React Hooks

State mit useState()



useState()-Hook (seit v16.8)

```
import React, { useState } from 'react'
```



ES6 – Array Destructuring

```
const myArray = [ 'react', 'barcamp', '2020' ]
```

ES5

```
var word1 = myArray[0]  
var word2 = myArray[1]  
var word3 = myArray[2]
```

ES6

```
const [ word1, word2, word3 ] = myArray
```

word1 // ➔ 'react'

word2 // ➔ 'barcamp'

word3 // ➔ '2020'



Beispiel Toggle

```
function SomeComponent() {  
  const [isOpen, setIsOpen] = useState(false)  
  
  return (  
    <button onClick={() => setIsOpen(!isOpen)}>  
      {isOpen ? 'Close' : 'Open'}  
    </button>  
  )  
}
```



Vorsicht: Unterschied zu Klassen-State

Klassen

```
state = {  
  isLoading: false,  
  data: null,  
  error: null,  
}
```

```
this.setState({  
  isLoading: true,  
  data: null,  
  error: null,  
})
```

Funktionen (Hooks)

```
const [isLoading, setIsLoading] = useState(false)  
const [data, setData] = useState(data)  
const [error, setError] = useState(error)
```

```
setIsLoading(true)  
setData(null)  
setError(null)
```

Mehrere State-Änderungen auf einmal?

- `useReducer()`-Hook

React Hooks

Seiteneffekte mit `useEffect()`



Was sind Seiteneffekte?

- Ziel von React: UI-Komponenten rendern
- Alles andere: **Seiteneffekte** (Konsolen-Ausgaben, Netzwerk-Requests, Zugriff Dateisystem, Cookies lesen/schreiben, ...)

Änderungen

berechnen

Seiteneffekt

(z.B. console.log())

Event

(z.B. Eingabe)

App neuzeichnen

Seiteneffekt

(z.B. Daten
herunterladen)

useEffect()-Hook (seit v16.8)

```
import React, { useEffect } from 'react'
```

useEffect(callback, dependencyArray)

Was soll passieren?

Wann (= wie oft) soll Effect ausgeführt werden?
→ **Immer**, nur bei **Start** oder bei jedem **Update**?

Dependency Array

- **(kein Wert)** = Immer
- **[]** = Einmal bei Komponenten-Start
- **[a, b]** = Immer, wenn sich a oder b geändert hat (Referenz-Vergleich! ➔ Immutability)

Beispiel <title> setzen

```
function Page({ title }) {  
  useEffect(() => {  
    document.title = title  
  }, [title])  
  
  return (  
    <>...</>  
  )  
}
```


Effekt „aufräumen“

```
useEffect(() => {  
  ...  
  return () => {  
    // Clean up  
  }  
}, [...])
```

- ➔ Analog zu unsubscribe() bei RxJS
- ➔ Wird aber von React aufgerufen

Beispiel Effect „aufräumen“

```
useEffect(() => {  
  addEventListener("resize", onResize)  
  
  return () => {  
    removeEventListener("resize", onResize)  
  }  
}, [])
```

Demo: Effekte „aufräumen“

Wann werden Effekte „aufgeräumt“?

- **Rendern**
- Effekt **ausführen**
- **Rendern**
- Effekt **aufräumen**
- Effekt **ausführen**
- ...

Was ist, wenn Code vor Rendern laufen muss?

- `useLayoutEffect()`-Hook

Warum ständiges „Aufräumen“?

- Initiale Anzeige möglichst schnell
- [Kann Bugs verhindern](#)

Ist das nicht total unperformant?!

- Klares **Jein**
- Dependency Array hilft
- Falls Problem: Anzahl Effekt-Evaluierungen reduzieren

React Hooks

Custom Hooks



Custom Hooks

- Eigene Hooks auf Basis von React Hooks

```
import { useEffect } from 'react'
```

```
export function useTitle(title) {  
  useEffect(() => {  
    document.title = title  
  }, [title])  
}
```

➔ **useTitle**('Hallo Köln')

Custom Hooks

- Wiederverwendbar?
 - State ohne komplizierte Patterns?
- ➔ Business-Logik kann in Custom Hooks ausgelagert werden
- ➔ 1 Feature = 1 Custom Hook

Mixins?

- Probleme wie **implizite Dependencies**, **Namens-Kollisionen** etc.

Hooks

- JavaScript „kümmert“ sich
- TypeScript?
- Tests?
- **Trennung von UI und Logik!!!!!!!!!!!!!!!!!!!!**

Sammlung von Custom Hooks

- <https://hooks-guide.netlify.com/>
(übersichtlich, wichtigste Hooks)
- <https://nikgraf.github.io/react-hooks/> (Such-Funktion, sehr viele Hooks)
- <https://github.com/rehooks/awesome-react-hooks> (Übersicht diverser Ressourcen)

React Hooks

Rules of Hooks & ESLint-Plugin-React-Hooks



Rules of Hooks

- **Namenskonvention** für (Custom) Hooks: use...
- Hooks nicht innerhalb von Bedingungen/Schleifen oder verschachtelten Funktionen aufrufen
→ Bedingungen/Schleifen **innerhalb** von Hook möglich!
- Hooks nur im React-Kontext aufrufen (nicht in beliebigen JavaScript-Funktionen)
→ **Innerhalb** von **Komponenten/Custom Hooks**



ESLint-Plugin-React-Hooks

- [Offizielles Plugin für Hooks](#)
- Überprüft Rules of Hooks & **Dependency Array**
➔ (tolle) Autofix-Funktion

P.S.: [ESLint funktioniert auch mit TypeScript](#)

Create-React-App v3

- ESLint-Plugin-React-Hooks ist direkt mit eingebaut

```
"react-hooks/rules-of-hooks": "error", // Checks rules of Hooks  
"react-hooks/exhaustive-deps": "warn" // Checks effect dependencies
```


React Hooks

Weitere Hooks



Übersicht weitere Hooks

- **useContext()**: Consumer Context-API
- **useCallback()** & **useMemo()**: Memoization + Endlosschleifen verhindern
- **useRef()**: Zugriff auf DOM-Elemente, Ersatz für [Instanz-Variablen](#)
- **useImperativeHandle()**: imperative Code-Ausführung von „außen“ ermöglichen
- **useDebugValue()**: Ausgabe in React DevTools

Potenzielle **Fallstricke**



A close-up photograph of a ginger cat with orange and white fur, resting its head on its paws. The cat is lying down, and its eyes are closed. The background is a plain, light-colored surface.

Testen mit
Enzyme?

Probleme mit Enzyme

- [Hooks-Support in Enzyme nicht vollständig/fehlerfrei](#)
- [Shallow-Rendering wird nicht mehr empfohlen](#) (von React-Team)

Besseres Tool mit Hooks-Support:
React-Testing-Library

Stale Closures-Problem

- Closure: Funktion „erinnert“ sich an Variablen aus Umgebung, in der sie definiert wurde
- Gefahr: Variable (z.B. State) wird **aktualisiert**, wegen Closure trotzdem **alter Wert**

Tipps

- ESLint-Plugin-React-Hooks
- State aktualisieren mit Funktion:
setIsOpen(isOpen => !isOpen)

WICHTIG!

Hooks **KEIN 1:1-Ersatz** für Lifecycle-Methoden!

- ➔ Hooks quasi neue **Primitive** für Business-Logik in React (1 Custom Hook = 1 Feature)
- ➔ Erfordern veränderte Denkweise

Lifecycle-Methoden

- **WANN** soll Code ausgeführt werden?

Hooks

- Was soll **SYNCHRON** bleiben?

Dependency Array Teil von diversen Hooks

- `useEffect()`
- `useCallback()`
- `useMemo()`
- `useLayoutEffect()`

➔ Was soll synchron bleiben?

useMemo()

- Erstellt memoisierten Wert
- Referenz bleibt gleich, solange sich Dependency nicht ändert
- `const memoizedValue = useMemo(callback, dependencyArray)`



Offensichtlicher Use Case

Performance-Optimierung

Neuer Use Case

Werte synchron halten

Beispiel useMemo()

```
const FilteredCalendar = ({ dateRange }) => {  
  const filteredItems = useMemo(() => (  
    getCalendarItemsInRange(dateRange)  
  ), [dateRange])  
  
  return (  
    <>...</>  
  )  
}
```

➔ **filteredItems** mit **dateRange** synchron halten

Klassen können nicht zu 100% ersetzt werden

- `getSnapshotBeforeUpdate()` 🤔 🙄
- `componentDidCatch()` ➔ Higher-Order Component für Error Boundaries

Fazit



React-Komponenten: Klasse oder Funktion

Klassen haben diverse Nachteile/Probleme:

- **Fehleranfällig**
- **Code-Länge**
- Prinzip der **Kohäsion** wird verletzt
- Klassen **!= funktionale Natur** von React
- **Wiederverwendbare Logik mit State** nur mit **komplizierten Patterns**/externen Bibliotheken



Hooks lösen Probleme von Klassen

- **Wiederverwendbarkeit** (1 Custom Hook pro Feature)
- **Trennung** von **UI** und **Logik**
- Erfordern **veränderte Denkweise** („was soll synchron bleiben?“)
- Hooks schaffen **neue Herausforderungen** (z.B. Testing, Stale Closures, was sind Best Practices?)
- Hooks sind **Zukunft von React** (und Vue 🧐)

Danke fürs Zuhören!

Tim Kraut

AWESOME! Software

Twitter: **@Tim_Kraut**

