Actions · Redux

Actions

First, let's define some actions.

Actions are payloads of information that send data from your application to your store. They are the *only* source of information for the store. You send them to the store using `store.dispatch()`.

Here's an example action which represents adding a new todo item:

```
const ADD_TODO = 'ADD_TODO'
```

```
{
  type: ADD_TODO,
  text: 'Build my first Redux app'
}
```

Actions are plain JavaScript objects. Actions must have a `type` property that indicates the type of action being performed. Types should typically be defined as string constants. Once your app is large enough, you may want to move them into a separate module.

```
import { ADD_TODO, REMOVE_TODO } from '../actionTypes'
```

> **Note on Boilerplate**
>
> You don't have to define action type constants in a separate file, or even to define them at all. For a small project, it might be easier to just use string literals for action types. However, there are some benefits to explicitly declaring constants in larger codebases. Read Reducing Boilerplate for more practical tips on keeping your codebase clean.

Other than `type`, the structure of an action object is really up to you. If you're interested, check out Flux Standard Action for recommendations on how actions could be constructed.

We'll add one more action type to describe a user ticking off a todo as completed. We refer to a particular todo by `index` because we store them in an array. In a real app, it is wiser to generate a unique ID every time something new is created.

```
{
```

```
  type: TOGGLE_TODO,
  index: 5
}
```

It's a good idea to pass as little data in each action as possible. For example, it's better to pass `index` than the whole todo object.

Finally, we'll add one more action type for changing the currently visible todos.

```
{
  type: SET_VISIBILITY_FILTER,
  filter: SHOW_COMPLETED
}
```

Action Creators

Action creators are exactly that—functions that create actions. It's easy to conflate the terms “action” and “action creator,” so do your best to use the proper term.

In Redux action creators simply return an action:

```
function addTodo(text) {
  return {
    type: ADD_TODO,
    text
  }
}
```

This makes them portable and easy to test.

In traditional Flux action creators often trigger a dispatch when invoked, like so:

```
function addTodoWithDispatch(text) {
  const action = {
    type: ADD_TODO,
    text
  }
  dispatch(action)
}
```

In Redux this is *not* the case.

Instead, to actually initiate a dispatch, pass the result to the `dispatch()` function:

```
dispatch(addTodo(text))
dispatch(completeTodo(index))
```

Alternatively, you can create a **bound action creator** that automatically dispatches:

```
const boundAddTodo = (text) => dispatch(addTodo(text))
const boundCompleteTodo = (index) => dispatch(completeTodo(index))
```

Now you'll be able to call them directly:

```
boundAddTodo(text)
boundCompleteTodo(index)
```

The `dispatch()` function can be accessed directly from the store as `store.dispatch()`, but more likely you'll access it using a helper like react-redux's `connect()`. You can use `bindActionCreators()` to automatically bind many action creators to a `dispatch()` function.

Action creators can also be asynchronous and have side-effects. You can read about async actions in the advanced tutorial to learn how to handle AJAX responses and compose action creators into async control flow. Don't skip ahead to async actions until you've completed the basics tutorial, as it covers other important concepts that are prerequisite for the advanced tutorial and async actions.

Source Code

`actions.js`

```
/*
 * action types
 */

export const ADD_TODO = 'ADD_TODO'
export const TOGGLE_TODO = 'TOGGLE_TODO'
export const SET_VISIBILITY_FILTER = 'SET_VISIBILITY_FILTER'

/*
 * other constants
 */
```

```
export const VisibilityFilters = {
  SHOW_ALL: 'SHOW_ALL',
  SHOW_COMPLETED: 'SHOW_COMPLETED',
  SHOW_ACTIVE: 'SHOW_ACTIVE'
}

/*
 * action creators
 */

export function addTodo(text) {
  return { type: ADD_TODO, text }
}

export function toggleTodo(index) {
  return { type: TOGGLE_TODO, index }
}

export function setVisibilityFilter(filter) {
  return { type: SET_VISIBILITY_FILTER, filter }
}
```

Next Steps

Now let's define some reducers to specify how the state updates when you dispatch these actions!