React Router 4 (beta 8) won't render components if using redux connect #4671

New issue

Closed **klase** opened this issue on 9 Mar · 32 comments

klase commented on 9 Mar • edited

Version

4.0.0-beta.8

If a component containing `<Route>` components is exported with a Redux connect wrapper non of the components mapped to those routes components are rendered. Let's say you have

```
<BrowserRouter>
 <App />
</BrowserRouter>
```

and App looks like:

```
class App extends Component {

  render() {
    return (
      <div>
          <Route exact path="/" component={Home} />
          <Route path="/Test1" component={Test1}/>
          <Route path="/Test2" component={Test2} />
      </div>
    );
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(App);
```

then none of the components get rendered on route changes. If you remove the connect wrapper all works as expected.

41 8

sdcooke commented on 9 Mar

Navigating isn't changing the rendered route for me - I think it's related. componentWillReceiveProps isn't being called on the Route which is likely to be caused by
https://facebook.github.io/react/docs/context.html#updating-context

peter-mouland commented on 9 Mar • edited

I don't know if you have the same problem i did, but I had fixed `react-router-dom` to beta.7 without using a map / lock file, which meant react-router-dom was fixed to v4-beta.7 but it had downloader `react-router` v4-beta.8. Once i updated, everything worked again

hihuz commented on 9 Mar • edited

Hey klase, I may be mistaken but I think this is exactly the problem I had these previous days.
I was used to the v4 alpha API where doing what you shown above worked fine.

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

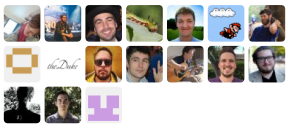
No milestone

Notifications

Subscribe

You're not receiving notifications from this thread.

17 participants



In the latest beta API however they kind of went back to their previous way of doing things (that is using react context exclusively to communicate changes), connect() prevents that from working by ways of shouldComponentUpdate.

They have provided again in the beta the withRouter() HOC that solves this nicely, just import it and do

```
export default withRouter(connect(mapStateToProps, mapDispatchToProps)(App));
```

This should work just fine after that.
Note that you would also need to wrap any redux connected component that has router components inside it with a withRouter(). That means if you have router Links in your redux connected Nav component for example, you would need to wrap it with withRouter() as well.

Hope this helps.

Here is a helpful discussion over on the react-redux repo

47 1 8 2

timdorr commented on 9 Mar Collaborator

There's a guide to help with this: https://github.com/ReactTraining/react-router/blob/v4/packages/react-router/docs/guides/blocked-updates.md

I'm looking into removing this issue from react-redux's end soon.

3 24 9

timdorr closed this on 9 Mar

sdcooke commented on 9 Mar

@timdorr if I understand correctly, are you saying that react-router simply isn't going to be compatible with shouldComponentUpdate? react-redux isn't the only thing that uses that function. I'd rather not have to use that workaround every time I've used `React.PureComponent`

timdorr commented on 9 Mar Collaborator

It's not *not* compatible with `sCU`, but there are caveats to using it. That's what the guide is for.

hihuz commented on 9 Mar

Hmm sorry Tim, I just wanted to help out since I was in a similar situation recently but my comment was probably not helpful in the end.
The last thing I wanted to do was to waste your time (or the time of any other collaborator), I know you have a lot to do, and your work on these projects that I use a lot is greatly appreciated.

klase commented on 9 Mar

@hihuz your comment was very helpful to me, appreciate it.

1

timdorr commented on 9 Mar Collaborator

@hihuz No worries. Believe me, I have seen some terrible comments 😄

2

epeli commented on 14 Mar • edited Contributor

Hi!

I'm having trouble working around this issue like suggested in blocked-updates.md. It seems that a pure component at any level of the component tree will prevent updates for react-router. I'm using react-router 4.0.0.

I have multiple connect()s or other pure components all over my app so it's not very feasible go and wrap them with `withRouter()`. Most of them have otherwise nothing to do with react-router.

I wonder why react-router uses context **values** to propagate the route changes to begin with? React Redux for example circumvents this issue by listening the store on each connect() so nesting it under other pure components does not matter. So why not to listen to the `history` object on every `Route` component?

As a proof of concept I created this workaround (mentioned previously in #4676 (comment)):

```
class ListeningRoute extends React.Component {
    componentWillMount() {
        this.unlisten = this.props.history.listen(location => {
            this.setState({location});
        });
    }

    componentWillUnmount() {
        this.unlisten();
    }

    render() {
        return <Route {...this.props}  location={this.state.location} />;
    }
}
ListeningRoute = withRouter(ListeningRoute);
```

Seems to solve the issue very nicely in my app.

UPDATE: This has some issues. Read on...

UPDATE2: React Redux wrapper https://gist.github.com/epeli/ddd916d568ed2c1b30a7c714effabd01

2

epeli referenced this issue on 14 Mar

"withRouter" doesn't rerender elements when state is changed #4676 Closed

sdcooke commented on 14 Mar

We're doing a similar thing to **@epeli** - fortunately we were already wrapping every `Route` so we only have to make the change in one place. The suggested workaround isn't an option for us - you can't expect developers on a large codebase to remember to add an unrelated higher order component every time `PureComponent` or `shouldComponentUpdate` is used.

I'm very grateful for react-router and acknowledge that I don't necessarily know best but it does seem odd to stick to a feature that the react documentation says "Don't do it" whilst forcing a workaround on a useful and documented feature.

This was referenced on 14 Mar

router stops working if component is wrapped with react-redux connect() #4744 Closed

React Router v4 react-bootstrap/react-router-bootstrap#186 Closed

lourd commented on 16 Mar • edited

Throwing in my $0.02 on the pain of this use case as well. I upgraded my react-router-dom dependency from 4.0.0-beta.5 to 4.0.0 and all of my routes stopped changing when I clicked on links. Came down to the custom, redux-connected route component that we use.

The relevant snippet:

```
@withRouter // New addition needed to make component update on location change
@connect(mapStateToProps)
export default class SKRoute extends React.Component {
  static propTypes = {
    component: PropTypes.func.isRequired,
```

```
    restrictTo: PropTypes.oneOf(Object.values(Permissions)),
    authorization: PropTypes.bool.isRequired,
    animation: PropTypes.string,
    enterAnimationLength: PropTypes.number.isRequired,
    leaveAnimationLength: PropTypes.number.isRequired,
    reroute: PropTypes.string,
  }

  static defaultProps = {
    animation: animations.fadeInOut,
    enterAnimationLength: 250,
    leaveAnimationLength: 250,
  }

  render() {
    const {
      reroute,
      authorization,
      component: Component,
      animation,
      enterAnimationLength,
      leaveAnimationLength,
      ...props
    } = this.props
    return (
      <Route {...props}>
        {({ match, location, history }) => { // using implicit `children` prop
          if (!match && !authorization) return null
```

Had to use the `withRouter` decorator to make it work. And it does! But boy is the component tree inefficient

```
▼<withRouter(Connect(SKRoute)) key="/login" path="/login" component=Connect(
  ▼<Route render=render()>
    ▼<Connect(SKRoute) path="/login" component=Connect() restrictTo="notLogge
      ▼<SKRoute path="/login" component=Connect() restrictTo="notLoggedIn"…>
        ▼<Route path="/login" restrictTo="notLoggedIn" match={path: "/", url:
```

Why did the location reactivity mechanism switch from history subscriptions back to context?

As always, thanks for y'alls hard work!

pshrmn commented on 16 Mar — Collaborator

@lourd This article details why subscriptions were removed.

As far as your component tree goes, it is difficult to say without seeing what is rendering your `<SKRoute>`s, but if it has access to the location, you can just pass that as a prop to all of them. The important thing is just that the component gets the updated location object when it updates.

```
const Parent = withRouter(({ location }) => (
  <div>
    <SKRoute path='/login' location={location} ... />
    <SKRoute path='/forgot' location={location} ... />
    <SKRoute path='/signup' location={location} ... />
    <SKRoute path='/terms' location={location} ... />
  </div>
))
```

lourd commented on 16 Mar • edited

Oh wow, you've got a whole article ready! Now that's some support, haha. Thank you!

I resolved my issue by breaking up `SKRoute` into two components. I made the connected component a sub-component of `<Route>`, receiving the route props and eliminating the need for `withRouter`.

```
function SKRoute(props) {
  return (
    <Route
      path={props.path}
      exact={props.exact}
      strict={props.strict}
    >
      {routeProps => <EnhancedRoute {...props} {...routeProps} />}
    </Route>
```

```
  )
}
```

Now the component tree only includes one `Route`.

```
▼<SKRoute key="/" path="/" exact=true component=Connect()…>
  ▼<Route path="/" exact=true>
    ▼<Connect(EnhancedRoute) path="/" exact=true component=Connect()…>
      ▼<EnhancedRoute path="/" exact=true component=Connect()…>
```

Huzzah for composability!

1

epeli commented on 16 Mar • edited — Contributor

Nice article, thanks! It points out the flaws in my `ListeningRoute` workaround. A better way would be to pass the location prop via Redux store with `connect()`. It should not have the flaws but that's good only for Redux users :/

I wrote a response to the article explaining how React Redux does this:
https://medium.com/@esamatti/subscriptions-in-react-redux-do-work-547bf9d66aa3#.1alui01ky

pshrmn commented on 16 Mar — Collaborator

@epeli with redux, there is only one object that needs to be referenced, the store. With react router, there is only one history, but each `<Route>` also has its own `match` object that its children need to reference.

If react router implemented a subscription system, every `<Route>` would need to have its location-aware children subscribe to it. Then, when the `<Route>` updates, it would force update the subscribers. The context would need to be re-written so that children "pull" the context value from their parent, because they would not be able to rely on `this.context`.

(React Router does actually use a subscription system, but only through the `<Router>` component.)

A complicated subscription solution makes sense for redux because of the frequency of re-renders that the store might cause. Every time an action is dispatched to the redux store, the application needs to re-render. This may very well be 60 or more times per second. This means that react-redux has to minimize the scope of each update to maximize performance. It isn't as picky as mobx, but it still wants to limit what is updated based on what has changed since the last render.

When does react router require re-renders? When the location changes. That isn't something we have to worry about happening dozens of times a second (unless someone gets themselves in a redirect loop...). That isn't to say that react router doesn't care about efficiency, just that the tradeoff of a complicated subsription system doesn't justify any performance gains from only updating location-aware components. Instead, react router's concern is that every component that is location-aware re-renders using this new information. If we can ensure a full component tree re-render (i.e., what React does naturally) by getting components with shallow prop checks to re-render when the location changes, then we can be sure that every location-aware component updates properly.

1

epeli commented on 16 Mar • edited — Contributor

> **@epeli** with redux, there is only one object that needs to be referenced, the store. With react router, there is only one history, but each also has its own match object that its children need to reference.

Why not wrap them all into a single container and get it from there? Just one key for the context. No need to pass them individually. Or just regenerate them when `Route` mounts from the `history` object?

You are right about the performance tradeoff but you are also doing another more concerning tradeoff: Compatiblity with the rest of the React ecosystem. `React.PureComponent` is part of the React core and React Router does not work with it unless React Router specific hack is added to it. That's not good tradeoff in my opinion.

2

pshrmn commented on 16 Mar — Collaborator

When you use a `PureComponent`, you are effectively stating "do not update me unless my props or state change". To that end, I wouldn't call passing it a prop that changes to ensure it updates a hack. Instead, I would probably question why the component is "pure". The context isn't supposed to be magic, just convenient.

epeli commented on 16 Mar — Contributor

> When you use a PureComponent, you are effectively stating "do not update me unless my props or state change".

But when I put a `<Route>` somewhere I feel like I'm saying "but as exception under this always render when the route changes"

> Instead, I would probably question why the component is "pure".

As stated before that's not always the choice of the user. Some 3rd party component might make that decision for you. Also in our code bases we would have to put the location prop into dozens of components using React Redux `connect()` which otherwise have no references to React Router.

1

pshrmn commented on 16 Mar — Collaborator

You don't have to render `<Route>`s everywhere. You only need to use `withRouter` / `<Route>` when the parent component that creates the element does not have access to the `location` through its props.

```
class Picky extends PureComponent {...}

// when there is no location to reference, we need to use a
// <Route> to get access
<Router>
  <Route component={Picky} />
</Router>

// however, if the component that creates the Picky element has the
// location, we do not need to use a <Route>/withRouter
<Router>
  <Route path='/some-place' render={({ location }) => (
    // we have location in scope, so we can just pass it
    <Picky location={location} />
  )}/>
</Router>
```

I'm not going to try to convince you that this is convenient, but imagine if `context` didn't exist. Then, you would have to pass these props to *every* component no matter which type of component you used. Perhaps future iterations of the context model will address the issue and this will no longer be necessary, but for the time being I think that this is a good solution.

sdcooke commented on 17 Mar

I agree with **@epeli** - it's about the developer experience. I feel like the suggested solution is fine for small applications with a few developers who all know the quirks of context but things get painful quickly when you have hundreds of components. We have `<Route>`s that are nested quite a few layers down so it's easy to inject a pure component between them without realising what you've done. I feel like the least error-prone ways to use `react-router` as it stands are to either always pass location and match as props and act as if context weren't being used or to wrap all of the `react-router` components (which is what we've opted for, since we had already wrapped them to centralise our urls).

I've forked `react-router` so I can have a play with a subscription-based system - I expect there's a high chance I'll run into the issues that sparked the decision to ditch subscriptions and then maybe I'll understand it but I've not yet understood why. For example, one of the reasons mentioned is "stale context" and I don't see why something like this couldn't work:

```
class Router extends React.Component {
    routerContext = {
        history: this.props.history,
        route: { ... }
```

```
    };

    getChildContext() {
        return {router: this.routerContext};
    }

    componentWillReceiveProps(nextProps) {
        this.routerContext.history = nextProps.history;
    }

    componentWillMount() {
        // listen for history changes then change this.routerContext.route and notify subscriptions
    }
}
```

I hope I'm not wasting anyone's time - I'm just trying to provide a different perspective based on my company's experiences.

maxdeviant referenced this issue on 20 Mar

Proposal: react-router-mobx #4785 Closed

theduke commented on 27 Mar • edited

This MUST be mentioned *very* prominently in the docs,
it's an issue many will run into.

I spent a long time scratching my head, trying to figure out why my routes aren't working.

@timdorr

57

selbekk commented on 27 Mar

Yeah this stopped my update to v4 as well. +1

epeli commented on 27 Mar Contributor

Here's a React Redux wrapper I'm currently using to workaround this issue:

https://gist.github.com/epeli/ddd916d568ed2c1b30a7c714effabd01

It keeps the location in the Redux store and uses `connect()` pass it to the React Router components `Switch` and `Route`.

jshthornton commented on 30 Mar • edited

I threw together this module if anyone else needs a connected route or switch whilst using redux
https://github.com/team-griffin/react-router-connected/tree/develop

https://www.npmjs.com/package/@team-griffin/react-router-connected

2

sethreidnz commented on 26 Apr • edited

I realise this is a closed issue but is there no interest in solving this problem? Or is it a feature? I really like React Router v4 but the way that I would have built every app in the past this feature would break them and I gave up and reverted to v3 for deadline pressure on my project atm.

The above solutions probably will work but feel like patches and or major mixing of concerns (`withRouter` for example).

7

gnoff commented on 8 May

Like all complex things sometimes the right answer is both. As you can tell from react-redux the api supports a few key optional properties (removing pure component logic for instance). I don't see why this library can't take a similarly pragmatic approach. Both sides can make compelling cases for whether subscription model is appropriate / desirable / safe / pointless / wasteful etc.... I personally am desperately in need of the subscription style but I understand that it has a tradeoff.

Is there any appetite for supporting this via configuration?

6 1 1

jkimau referenced this issue in **jkimau/react-todo** on 30 May

react router v4 won't render component properly #8 Closed

NickSpinosa commented on 14 Jun

hey if anyone is still blocked by this (I was for a solid 3 hours I hate to admit), a work around I found was to wrap my root component with `withRouter` this allows the location context to be passed to every component. It's not perfect, but it beats tracking every route down through your component tree. Also I found that using webpack dev server with hot reloading somehow bypasses this issue, so I didn't run into it until it came time to deploy. Definitely needs to be all over the docs. Thanks for all the help though everyone!

8

antonmedv commented on 28 Jun

This is really annoying. Every time need to google to find proper docs hidden in issues on github. :(

selbekk commented on 29 Jun

@antonmedv - this is actually covered in the documentation - and if it wasn't - why not try to rectify the perceived problem by submitting a PR?

1

Pinpickle commented 26 days ago • edited

I just battled with this for a few hours, reading up on context and the various arguments for why subscriptions are bad/good. I believe that removing subscriptions was a step back for this library, for the reasons mentioned in this thread and elsewhere. I'll add my 2 cents.

I think it's worth pointing out that the React docs have an extra warning for trying to update context:

> ## Updating Context
>
> Don't do it.
>
> React has an API to update context, but it is fundamentally broken and you should not use it.

This warning is in addition to the usual "context is not the React way, context has a volatile API and will probably change" warning.

They also link to this blog post, indicating their support that subscriptions are the correct way to handle changing context at the moment. This means that React Router is using a feature that React is actively saying "do not use", which is similar to using a deprecated feature in my eyes. If this dropped warnings at runtime instead of in documentation, it'd probably be dropped without a second thought.

From a quick glance, I believe implementing a working subscription system is possible, and the problems mentioned in **@pshrmn**'s article (stale context and re-renders) could be fixed with never changing context and some immutable objects or shallow compares. But I am not that clued-in on the inner-workings of the module so I can't speak much to that.

However, this line of the article stuck out:

> The above “solutions” are better defined as “tricks”. You can twist and turn your code to make it look like it works, but it will never actually feel right.

Sometimes, we have to write hacky solutions to problems (called "tricks" here). React's context feature is kind of broken so this is one of those cases. Abstraction lets us tuck those hacks away and not have to expose them. Removing subscriptions has exposed that hack and requires every developer to implement it in one way or another (albeit wrapped in a single function). We now have to add a "trick" to arbitrary components in our hierarchy, which definitely doesn't feel right to me (to use the words from the article).

Furthermore on abstraction, developer experience aside, it forces knowledge of implementation details that the consumers of these libraries don't need to know. Such as:

- `react-router` uses context to share information with components
- `react-redux` (or other libraries) use `shouldComponentUpdate` for optimisation

This leads to code that is difficult to understand, maintain and is likely to contain hard-to-find bugs. I believe that this is going against the vision of React Router in a big way, but in general is something we as developers are trying to avoid.

21

andriy-f commented 10 days ago • edited

I somehow managed to use react-router with redux without using withRouter. The solution was to do this

```
const App = () => (
   <SomeReduxComponent>
    <Routes /> // <-this contains react-router routes
   </SomeReduxComponent>

export default App
```

```
const SomeReduxComponent = (props) => (
   Markup...
   {props.children}
  Markup...
)

export default connect()(SomeReduxComponent)
```

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Comment