Version: 6.x

Navigation lifecycle

In a previous section, we worked with a stack navigator that has two screens (`Home` and `Details`) and learned how to use `navigation.navigate('RouteName')` to navigate between the routes.

An important question in this context is: what happens with `Home` when we navigate away from it, or when we come back to it? How does a route find out that a user is leaving it or coming back to it?

If you are coming to react-navigation from a web background, you may assume that when user navigates from route `A` to route `B`, `A` will unmount (its `componentWillUnmount` is called) and `A` will mount again when user comes back to it. While these React lifecycle methods are still valid and are used in react-navigation, their usage differs from the web. This is driven by more complex needs of mobile navigation.

Example scenario

Consider a stack navigator with screens A and B. After navigating to A, its `componentDidMount` is called. When pushing B, its `componentDidMount` is also called, but A remains mounted on the stack and its `componentWillUnmount` is therefore not called.

When going back from B to A, `componentWillUnmount` of B is called, but `componentDidMount` of A is not because A remained mounted the whole time.

Similar results can be observed (in combination) with other navigators as well. Consider a tab navigator with two tabs, where each tab is a stack navigator:

```
function App() {
  return (
    <NavigationContainer>
      <Tab.Navigator>
        <Tab.Screen name="First">
          {() => (
            <SettingsStack.Navigator>
              <SettingsStack.Screen
```

```
                name="Settings"
                component={SettingsScreen}
              />
              <SettingsStack.Screen name="Profile" component={ProfileScreen} />
            </SettingsStack.Navigator>
          )}
        </Tab.Screen>
        <Tab.Screen name="Second">
          {() => (
            <HomeStack.Navigator>
              <HomeStack.Screen name="Home" component={HomeScreen} />
              <HomeStack.Screen name="Details" component={DetailsScreen} />
            </HomeStack.Navigator>
          )}
        </Tab.Screen>
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Try this example on Snack

We start on the `HomeScreen` and navigate to `DetailsScreen`. Then we use the tab bar to switch to the `SettingsScreen` and navigate to `ProfileScreen`. After this sequence of operations is done, all 4 of the screens are mounted! If you use the tab bar to switch back to the `HomeStack`, you'll notice you'll be presented with the `DetailsScreen` - the navigation state of the `HomeStack` has been preserved!

React Navigation lifecycle events

Now that we understand how React lifecycle methods work in React Navigation, let's answer the question we asked at the beginning: "How do we find out that a user is leaving (blur) it or coming back to it (focus)?"

React Navigation emits events to screen components that subscribe to them. We can listen to `focus` and `blur` events to know when a screen comes into focus or goes out of focus respectively.

Example:

```
function Profile({ navigation }) {
  React.useEffect(() => {
    const unsubscribe = navigation.addListener('focus', () => {
      // Screen was focused
      // Do something
    });

    return unsubscribe;
  }, [navigation]);

  return <ProfileContent />;
}
```

Try this example on Snack

See Navigation events for more details on the available events and the API usage.

Instead of adding event listeners manually, we can use the `useFocusEffect` hook to perform side effects. It's like React's `useEffect` hook, but it ties into the navigation lifecycle.

Example:

```
import { useFocusEffect } from '@react-navigation/native';

function Profile() {
  useFocusEffect(
    React.useCallback(() => {
      // Do something when the screen is focused

      return () => {
        // Do something when the screen is unfocused
        // Useful for cleanup functions
      };
    }, [])
  );

  return <ProfileContent />;
}
```

Try this example on Snack

If you want to render different things based on if the screen is focused or not, you can use the `useIsFocused` hook which returns a boolean indicating whether the screen is focused.

Summary

- While React's lifecycle methods are still valid, React Navigation adds more events that you can subscribe to through the `navigation` prop.
- You may also use the `useFocusEffect` or `useIsFocused` hooks.

Edit this page