

NILS HARTMANN
<https://nilshartmann.net>

One Year

React Hooks

A (Critical) Review

Slides: <https://nils.buzz/react-meetup-hooks>

NILS HARTMANN

nils@nilshartmann.net

Developer, Architect, Trainer from Hamburg (Freelancer)

JavaScript, TypeScript
React
GraphQL
Java

Trainings, Workshops and
Coachings



[HTTPS://NILSHARTMANN.NET](https://nilshartmann.net)

REACT HOOKS

Anyone NOT knowing what Hooks are?

REACT HOOKS

Anyone NOT knowing what Hooks are?

Hooks 2 Minute intro

- add State, Lifecycle, Sideeffects in functional components
(almost no need for class components anylonger)

REACT HOOKS

Anyone NOT knowing what Hooks are?

Hooks 2 Minute intro

- add State, Lifecycle, Sideeffects in functional components
(almost no need for class components anylonger)
- "Hooks into your components lifecycle"

REACT HOOKS

Anyone NOT knowing what Hooks are?

Hooks 2 Minute intro

- add State, Lifecycle, Sideeffects in functional components
(almost no need for class components anylonger)
- "Hooks into your components lifecycle"
- Regular JavaScript functions...
 - ...but must start with 'use'
 - ...but must not be used in conditionals, for/loops, Class components
 - ...but behaviour is tied to React

REACT HOOKS

Hooks example

```
import React, { useState } from "react";

export default function SettingsForm(props) {

  const [favColor, setFavColor] = useState("blue");

  return <•••>
    <input value={favColor}
          onChange={e => setFavColor(e.target.value)} />
  <•••>
}
```

- **useState** returns value and setter-function
- When state changes, component re-renders
 - component function will run again

REACT HOOKS

One Year of Hooks...

There are some built-in Hooks, like

- **useState**
 - **useReducer** handle state in a Redux-like way but only for one component
 - **useEffect** for sideeffects (replaces lifecycle methods in classes)
 - **useContext** to receive a Context object
 - **useCallback/useMemo/useRef**: solve problems that arise due to using... Hooks
-

REACT HOOKS

One Year of Hooks...

Libraries ship with Hooks, like

- Redux (useSelector, useDispatch, useStore)
- Router (useHistory, useParams, useLocation)
- Apollo Client (useQuery, useMutation)
- React Intl (useIntl)
- React i18n (useTranslation)

Community has them too,

- <https://usehooks.com>
- <https://nikgraf.github.io/react-hooks/>
- <https://www.hooks.guide/>

React Hooks

Good or Evil?

SOME VOICES...



Philipp Spiess
@PhilippSpiess

React Hooks are awesome! 😍

I made: 📦 `useSubstate` - A lightweight hook to subscribe to your single app state



Works with your existing Redux store



.Concurrent React ready (avoids rendering stale state)



Avoids unnecessary re-renders

Check it out: [github.com/philipp-spiess...](https://github.com/philipp-spiess/useSubstate)

Tweet übersetzen

7:51 nachm. · 29. Okt. 2018 · [Twitter Web App](#)

11 Retweets **76** „Gefällt mir“-Angaben

<https://twitter.com/philippspiess/status/1056981916489015296>



*"With Hooks,
React loses its innocence
and becomes Angular"*

Attendee of one of my workshops



Dan Abramov
@dan_abramov



Hooks are weird

8:22 vorm. · 29. Okt. 2018 · [Twitter Web App](#)

33 Retweets **276** „Gefällt mir“-Angaben

https://twitter.com/dan_abramov/status/1056808552180793344



Tom Dale
@tomdale



My feelings about React hooks are mixed, but I do feel strongly about one thing: there's no way React would have gained its current popularity if this was the starting place.

[Tweet übersetzen](#)

12:04 vorm. · 7. Sep. 2019 · [Twitter Web App](#)

21 Retweets **152** „Gefällt mir“-Angaben

<https://twitter.com/tomdale/status/1170095532066430977>



Tom Dale
@tomdale



Old React was simple and fun and “just JavaScript.” New React is more powerful and more correct and better all around—at the expense of becoming a weird magical meta-language on top of JavaScript.

[Tweet übersetzen](#)

12:04 vorm. · 7. Sep. 2019 · [Twitter Web App](#)

4 Retweets 73 „Gefällt mir“-Angaben

<https://twitter.com/tomdale/status/1170095532922064901>

"awesome"

"weird"

"mixed feelings"

"angular"

WHY?

A LOOK AT THE API

REACT HOOKS

A LOOK AT THE API

useContext to access React Context in your functional component

```
export default function SettingsForm(props) {  
  const contextValue = React.useContext(ThemeContext);  
  
  return <p>Your context color: {contextValue.color}</p>  
}
```

A LOOK AT THE API

useContext to access React Context in your functional component

```
export default function SettingsForm(props) {  
  const contextValue = React.useContext(ThemeContext);  
  
  return <p>Your context color: {contextValue.color}</p>  
}
```

- **Noteable:** This is probably easy to understand:
I want to use context "ThemeContext" here in my component

A LOOK AT THE API

useContext to access React Context in your functional component

```
export default function SettingsForm(props) {  
  const contextValue = React.useContext(ThemeContext);  
  
  return <p>Your context color: {contextValue.color}</p>  
}
```

- **Noteable:** This is probably easy to understand:
I want to use context "ThemeContext" here in my component
- **But:** if context changes, SettingsForm will automatically be re-executed!
Why? Because it's a ... Hook ("weird magical meta-language")
"Something" happens in the background to make that work
There is no indicator that this will happen. Syntactically "only" JavaScript

A LOOK AT THE API

useState for local State in your functional component

```
export default function SettingsForm(props) {  
  const [ favColor, setFavColor ] = useState("red");  
  
  return <input value={favColor}  
            onChange={e => setFavColor(e.target.value)} />  
}
```

A LOOK AT THE API

useState for local State in your functional component

```
export default function SettingsForm(props) {  
  const [ favColor, setFavColor ] = useState("red");  
  
  return <input value={favColor}  
            onChange={e => setFavColor(e.target.value)} />  
}
```

- **Noteable:** Return Value
 - what is this? Tuple! (btw: I think Tuples will make it to JavaScript)
 - unusual (yet), but elegant, allows to name my variables as I want them to

A LOOK AT THE API

useState for local State in your functional component

```
export default function SettingsForm(props) {  
  const [ favColor, setFavColor ] = useState("red");  
  
  return <input value={favColor}  
            onChange={e => setFavColor(e.target.value)} />  
}
```

- **Noteable:** Return Value
 - what is this? Tuple! (btw: I think Tuples will make it to JavaScript)
 - unusual (yet), but elegant, allows to name my variables as I want them to
- **Noteable:** initial value, used only once even if this function is run on each render
Why? Because it's a ... Hook ("weird magical meta-language")

A LOOK AT THE API

useState for local State in your functional component

```
export default function SettingsForm(props) {  
  const [ favColor, setFavColor ] = useState("red");  
  
  return <input value={favColor}  
           onChange={e => setFavColor(e.target.value)} />  
}
```

- **Noteable:** Return Value
 - what is this? Tuple! (btw: I think Tuples will make it to JavaScript)
 - unusual (yet), but elegant, allows to name my variables as I want them to
- **Noteable:** initial value, used only once even if this method is run on each render
 - Why? Because it's a ... Hook ("weird magical meta-language")
- **Noteable:** setter-Function leads to re-render
 - Why? Because it's a ... Hook ("weird magical meta-language")

A LOOK AT THE API

useState another try

```
export default function SettingsForm(props) {  
  const [ easy, setEasy ] = useState(true);  
  
  function handleClick() {  
    setEasy(!easy);  
  
    console.log(easy);  
    // what is acutally easy here after first button click? 🤔  
  }  
  
  return <button onClick={handleClick}>Do it</button>  
}
```

- **Noteable:** setting a state is still "defered" or "async"
Might or might not be obvious

A LOOK AT THE API

useEffect for sideeffects

```
export default function SettingsForm(props) {  
  useEffect( () => { start(); return () => stop() }, [] )  
  
  return <...>  
}
```

- **Noteable:** Everything
 - what is a sideeffect? API call, DOM API, console.log?
 - what is 2nd argument ("dependency array")
 - unusual way to "limit" call to functions
 - btw: what does [] semantically mean?
Let's call the experts!

A LOOK AT THE API

useEffect for sideeffects



Andrew Clark @acdlite · 27. Nov. 2018

Intentionally underspecifying dependencies passed to `useEffect`/`useMemo` is the `any` of React Hooks.

You think you're being clever by passing an empty array, but you're probably wrong.

5

10

59



<https://twitter.com/acdlite/status/1067541310377123840>

A LOOK AT THE API

useEffect for sideeffects



Andrew Clark @acdlite · 27. Nov. 2018

Intentionally underspecifying dependencies passed to `useEffect`/`useMemo` is the `any` of React Hooks.

You think you're being clever by passing an empty array, but you're probably wrong.

5

10

59



tom @tomjfinney · 27. Nov. 2018

Why does the react docs for hooks then mention passing an empty array if you desire the effect to only be ran on mount and cleanup

2



2



A LOOK AT THE API

useEffect for sideeffects



Andrew Clark @acdlite · 27. Nov. 2018

Intentionally underspecifying dependencies passed to `useEffect`/`useMemo` is the `any` of React Hooks.

You think you're being clever by passing an empty array, but you're probably wrong.

5

10

59

↑



tom @tomjfinney · 27. Nov. 2018

Why does the react docs for hooks then mention passing an empty array if you desire the effect to only be ran on mount and cleanup

2

10

2

↑



Andrew Clark @acdlite · 27. Nov. 2018

Where do the docs say that? If they do, then they're wrong :(

1

10

1

↑



tom @tomjfinney · 27. Nov. 2018

[reactjs.org/docs/hooks-ref...](https://reactjs.org/docs/hooks-reference.html#the-last-bit-of-that-block) the last bit of that block

A LOOK AT THE API

useEffect for sideeffects



Dan Abramov
@dan_abramov

Antwort an @albertgao @mpocock1 und 2 weitere

Sorry, I don't think we have a broad agreement on the team about it either, and that's why our messaging is inconsistent. There is also some temptation to see Hooks+Suspense as one package, and ignore shortcomings of "fetch in effect" scenario that's common today. We'll fix.

[Tweet übersetzen](#)

11:55 nachm. · 28. Nov. 2018 · [Twitter Web Client](#)

2 „Gefällt mir“-Angaben

https://twitter.com/dan_abramov/status/1067914987753091074

A LOOK AT THE API

useEffect for sideeffects



Dan Abramov

Working on it.

@dan_abramov · 29. Nov. 2018



Dan Abramov

@dan_abramov · 28. Nov. 2018

Antwort an @albertgao @mpcock1 und 2 weitere

Sorry, I don't think we have a broad agreement on the team about it either, and that's why our messaging is inconsistent. There is also some temptation to see Hooks+Suspense as one package, and ignore shortcomings of "fetch in effect" scenario that's common today. We'll fix.



https://twitter.com/dan_abramov/status/1067917403709943808?s=20

A LOOK AT THE API

useEffect for sideeffects



Dan Abramov

Working on it.

@dan_abramov · 29. Nov. 2018



Dan Abramov

@dan_abramov · 28. Nov. 2018

Antwort an @albertgao @mpcock1 und 2 weitere

Sorry, I don't think we have a broad agreement on the team about it either, and that's why our messaging is inconsistent. There is also some temptation to see Hooks+Suspense as one package, and ignore shortcomings of "fetch in effect" scenario that's common today. We'll fix.



https://twitter.com/dan_abramov/status/1067917403709943808?s=20

- end of thread -

USING HOOKS

REACT HOOKS

USING HOOKS

Using Hooks: this is simple...

```
import React, { useState } from "react";

export default function SettingsForm(props) {

  const [ favColor, setFavColor ] = useState("blue");
  return <input value={favColor} onChange={...} />
}
```

USING HOOKS

Using Hooks: let's add context...

```
import React, { useState, useContext } from "react";

export default function SettingsForm(props) {

  const login = useContext(LoginContext);

  const [ favColor, setFavColor ] = useState("blue");
  return <input value={favColor} onChange={...} />
}
```

USING HOOKS

Using Hooks: and now... boom!

```
import React, { useState, useContext } from "react";

export default function SettingsForm(props) {

  const login = useContext(LoginContext);

  if (login.loggedIn) {
    return <Redirect to="/login" />
  }

  const [ favColor, setFavColor ] = useState("blue");
  return <input value={favColor} onChange={...} />
}
```

USING HOOKS

Using Hooks: and now... boom!

```
import React, { useState, useContext } from "react";

export default function SettingsForm(props) {

  const login = useContext(LoginContext);

  if (login.loggedIn) {
    return <Redirect to="/login" />
  }

  const [ favColor, setFavColor ] = useState("blue");
  return <input value={favColor} onChange={...} />
}
```

Why? Because it's a ... Hook ("weird magical meta-language")
Hooks must always be called in the same order

USING HOOKS

Linter: saves us!

- eslint-plugin-react-hooks is really good and helpful

```
export default function SettingsForm(pro  
  import useState  
  
  const login = useContext(LoginContext)  
  
  if (login.loggedIn) {  
    return <Redirect to="/login" />  
  }  
  
  const [ themeColor, setThemeColor ] = useState("blue");  
  return <input value={themeColor} onChange={e=>setThemeColor(e.target.va  
}
```

import useState

Returns a stateful value, and a function to update it.

@version — 16.8.0

@see — <https://reactjs.org/docs/hooks-reference.html#usestate>

React Hook "useState" is called conditionally. React Hooks must be called in the exact same order in every component render. Did you accidentally call a React Hook after an

USING HOOKS

...and another one: useHistory from React Router

```
import { useHistory } from "react-router-dom";

export default function SettingsForm(props) {

  function saveAndRedirect() {
    saveSettings().then(
      () => useHistory().push("/home") ← 😱
    );
  }

  return <•••><button onClick={saveAndRedirect}>Save</button><•••>
}
```

USING HOOKS

...this works

```
import { useHistory } from "react-router-dom";

export default function SettingsForm(props) {
  const history = useHistory();
  function saveAndRedirect() {
    saveSettings().then(
      () => history.push("/home")
    );
  }
  return <•••><button onClick={saveAndRedirect}>Save</button><•••>
}
```



Might not be big difference, but...

USING HOOKS

Might not be a big difference, but...

- we may have lots of unnecessary `useHistory` calls (or other Hooks)
- you have to know where you can use Hooks
- forces you to structure your code in exactly this way
- it's not "standard javascript"

Do you remember why React doesn't add a template language?

- To allow us the use of our "favorite" language: JavaScript
 - no need to learn a new language...

Does that mean Hooks (or React) are evil?

- No, but... they have their "price" (as classes have)
- It's "rethinking" again

Consequences

OF USING HOOKS

CONSEQUENCES

Can Custom Hooks replace existing patterns?

- Custom Hooks are another way for reusable logic
- HOCs?
- Render Properties?
- Would be good imho
 - esp. HOCs hard to understand
- **But...**

CONSEQUENCES

Example: React Router **withRouter** HOC

```
import { withRouter } from "react-router-dom";

function SettingsForm( { history } ) {

    function saveAndRedirect() {
        saveSettings().then( () => history.push("/home") );
    }

    return <•••><button onClick={saveAndRedirect}>Save</button><•••>
}

export default withRouter(SettingsForm);
```

A LOOK AT THE API

Example: React Router `useHistory` Custom Hook

```
import { useHistory } from "react-router-dom";

export default function SettingsForm( props ) {
  const history = useHistory();
  function saveAndRedirect() {
    saveSettings().then( () => history.push("/home") );
  }

  return <•••><button onClick={saveAndRedirect}>Save</button><•••>
}

}
```

Difference:

- Only one component (vs SettingsForm and HOC'd SettingsForm)
- SettingsForm "knows" about Router, it's not "just a prop" anymore

A LOOK AT THE API

Another one: React Router (with render prop)

```
// App.js
<Route path="/settings/:id"
      render={({match}) => <SettingsForm settingsId={match.params.id} />}
```



```
// SettingsForm.js
export default function SettingsForm( {settingsId} ) {

  // do something with settingsId
  return ...;
}
```

Noteable:

- SettingsForm does not know anything about Router
- Routing "Logic" (Params, Routes, ...) are at *one* place (good imho)

A LOOK AT THE API

Another one: React Router with new Route API and useParams

```
// App.js
<Route path="/settings/:id"><SettingsForm /></Route>
```

new Router
5.2 API
no render prop anymore!



A LOOK AT THE API

Another one: React Router with new Route API and useParams

```
// App.js
<Route path="/settings/:id"><SettingsForm /></Route> ← new Router  
5.2 API  
no render-Prop anymore

// SettingsForm.js
import { useParams } from "react-router-dom";

export default function SettingsForm( ) {
  const { settingsId } = useParams();

  // do something with settingsId
  return •••;
}
```

Noteable:

- SettingsForm knows about Router API and Routing "Logic" (which Params)
- What about "Colocation"?

A LOOK AT THE API

What about this one?

(from: <https://twitter.com/Wolverineks/status/1177818104048472065>)

```
function RouterContext({ children }) {  
  return children({  
    history: useHistory(),  
    params: useParams(),  
    ...  
  });  
}
```

A LOOK AT THE API

What about this one?

(from: <https://twitter.com/Wolverineks/status/1177818104048472065>)

```
function RouterContext({ children }) {  
  return children({  
    history: useHistory(),  
    params: useParams(),  
    ...  
  });  
}  
  
<Route path="/settings/:id">  
  <RouterContext>  
    {({ params }) => <SettingsForm settingsId={params.id} />}  
  </RouterContext>  
</Route>
```

A LOOK AT THE API

What about this one?

(from: <https://twitter.com/Wolverineks/status/1177818104048472065>)

```
function RouterContext({ children }) {  
  return children({  
    history: useHistory(),  
    params: useParams(),  
    ...  
  });  
}  
  
<Route path="/settings/:id">  
  <RouterContext>  
    {({ params }) => <SettingsForm settingsId={params.id} />}  
  </RouterContext>  
</Route>
```

Noteable: welcome back, render properties!



CONSEQUENCES

What does that mean?

- Hooks provide new ways for structuring code, architecture etc
- But they come with a price
 - nothing comes for free, nothing special here
- We probably we will see new patterns, best-practices evolving
 - React development won't become boring

A LOOK AT THE API

Example: Redux `useDispatch` and `useSelector` instead of `connect`

```
import { useDispatch, useSelector } from "react-redux";

export default function SettingsForm(props) {
  const favColor = useSelector(state => state.theme.favColor);
  const dispatch = useDispatch();

  const setNewColor = (r,g,b) => dispatch(actions.setNewColor(r,g,b));

  return <•••><ColorPicker onSet={setNewColor}/><•••>
}
```

A LOOK AT THE API

Example: Redux `useDispatch` and `useSelector` instead of `connect`

```
import { useDispatch, useSelector } from "react-redux";

export default function SettingsForm(props) {
  const favColor = useSelector(state => state.theme.favColor);
  const dispatch = useDispatch();

  const setNewColor = (r,g,b) => dispatch(actions.setNewColor(r,g,b));

  return <•••><ColorPicker onSet={setNewColor}><•••>
}
```

- Consequences:
 - We now have only one component, have seen that already
 - the component is bound to Redux, have seen that already

A LOOK AT THE API

Example: Redux `useDispatch` and `useSelector` instead of `connect`

```
import { useDispatch, useSelector } from "react-redux";

export default function SettingsForm(props) {
  const favColor = useSelector(state => state.theme.favColor);
  const dispatch = useDispatch();
  "forces" re-rendering of the ColorPicker component
  const setNewColor = (r,g,b) => dispatch(actions.setNewColor(r,g,b));
}

return <•••><ColorPicker onSet={setNewColor}/><•••>
}
```

- Consequences:
 - We now have only one component, have seen that already
 - the component is bound to Redux, have seen that already
 - It has different rendering behaviour (not related to redux)

A LOOK AT THE API

We can fix this:

```
import { useDispatch, useSelector } from "react-redux";

export default function SettingsForm(props) {
  const favColor = useSelector(state => state.theme.favColor);
  const dispatch = useDispatch();

  const setNewColor = React.useCallback(
    (r,g,b) => dispatch(actions.setNewColor(r,g,b)),
    [ dispatch ]
  );

  return <•••><ColorPicker onSet={setNewColor}/><•••>
}
```

A LOOK AT THE API

We can fix this:

```
import { useDispatch, useSelector } from "react-redux";

export default function SettingsForm(props) {
  const favColor = useSelector(state => state.theme.favColor);
  const dispatch = useDispatch();

  const setNewColor = React.useCallback(
    (r,g,b) => dispatch(actions.setNewColor(r,g,b)),
    [ dispatch ] ← remember the dependency array?
  );

  return <•••><ColorPicker onSet={setNewColor}/><•••>
}
```

- "Nice!" (Fortunately we only have *one* callback function here...)

A LOOK AT THE API

Again: is this really a problem?

- In most cases not as performance might be good enough to re-render all the time, so useCallback ist not a must
- I wonder how many CPU engery is wasted due to billions of unneccessary re-renders in React Apps world wide 😎
- But this is – esp. for Beginners – not easy to understand (call me a Beginner)

CONSEQUENCES

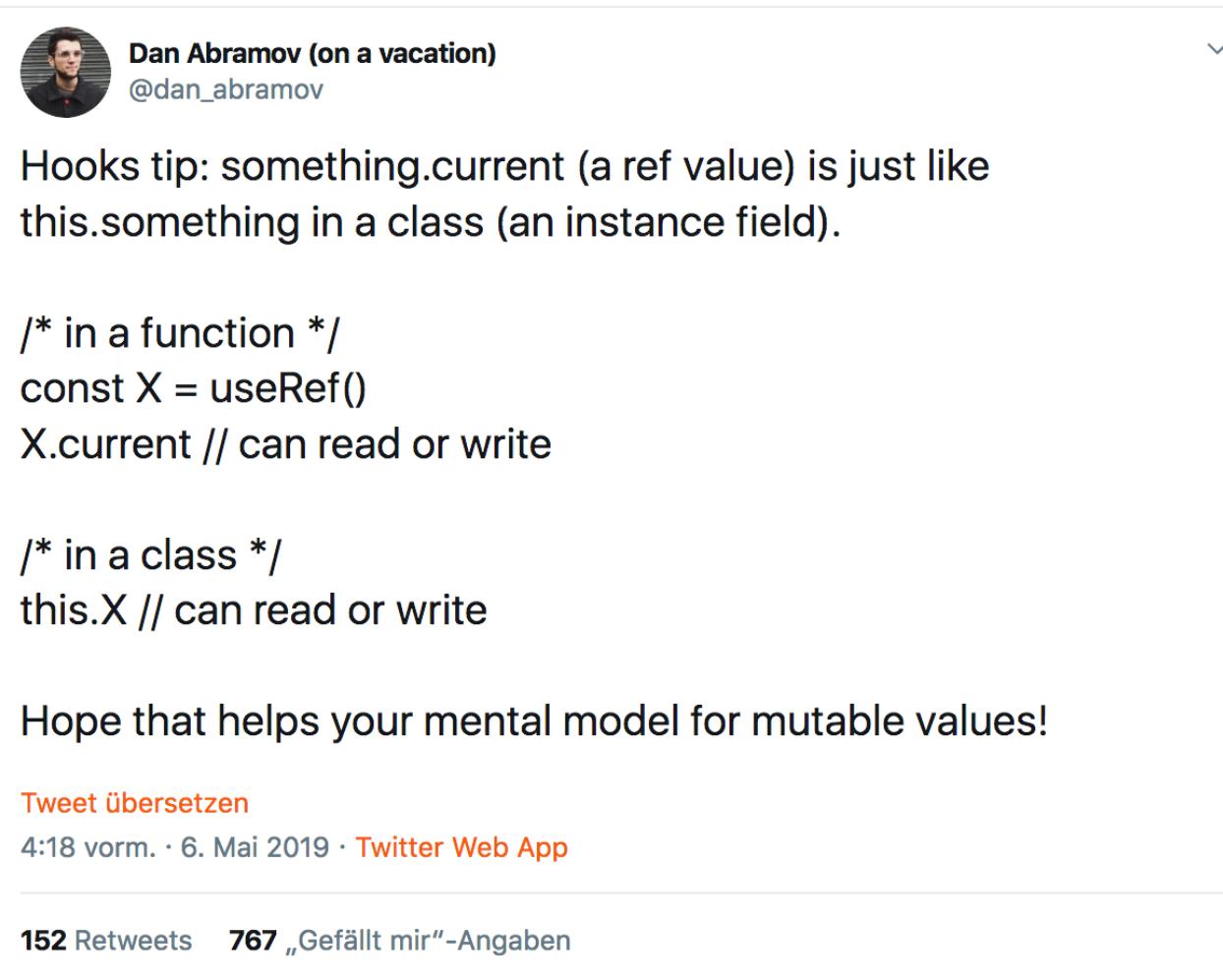
Let's have a look at the component lifecycle **TODO USEREF**

- we want to cancel the request

```
export default function App() {  
  
  const [ running, setRunning ] = useState(true);  
  
  useEffect( () => {  
    const id = setTimeout( () => setRunning(false), 500);  
    setRunning(true);  
  
    return () => clearTimeout(id);  
  }, [ running ])  
  
  return (<>Running: {running.toString()}</>);  
}
```

A LOOK AT THE API

useRef



Dan Abramov (on a vacation)
@dan_abramov

Hooks tip: something.current (a ref value) is just like this.something in a class (an instance field).

```
/* in a function */  
const X = useRef()  
X.current // can read or write
```

```
/* in a class */  
this.X // can read or write
```

Hope that helps your mental model for mutable values!

[Tweet übersetzen](#)
4:18 vorm. · 6. Mai 2019 · Twitter Web App

152 Retweets 767 „Gefällt mir“-Angaben

CONSEQUENCES

A look back to...

```
export default class App extends React.Component {  
  state = { running: false }  
  
  componentDidMount() {  
    this.interval = ...  
  }  
  
  componentWillUnmount() {  
    clearInterval(this.interval);  
  
  }  
  render() {  
    return <button onClick={() =>  
      clearInterval(this.interval)}>...</button>  
  }  
}
```

CONSEQUENCES

A look back to...

```
export default class App extends React.Component {  
  state = { running: false }  
  
  componentDidMount() {  
    this.interval = ...  
  }  
  
  componentWillUnmount() {  
    clearInterval(this.interval);  
  
  render() {  
    return <button onClick={() =>  
      clearInterval(this.interval)}>...</button>  
  }  
}  
• it's a class, but: clear lifecycle method
```

TypeScript

WITH HOOKS

TYPESCRIPT

Type Inference with useState works very good!

```
function useAppSelector<R>(fn: (state: AppState) => R): R {  
  return useSelector(fn)  
}
```

```
export default function SettingsForm(props) {  
  const favColor = useAppSelector(state => state.favColor);  
  
  return ...;  
}
```

TYPESCRIPT

...custom hook for typing useSelector?

```
function useAppSelector<R>(fn: (state: AppState) => R): R {  
  return useSelector(fn)  
}
```

```
export default function SettingsForm(props) {  
  const favColor = useAppSelector(state => state.favColor);  
  
  return ...;  
}
```

Summary

- Hooks are kind of a new (meta-)language
 - they solve problems
 - but they raise new ones
- Through new ways we even have more ways to structure our code
 - React was always about flexibility... some like, some don't
- I don't like hypes and that's probably the real reason I'm sceptical



Thanks a lot!

Slides: <https://nils.buzz/react-meetup-hooks>