

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**



2nd edition out in dec!

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

REACT HOOKS

Anyone NOT knowing what React Hooks are?

REACT HOOKS

Anyone NOT knowing what React Hooks are?

(My assumption: almost noone)

REACT HOOKS

Anyone NOT knowing what React Hooks are?

Hooks 2 Minute intro

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

REACT HOOKS

Anyone NOT knowing what React Hooks are?

Hooks 2 Minute intro

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

REACT HOOKS

Anyone NOT knowing what React 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

One Year

React Hooks

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)

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

One Year of Hooks...

...it seems, Hooks are *the new way* to go for React Apps
(Vue has them now, too btw)

But...

React Hooks

Good or **Evil?**

GOOD OR EVIL?

Who likes Hooks?

GOOD OR EVIL?

Who likes Hooks?

(My assumption: almost everyone)

GOOD OR EVIL?

Who likes Hooks?

(My assumption: almost everyone)

Who dislikes Hooks?

GOOD OR EVIL?

Who likes Hooks?

(My assumption: almost everyone)

Who dislikes Hooks?

(My assumption: almost noone)

GOOD OR EVIL?

Let's hear some more...



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/use-substate)

[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



"Unsure..."

Me



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>

SO... HOW ABOUT THIS?



Tom Dale
@tomdale



Old React was simple and fun and “just JavaScript.” New React is more powerful and more correct and better all around

[Tweet übersetzen](#)

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

4 Retweets **73** „Gefällt mir“-Angaben

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

SO... HOW ABOUT THIS?



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>

SO... HOW ABOUT THIS?

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 ] = React.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 ] = React.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 ] = React.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

A timer...

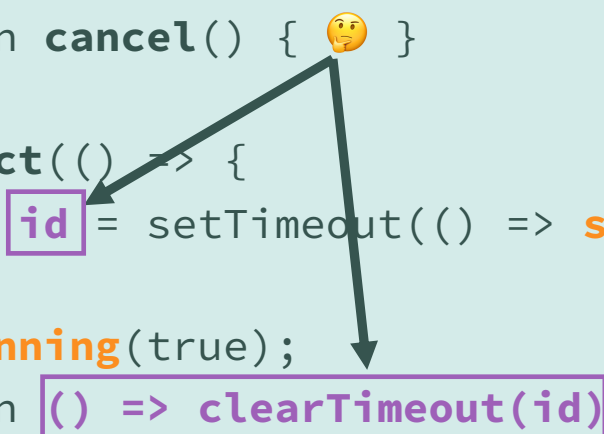
```
export default function App() {  
  const [running, setRunning] = useState(false);  
  
  useEffect(() => {  
    const id = setTimeout(() => setRunning(false), 2000);  
  
    setRunning(true);  
    return () => clearTimeout(id);  
  }, []);  
  
  return <button onClick={cancel}>Running: {running.toString()}</button>;  
}
```

A LOOK AT THE API

useRef: for fixing problems introduced by Hooks

- We want to cancel the running timeout
- Somehow need to get access to the cleanup function or the id

```
export default function App() {  
  const [running, setRunning] = useState(false);  
  
  function cancel() { 🤔 }  
  
  useEffect(() => {  
    const id = setTimeout(() => setRunning(false), 2000);  
  
    setRunning(true);  
    return () => clearTimeout(id);  
  }, []);  
  
  return <button onClick={cancel}>Running: {running.toString()}</button>;  
}
```



The diagram illustrates the flow of the `id` variable. An arrow points from the `id` variable in the `useEffect` hook to the `clearTimeout(id)` call in the cleanup function. Another arrow points from the `cancel` function to the `clearTimeout(id)` call, indicating that `cancel` is responsible for invoking the cleanup function.

A LOOK AT THE API

useRef: Remember Class Components?



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
```

← "weird magical meta-language"

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

← JavaScript Standard

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

https://twitter.com/dan_abramov/status/1125223181701263360

A LOOK AT THE API

useRef: for fixing problems introduced by Hooks

- We want to cancel the running timeout
- Somehow need to get access to the cleanup function or the id

```
export default function App() {  
  const [running, setRunning] = useState(false);  
  const timerRef = useRef();  
  function cancel() { clearTimeout(timerRef.current); }
```

```
  useEffect(() => {  
    const id = setTimeout(() => setRunning(false), 2000);  
    timerRef.current = id;  
    setRunning(true);  
    return () => clearTimeout(id);  
  }, []);
```

```
  return <button onClick={cancel}>Running: {running.toString()}</button>;  
}
```

USING HOOKS

HOW DOES IT LOOK TO USE 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={...} />
}
```



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

USING HOOKS

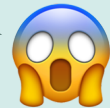
...and another one: useHistory from React Router

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

```
export default function SettingsForm(props) {
```

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

"weird magical meta-language"



```
  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...

- you have to know where you can use Hooks
- forces you to structure your code in exactly this way
- it's not "standard javascript"
- we even have/need a linter for Rules of Hooks

USING HOOKS

Might not be a big difference, but...

- you have to know where you can use Hooks
- forces you to structure your code in exactly this way
- it's not "standard javascript"
- we even have/need a linter for Rules of Hooks

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

USING HOOKS

Might not be a big difference, but...

- you have to know where you can use Hooks
- forces you to structure your code in exactly this way
- it's not "standard javascript"
- we even have/need a linter for Rules of Hooks

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

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

USING HOOKS

Might not be a big difference, but...

- you have to know where you can use Hooks
- forces you to structure your code in exactly this way
- it's not "standard javascript"
- we even have/need a linter for Rules of Hooks

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

- To enable us to use 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
 - Replacement for HOCs?
 - Replacement for Render Properties?
- **But...**

CONSEQUENCES

Example: "old" React Router (with **render prop**)

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

CONSEQUENCES

Example: "old" 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)

CONSEQUENCES

Example: 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!

CONSEQUENCES

Example: 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"?

CONSEQUENCES

What about this one?

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

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

CONSEQUENCES

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>  
</Router>
```

Noteable: welcome back, render properties! 🤖
But at least SettingsForm is Router-free

CONSEQUENCES

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

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

CONSEQUENCES

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();
  might "force" 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
- **But:** it also has different rendering behaviour (compared to connect)

CONSEQUENCES

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}/><...>
}
```

CONSEQUENCES

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...)
```

CONSEQUENCES

Is this really a problem?

- This problem is not related to Redux only
- In most cases not as performance might be good enough to re-render all the time, so useCallback (and useMemo) is not a must
- But this is – esp. for beginners – not easy to understand (call me a beginner)
- BTW: I wonder how many CPU engery is wasted due to billions of unneccessary function executions in React Apps world wide 🧐

One Year

React Hooks

Summary

ONE YEAR REACT HOOKS

Summary

ONE YEAR REACT HOOKS

Summary

- **If you're already using React, use Hooks.**
 - They will stay. It's the "New React". Classes will lose their relevance.
 - For (experienced) React developers they are a good innovation
 - We will see how Hooks-based architectures evolve

ONE YEAR REACT HOOKS

Summary

- **If you're already using React, use Hooks.**
 - They will stay. It's the "New React". Classes will lose their relevance.
 - For (experienced) React developers they are a good innovation
 - We will see how Hooks-based architectures evolve
- **However:**
 - While technically standard JavaScript functions, their usage is not
 - They are more like an own "magical meta-language" for React
 - Selling point "you only have to know JavaScript to learn React" is not valid anymore (if it has ever been)

ONE YEAR REACT HOOKS

Summary

- **If you're already using React, use Hooks.**
 - They will stay. It's the "New React". Classes will lose their relevance.
 - For (experienced) React developers they are a good innovation
 - We will see how Hooks-based architectures evolve
- **However:**
 - While technically standard JavaScript functions, their usage is not
 - They are more like an own "magical meta-language" for React
 - Selling point "you only have to know JavaScript to learn React" is not valid anymore (if it has ever been)
- **For people not familiar with React/new to React**
 - Hooks might scare people
 - As React becomes a little less "JS Standard", People might consider alternatives, like Web Components (Standard!)
 - We're still far away from "React Best Practices"

NILS HARTMANN
<https://nilshartmann.net>



Thanks a lot!

What do you think?

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

NILS@NILSHARTMANN.NET