Redux interview questions

Interview questions for your next interview

E-mail a friend | Download text file

Question

“ What is the concept of “single source of truth” in Redux? ”

Answer

The state of your whole application is stored in an object tree within a single store. This makes it easy to create universal apps, as the state from the server can be serialized and hybridized into the client with no extra coding effort. A single state tree also makes it easier to debug or introspect an application; it also enables persisting the app's state in development, for a faster development cycle.

Send a correction

Question

“ How is it different from MVC and Flux? ”

Answer

In classical widely known MVC architecture, there is a clear separation between data (model), presentation (view) and logic (controller). There is one issue with this, especially in large-scale applications: The flow of data is bidirectional. This means that one change (a user input or API response) can affect the state of an application in many places in the code — for example, two-way data binding. That can be hard to maintain and debug.

Flux is very similar to Redux. The main difference is that Flux has multiple stores that change the state of the application, and it broadcasts these changes as events. Components can subscribe to these events to sync with the current state. Redux doesn’t have a dispatcher, which in Flux is used to broadcast payloads to registered callbacks. Another difference in Flux is that many varieties are available, and that creates some confusion and inconsistency.

Send a correction

Question

“ What are “actions” in Redux? ”

Answer

In a nutshell, actions are events. Actions send data from the application (user interactions, internal events such as API calls, and form submissions) to the store. The store gets information only from actions. Internal actions are simple JavaScript objects that have a type property (usually constant), describing the type of action and payload of information being sent to the store.

```
{
    type: LOGIN_FORM_SUBMIT,
    payload: {username: 'alex', password: '123456'}
}
```

Actions are created with action creators. That sounds obvious, I know. They are just functions that return actions.

```
function authUser(form) {
    return {
        type: LOGIN_FORM_SUBMIT,
        payload: form
    }
}
```

Calling actions anywhere in the app, then, is very easy. Use the dispatch method, like so:

```
dispatch(authUser(form));
```

Question

“ What is the role of reducers in Redux? ”

Answer

In Redux, reducers are functions (pure) that take the current state of the application and an action and then return a new state. Understanding how reducers work is important because they perform most of the work. Here is a very simple reducer that takes the current state and an action as arguments and then returns the next state:

```
function handleAuth(state, action) {
    return _.assign({}, state, {
        auth: action.payload
    });
}
```

For more complex apps, using the combineReducers() utility provided by Redux is possible (indeed, recommended). It combines all of the reducers in the app into a single index reducer. Every reducer is responsible for its own part of the app's state, and the state parameter is different for every reducer. The combineReducers() utility makes the file structure much easier to maintain.

If an object (state) changes only some values, Redux creates a new object, the values that didn’t change will refer to the old object and only new values will be created. That's great for performance. To make it even more efficient you can add Immutable.js.

```
const rootReducer = combineReducers({
    handleAuth: handleAuth,
    editProfile: editProfile,
    changePassword: changePassword
});
```

Send a correction

Question

“ What is ‘Store’ in Redux? ”

Answer

Store is the object that holds the application state and provides a few helper methods to access the state, dispatch actions and register listeners. The entire state is represented by a single store. Any action returns a new state via reducers. That makes Redux very simple and predictable.

```
import { createStore } from ‘redux’;
let store = createStore(rootReducer);
let authInfo = {username: ‘alex’, password: ‘123456’};
store.dispatch(authUser(authInfo));
```

Send a correction

Question

“ How is state changed in Redux? ”

Answer

The only way to change the state is to emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state. Instead, they express an intent to transform the state. Because all changes are centralized and happen one by one in a strict order, there are no subtle race conditions to watch out for. As actions are just plain objects, they can be logged, serialized, stored, and later replayed for debugging or testing purposes.

Send a correction

Previous Next

Ask a question

Ask a question, and one of our engineers will answer it.