|  |  |
| --- | --- |
| What is a predictable state container for JS apps? | Redux |
| What helps you write apps that behave consistently, run in different environments, and are easy to test? | Redux |
| What is useable with any view library? | Redux |
| What is a state management framework? | Redux |
| What has a single state object? | Redux |
| What is responsible for the entire state of your application? | Single state object in Reux |
| Where is the state for a React app with 10 components, each with its own local state, stored? | Single state object housed in the Redux *store* |
| What is the single source of truth when it comes to application state? | The Redux store |
| Any time part of your app wants to update state, it must do so through what? | The Redux store |
|  |  |
| What is an object which holds and manages application state? | The Redux store |
| What is a method on the Redux object which is used to create the Redux store? | createStore() |
| What method takes a reducer function as a required object? | createStore() e.g.  var store = Redux.createStore(reducer); |
| What function simply takes state as an argument and returns state? | Reducer function |
| What object provides several methods that allow you to interact with it? | The Redux store |
| What method can retrieve the current state held in the Redux store object? | getState() e.g.  var currentState = store.getState(); |
| What is one of Redux’s core tasks as a state management framework? | Updating state |
| What are triggered by dispatching actions? | All state updates |
| What is simply a JS object that contains info about an action event that has occurred? | An action |
| What receives action objects, then updates its state accordingly? | Redux store |
| What sometimes also carries data? | An action |
| What must carry a type property that specifies the type of action that occurred? | Actions |
| What is as simple as declaring an object with a type property? | Writing a redux action e.g.  var action = {  type:'LOGIN'  }; |
| After creating an action, the next step is sending it where? | To the Redux store |
| What are defined to send the action to the Redux store? | Action creators |
| What is simply a JS function that returns an action? | An action creator e.g.  var actionCreator = () => {  return action;  } |
| What is used to dispatch actions to the Redux store? | The dispatch method |
| What sends an action back to the store? | Calling store.dispatch() and passing the value returned from the action creator |
| From the previous exercise, what is equivalent to:  store.dispatch(actionCreator()); | store.dispatch({ type: 'LOGIN' }); |
| What’s this an example of:  const store = Redux.createStore(  (state = {login: false}) => state  );  const loginAction = () => {  return {  type: 'LOGIN'  }  };  // Dispatch the action here:  store.dispatch(loginAction());  //can also do:  //store.dispatch({ type: 'LOGIN' }); | Initializing state, creating an action, dispatching an action to the store |
| What’s job is to know how to respond to a created and dispatched action? | Reducer function |
| What are responsible for the state modifications that take place in response to actions? | Reducer functions |
| What takes state and action as arguments and always returns a new state? | Reducer function |
| What has no other side effects / other responsibilities besides taking state and action and returning a new state? | Reducer function |
| What is read-only? | state |
| What must always return a new copy of state and never modify state directly? | Reducer function |
| What does not reinforce state immutability, so it’s the responsibility of the dev? | Redux |
| e.g. | const defaultState = {  login: false  };  const reducer = (state = defaultState, action) => {  // change code below this line  if (action.type == 'LOGIN') {  //return state = {'login':true};  } else {  return defaultState;  }  // change code above this line  };  const store = Redux.createStore(reducer);  const loginAction = () => {  return {  type: 'LOGIN'  }  }; |
| What can handle multiple action types? | Redux store |
| *Say you are managing user authentication in your Redux store. You want to have a state representation for when users are logged in and when they are logged out. You represent this with a single state object with the property authenticated. You also need action creators that create actions corresponding to user login and user logout, along with the action objects themselves.* |  |
| What’s common for handling multiple action types in Redux? | Switch statements |
| What returns the current state ? | Default switch statement |
| Example? | const defaultState = {  authenticated: false  };  const authReducer = (state = defaultState, action) => {  // change code below this line  switch (action.type) {  case 'LOGIN':  return {authenticated:true};  break;  case 'LOGOUT':  return {authenticated:false};  break;  default:  return state;  break;  }  // change code above this line  };  const store = Redux.createStore(authReducer);  const loginUser = () => {  return {  type: 'LOGIN'  }  };  const logoutUser = () => {  return {  type: 'LOGOUT'  }  }; |
| A common practice is to assign what as read-only constants, and then reference those constants wherever they are used? | Action types |
| It is common practice to write what in all uppercase? | Constant names |
| What allows you to subscribe listener functions to the store? | store.subscribe() |
| What are called whenever an action is dispatched against the store? | Listener functions |
| What’s simple use could be to subscribe a function to your store that simply logs a message every time an action is received and the store is updated? | Listener functions / store.subscribe() |
| Using callback function and passing to store.subscribe() | // change code below this line  //callback function which inc count every time  //the store receives an action  var updateCount = () => {  count++;  }  //pass updateCount() to store.subscribe()  //will be called whenever an action is dispatched  store.subscribe(updateCount);  // change code above this line |
| What is provided as a solution for complex state model? | Reducer composition |
| You define multiple what to handle different pieces of your apps state? | reducers |
| Then compose those reducers together into one what? | Root reducer |
| What is then passed into the Redux createStore() method? | The root reducer |
| What allows us to combine multiple reducers together? | combineReducers() method |
| What accepts an object as an argument in which you define properties which associate keys to specific reducer functions? | combineReducers() |
| The name you give what will be used by Redux as the name for the associated piece of state? | Name you give to the keys |
| It is good practice to create what for each piece of app state when they are distinct or unique in some way? | A reducer |
| In note-taking app, what could handle authentication while another handles text being submitted? | One reducer for auth, one reducer for text submissions |
| What would the combineReducers() potentially look like? | const rootReducer = Redux.combineReducers({  auth: authenticationReducer,  notes: notesReducer  }); |
| Now, in the above, what will handle all the state associated with our notes and handled by our notesReducer? | The key “notes” |
| What would then be a single object containing auth and notes properties? | The state held in the Redux store |
| const INCREMENT = 'INCREMENT';  const DECREMENT = 'DECREMENT';  const counterReducer = (state = 0, action) => {  switch(action.type) {  case INCREMENT:  return state + 1;  case DECREMENT:  return state - 1;  default:  return state;  }  };  const LOGIN = 'LOGIN';  const LOGOUT = 'LOGOUT';  const authReducer = (state = {authenticated: false}, action) => {  switch(action.type) {  case LOGIN:  return {  authenticated: true  }  case LOGOUT:  return {  authenticated: false  }  default:  return state;  }  };  //combine counterReducer and authReducer using combineReducers()  //const rootReducer  const store = Redux.createStore(rootReducer); | const INCREMENT = 'INCREMENT';  const DECREMENT = 'DECREMENT';  const counterReducer = (state = 0, action) => {  switch(action.type) {  case INCREMENT:  return state + 1;  case DECREMENT:  return state - 1;  default:  return state;  }  };  const LOGIN = 'LOGIN';  const LOGOUT = 'LOGOUT';  const authReducer = (state = {authenticated: false}, action) => {  switch(action.type) {  case LOGIN:  return {  authenticated: true  }  case LOGOUT:  return {  authenticated: false  }  default:  return state;  }  };  //combine counterReducer and authReducer using combineReducers()  const rootReducer = Redux.combineReducers({  count: counterReducer,  auth: authReducer  });  const store = Redux.createStore(rootReducer); |
| What can be used to send more than just type, e.g. other specific data? | actions |
| What usually generate from some other user interaction and tend to carry some data with them? | actions |
| What often needs to know about this other data? | The Redux store |
|  | const ADD\_NOTE = 'ADD\_NOTE';  const notesReducer = (state = 'Initial State', action) => {  switch(action.type) {  // change code below this line  case "ADD\_NOTE":  return state=action.text;  break;  // change code above this line  default:  return state;  }  };  const addNoteText = (note) => {  // change code below this line  return {  type: 'ADD\_NOTE',  text: note  };  // change code above this line  };  const store = Redux.createStore(notesReducer);  console.log(store.getState());  store.dispatch(addNoteText('Hello!'));  console.log(store.getState()); |
|  | //Initial State  //Hello! |
| What is a middleware provided by Redux for handling inevitable asynchronous actions? | Redux Thunk |
| What do you pass as an argument to Redux.applyMiddleware()? | Redux Thunk |
| What is then provided as a second optional parameter to the createStore()? | This statement…? E.g.  const store = Redux.createStore(  asyncDataReducer,  Redux.applyMiddleware(ReduxThunk.default)  ); |
| In order to create what, you return a function in the action creator that takes dispatch as an argument? | Asynchronous action |
| It’s common to dispatch an action before initiating any what so that your app state knows that some data is being requested? | Asynchronous behavior, e.g. so your app can display a loading icon |
| Once you receive the data, you then dispatch another what which carries the data as a payload along with info that the action is completed? | action |
| What are you passing as a parameter to this special action creator? | dispatch |
| What is an example of dispatching an action to the store and then a dispatching another action to the store once data is received i.e. an asynchronous action? | //uses setTimeout to simulate an asynchronous action  const REQUESTING\_DATA = 'REQUESTING\_DATA'  const RECEIVED\_DATA = 'RECEIVED\_DATA'  const requestingData = () => { return {type: REQUESTING\_DATA} }  const receivedData = (data) => { return {type: RECEIVED\_DATA, users: data.users} }  const handleAsync = () => {  return function(dispatch) {  // dispatch request action here  store.dispatch(requestingData())  setTimeout(function() {  let data = {  users: ['Jeff', 'William', 'Alice']  }  // dispatch received data action here  store.dispatch(receivedData(data))  }, 2500);  }  };  const defaultState = {  fetching: false,  users: []  };  const asyncDataReducer = (state = defaultState, action) => {  switch(action.type) {  case REQUESTING\_DATA:  return {  fetching: true,  users: []  }  case RECEIVED\_DATA:  return {  fetching: false,  users: action.users  }  default:  return state;  }  };  const store = Redux.createStore(  asyncDataReducer,  Redux.applyMiddleware(ReduxThunk.default)  ); |
| Use the names that are provided and define incAction and decAction action creators, the counterReducer(), INCREMENT and DECREMENT action types, and finally the Redux store. Once you're finished you should be able to dispatch INCREMENT or DECREMENT actions to increment or decrement the state held in the store. | // define a constant for increment action types  const INCREMENT = 'INCREMENT';  // define a constant for decrement action types  const DECREMENT = 'DECREMENT';  // define the counter reducer which will increment or decrement the state based on the action it receives  const counterReducer = (state=0,action) => {  switch (action.type) {  case 'INCREMENT':  return state = state+1;  break;  case 'DECREMENT':  return state = state-1;  break;  default:  return state;  break;  }  };  // define an action creator for incrementing  const incAction = () => {  return {  type:INCREMENT  };  }  // define an action creator for decrementing  const decAction = () => {  return {  type:DECREMENT  };  }  // define the Redux store here, passing in your reducers  const store = Redux.createStore(counterReducer); |
| What means you never modify a state directly, but instead retunr a new copy of state? | State immutability |
| What are primiative values and are immutable by nature? | Strings and numbers |
| What are mutable? | Arrays and objects |
| What make good data structures because they are mutable and good for representing many type of info? | Array and objects |
|  | const ADD\_TO\_DO = 'ADD\_TO\_DO';  // A list of strings representing tasks to do:  const todos = [  'Go to the store',  'Clean the house',  'Cook dinner',  'Learn to code',  ];  const immutableReducer = (state = todos, action) => {  switch(action.type) {  case ADD\_TO\_DO:  // don't mutate state here or the tests will fail  //The concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.  return state.concat(action.todo);  default:  return state;  }  };  // an example todo argument would be 'Learn React',  const addToDo = (todo) => {  return {  type: ADD\_TO\_DO,  todo  }  }  const store = Redux.createStore(immutableReducer); |
| What in ES6 helps enforce state immutability? | Spread operator ... |
| What helps with producing a new array from an existing array? | Spread operator ... |
| Let newArray = [...myArray]; | Using spread operator |
| Ho clone an array but add additional values in the new array? | you could write [...myArray, 'new value']. This would return a new array composed of the values in myArray and the string 'new value' as the last value. |
|  | const ADD\_TO\_DO = 'ADD\_TO\_DO';  // A list of strings representing tasks to do:  const todos = [  'Go to the store',  'Clean the house',  'Cook dinner',  'Learn to code',  ];  const immutableReducer = (state = todos, action) => {  switch(action.type) {  case ADD\_TO\_DO:  return [...state, action.todo];  default:  return state;  }  };  // an example todo argument would be 'Learn React',  const addToDo = (todo) => {  return {  type: ADD\_TO\_DO,  todo  }  }  const store = Redux.createStore(immutableReducer); |
| slice(), concat(), and spread operator are useful for what? | Removing items from arrays while maintaining state immutability |
|  | const immutableReducer = (state = [0,1,2,3,4,5], action) => {  switch(action.type) {  case 'REMOVE\_ITEM':  return state.slice(0, action.index).concat(state.slice(action.index+1));  default:  return state;  }  };  const removeItem = (index) => {  return {  type: 'REMOVE\_ITEM',  index  }  }  const store = Redux.createStore(immutableReducer); |
| What operation helps with avoiding mutating object states? | Object.assign() |
| How to create newObject as a new object, which contains the properties that currently exist in obj1 and obj2? | const newObject = Object.assign({}, obj1, obj2); |
| Make sure first object is empty object;  Make sure state is second object; | const defaultState = {  user: 'CamperBot',  status: 'offline',  friends: '732,982',  community: 'freeCodeCamp'  };  const immutableReducer = (state = defaultState, action) => {  switch(action.type) {  case 'ONLINE':  // don't mutate state here or the tests will fail  var newObject = {status:'online'};  return Object.assign({}, state, newObject);  default:  return state;  }  };  const wakeUp = () => {  return {  type: 'ONLINE'  }  };  const store = Redux.createStore(immutableReducer); |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |