060620 Reducers

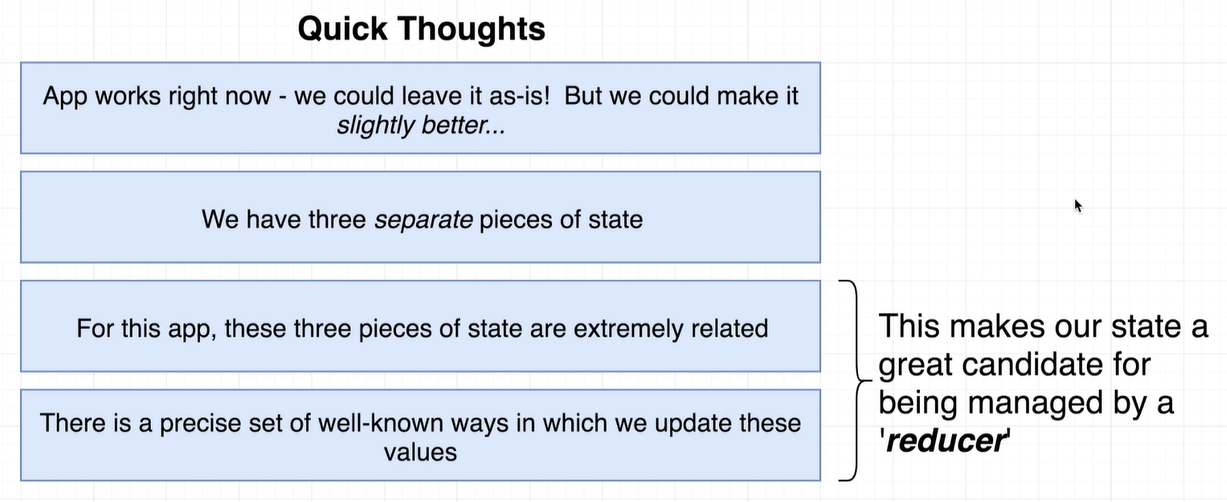
Will either use useState or useReducer to manage state in apps

Will never have examples of the identical variable being managed by both state and reducer

A different way to use state

Color mixing app

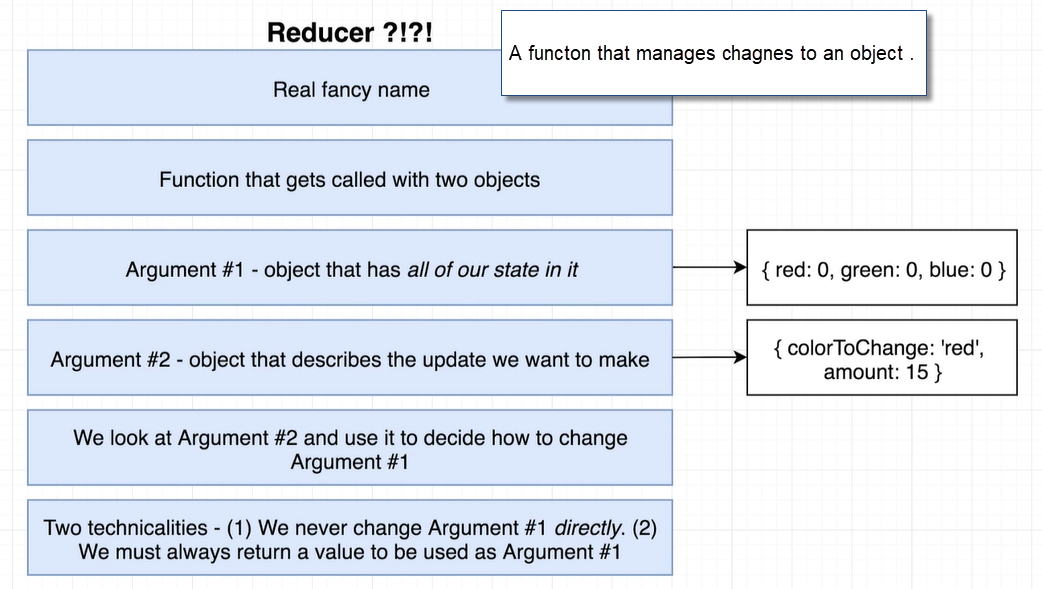
<https://www.udemy.com/course/the-complete-react-native-and-redux-course/learn/lecture/15706702#overview>



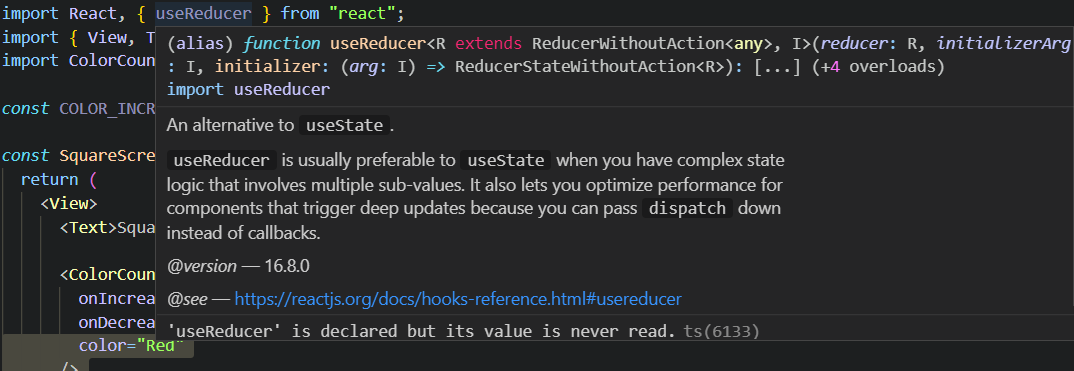
2 Qualities that make for use of reducer

1. Multiple pieces of state very closely related – colors red, green, blue
2. Precise set of well known ways in which values are being updated. – increase or decrease

**A reducer is a function that manages changes to an object**



Import useReducer



Creating the reducer

// https://www.udemy.com/course/the-complete-react-native-and-redux-course/learn/lecture/15706710#overview

// a hook adds functionality to a functional compoent

import React, { useReducer } from "react";

import { View, Text, StyleSheet } from "react-native";

import ColorCounter from "../components/ColorCounter";

*const* COLOR\_INCREMENT = 35;

// define the reducer function

//technically could define this function inside of square screen but by convention

// it is done outside to aleiviate confusion about state

// state -The first argument is state which in this case is  {red:0,green:0,blue:0}

// action -the second argument is an object that  describes how we want to change the state object

// by conention called action

*const* reducer =(*state*,*action*   )*=>*{

// inspect thte action object and make some change to state object

// usually done with switch  statement

//state === {red:number,green:number,blue:number}

// action === {colorToChange: "red"||"green"||"blue",amount:15||-15}

switch (action.colorToChange) {

  case "red":

      // inspect action object and decide how much to change the color by

      // never change state object directly ( likein useState)

      // rebuild into a new object with state updated how we want it to be

      // take  a copy of all of state object and overwrite the existing red property

      // not changing state object, just the copy of it - not changing state directly

      return{...state, red:state.red + action.amount}

  case "green":

    return{...state, green:state.green + action.amount}

  case "blue":

  return{...state, blue:state.blue+ action.amount}

  default:

    // for default , ie no case true just return the state object

  // the reducer must always return something

    return state;

  // reducer is a lot like setter is useState

}

}

*const* SquareScreen = () *=>* {

  // syntax for useReducer

// pass in reducer function and the initial state object as arguments to useReducer

// initially state variable will be the state defined in the second argument object

*const* [state, dispatch]=useReducer(reducer,{red:0,green:0,blue:0})

The reducer with comments but no validation for negative color numers yet

// https://www.udemy.com/course/the-complete-react-native-and-redux-course/learn/lecture/15706718#overview

// a hook adds functionality to a functional compoent

import React, { useReducer } from "react";

import { View, Text, StyleSheet } from "react-native";

import ColorCounter from "../components/ColorCounter";

*const* COLOR\_INCREMENT = 35;

// define the reducer function

//technically could define this function inside of square screen but by convention

// it is done outside to aleiviate confusion about state

// state -The first argument is state which in this case is  {red:0,green:0,blue:0}

// action -the second argument is an object that  describes how we want to change the state object

// by conention called action

*const* reducer = (*state*, *action*) *=>* {

  // inspect thte action object and make some change to state object

  // usually done with switch  statement

  //state === {red:number,green:number,blue:number}

  // action === {colorToChange: "red"||"green"||"blue",amount:15||-15}

  switch (action.colorToChange) {

    case "red":

      // inspect action object and decide how much to change the color by

      // never change state object directly ( likein useState)

      // rebuild into a new object with state updated how we want it to be

      // take  a copy of all of state object and overwrite the existing red property

      // not changing state object, just the copy of it - not changing state directly

      return { ...state, red: state.red + action.amount };

    case "green":

      return { ...state, green: state.green + action.amount };

    case "blue":

      return { ...state, blue: state.blue + action.amount };

    default:

      // for default , ie no case true just return the state object

      // the reducer must always return something

      return state;

    // reducer is a lot like setter is useState

  }

};

*const* SquareScreen = () *=>* {

  // syntax for useReducer

  // pass in reducer function and the initial state object as arguments to useReducer

  // initially state variable will be the state defined in the second argument object

  //in useReducer pass in the reducer funcition, and the initial state of state oject

*const* [state, dispatch] = useReducer(reducer, { red: 0, green: 0, blue: 0 });

  // de structuring

*const* { red, green, blue } = state;

  // when use reducer gets called we get back our current state, very similar to how useState

  //hook returns via the setter

  // anytime state changes due to the reducer the entier component is going to re render

  // the dispatch function runs the reducer to make the change , a better name

  // for dispatch would be runMyReducer

  // every time we need to change state object need to invoke dispatch  (rnuMyReducer)

  // dispatch can by named anything but by convention is dispatch

  // and pass in an argument to be used as our "action object" const reducer =(state,action   )=>{

  // what gets passed into dispatch will be provided as the second arguent of the reducer

  return (

    <View>

      <Text>Square screen </Text>

      <ColorCounter

        // dispatch with the appropriate action object

        onIncrease={() *=>*

          dispatch({ colorToChange: "red", amount: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ colorToChange: "red", amount: -1 \* COLOR\_INCREMENT })

        }

        color="Red"

      />

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ colorToChange: "blue", amount: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ colorToChange: "blue", amount: -1 \* COLOR\_INCREMENT })

        }

        color="Blue"

      />

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ colorToChange: "green", amount: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ colorToChange: "green", amount: -1 \* COLOR\_INCREMENT })

        }

        color="Green"

      />

      <View

        style={{

          height: 150,

          width: 150,

          backgroundColor: `rgb(${red}, ${green}, ${blue})`,

        }}

      />

    </View>

  );

};

*const* styles = StyleSheet.create({});

export default SquareScreen;

// this component will display one view box, that will show the user adjusted color

Adding validation to prevent negative or too high of a number

//https://www.udemy.com/course/the-complete-react-native-and-redux-course/learn/lecture/15706724#questions

import React, { useReducer } from "react";

import { View, Text, StyleSheet } from "react-native";

import ColorCounter from "../components/ColorCounter";

*const* COLOR\_INCREMENT = 35;

*const* reducer = (*state*, *action*) *=>* {

  switch (action.colorToChange) {

    case "red":

      // adding validation or stop if going over 255 or below 0

      // if (state.red + action.amount > 255 || state.red + action.amount < 0) {

      //   return state;

      // }

      // using ternary expression

      return state.red + action.amount > 255 || state.red + action.amount < 0

        ? state

        : { ...state, red: state.red + action.amount };

    case "green":

      return state.green + action.amount > 255 ||

        state.green + action.amount < 0

        ? state

        : { ...state, green: state.green + action.amount };

    case "blue":

      return state.blue + action.amount > 255 || state.blue + action.amount < 0

        ? state

        : { ...state, blue: state.blue + action.amount };

    default:

      return state;

  }

};

*const* SquareScreen = () *=>* {

*const* [state, dispatch] = useReducer(reducer, { red: 0, green: 0, blue: 0 });

*const* { red, green, blue } = state;

  return (

    <View>

      <Text>Square screen </Text>

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ colorToChange: "red", amount: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ colorToChange: "red", amount: -1 \* COLOR\_INCREMENT })

        }

        color="Red"

      />

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ colorToChange: "blue", amount: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ colorToChange: "blue", amount: -1 \* COLOR\_INCREMENT })

        }

        color="Blue"

      />

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ colorToChange: "green", amount: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ colorToChange: "green", amount: -1 \* COLOR\_INCREMENT })

        }

        color="Green"

      />

      <View

        style={{

          height: 150,

          width: 150,

          backgroundColor: `rgb(${red}, ${green}, ${blue})`,

        }}

      />

    </View>

  );

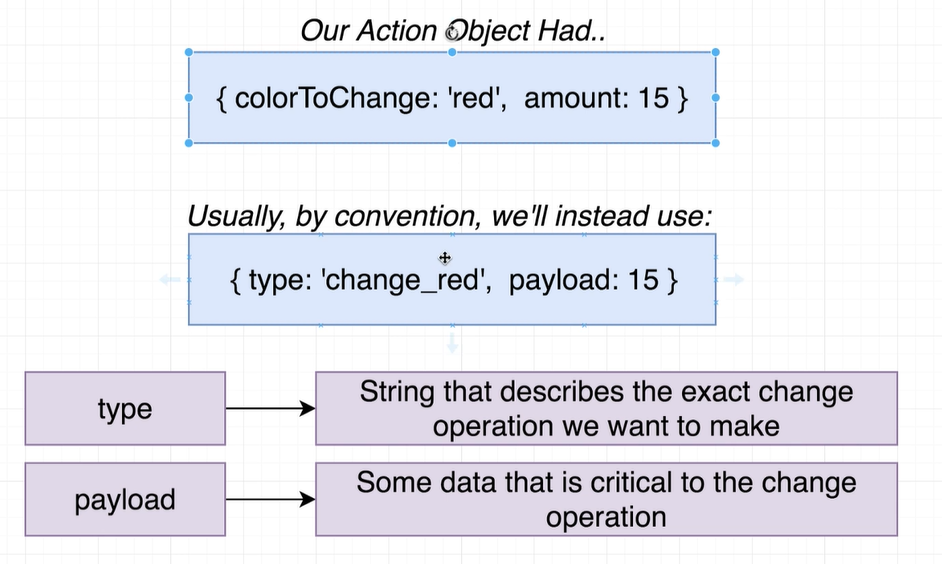
};

*const* styles = StyleSheet.create({});

export default SquareScreen;

// this component will display one view box, that will show the user adjusted color

Reducer conventions



Code changed to use community conventions

//https://www.udemy.com/course/the-complete-react-native-and-redux-course/learn/lecture/15706732#questions

// changing code to use community conventions

import React, { useReducer } from "react";

import { View, Text, StyleSheet } from "react-native";

import ColorCounter from "../components/ColorCounter";

*const* COLOR\_INCREMENT = 35;

// community conventions related to the action object in the reducer

// they are just conventions so can be broken but others may have difficluty understanding code

/////// what we did

// state ==={red:number,green:number,blue:number}

// action === {colorToChange:"red"||"green"||"blue", amount:15 || -15}

////// changing to use community convention

//action ===  { type: "change\_red" || "change\_green" || "change\_blue", payload: 15 || -15 };

//

*const* reducer = (*state*, *action*) *=>* {

  switch (action.type) {

    case "change\_red":

      return state.red + action.payload > 255 || state.red + action.payload < 0

        ? state

        : { ...state, red: state.red + action.payload };

    case "change\_green":

      return state.green + action.payload > 255 ||

        state.green + action.payload < 0

        ? state

        : { ...state, green: state.green + action.payload };

    case "change\_blue":

      return state.blue + action.payload > 255 ||

        state.blue + action.payload < 0

        ? state

        : { ...state, blue: state.blue + action.payload };

    default:

      return state;

  }

};

*const* SquareScreen = () *=>* {

*const* [state, dispatch] = useReducer(reducer, { red: 0, green: 0, blue: 0 });

*const* { red, green, blue } = state;

  return (

    <View>

      <Text>Square screen </Text>

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ type: "change\_red", payload: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ type: "change\_red", payload: -1 \* COLOR\_INCREMENT })

        }

        color="Red"

      />

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ type: "change\_blue", payload: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ type: "change\_blue", payload: -1 \* COLOR\_INCREMENT })

        }

        color="Blue"

      />

      <ColorCounter

        onIncrease={() *=>*

          dispatch({ type: "change\_green", payload: COLOR\_INCREMENT })

        }

        onDecrease={() *=>*

          dispatch({ type: "change\_green", payload: -1 \* COLOR\_INCREMENT })

        }

        color="Green"

      />

      <View

        style={{

          height: 150,

          width: 150,

          backgroundColor: `rgb(${red}, ${green}, ${blue})`,

        }}

      />

    </View>

  );

};

*const* styles = StyleSheet.create({});

export default SquareScreen;

// this component will display one view box, that will show the user adjusted color

SquareScreen.js before reducer

import React, { useState } from "react";

import { View, Text, StyleSheet } from "react-native";

import ColorCounter from "../components/ColorCounter";

*const* COLOR\_INCREMENT = 35;

*const* SquareScreen = () *=>* {

*const* [red, setRed] = useState(0);

*const* [green, setGreen] = useState(0);

*const* [blue, setBlue] = useState(0);

  // helper function to stop increase beyond 256 or decrease below 0

*const* setColor = (*color*, *change*) *=>* {

    switch (color) {

      case "red":

        red + change > 255 || red + change < 0 ? null : setRed(red + change);

        return;

      case "green":

        green + change > 255 || green + change < 0

          ? null

          : setGreen(green + change);

        return;

      case "blue":

        blue + change > 255 || blue + change < 0

          ? null

          : setBlue(blue + change);

        return;

      default:

        return;

    }

  };

  return (

    <View>

      <Text>Square screen </Text>

      <ColorCounter

        onIncrease={() *=>* setColor("red", COLOR\_INCREMENT)}

        onDecrease={() *=>* setColor("red", -1 \* COLOR\_INCREMENT)}

        color="Red"

      />

      <ColorCounter

        onIncrease={() *=>* setColor("blue", COLOR\_INCREMENT)}

        onDecrease={() *=>* setColor("blue", -1 \* COLOR\_INCREMENT)}

        color="Blue"

      />

      <ColorCounter

        onIncrease={() *=>* setColor("green", COLOR\_INCREMENT)}

        onDecrease={() *=>* setColor("green", -1 \* COLOR\_INCREMENT)}

        color="Green"

      />

      <View

        style={{

          height: 150,

          width: 150,

          backgroundColor: `rgb(${red}, ${green}, ${blue})`,

        }}

      />

    </View>

  );

};

*const* styles = StyleSheet.create({});

export default SquareScreen;

// this component will display one view box, that will show the user adjusted color