Immutable Updates in Redux

Immutability is a key concept to understand when you're using Redux, but it can be difficult to understand at first. Writing code to do immutable state updates is tricky. Below, you will find a few common tricks. Try them out on your own, whether in the browser developer console or in a real app. Pay particular attention to the nested object updates, and practice those. I find those to be the trickiest.

All of this actually applies to React state too, so the things you learn in this guide will apply whether you use Redux or not.

Finally, I should mention that some of this code can become easier to write by using a library like Immutable.js, though it comes with its own tradeoffs. If you recoil at the syntax below, check out Immutable.js.

The Spread Operator

These examples make heavy use of the **spread** operator for arrays and objects. It's represented by `...` and when placed before an object or array, it unwraps the children within.

```
// For arrays:
let nums = [1, 2, 3];
let newNums = [...nums]; // => [1, 2, 3]
nums === newNums // => false! not the same array

// For objects:
let person = {
  name: "Liz",
  age: 32
}
let newPerson = {...person};
person === newPerson // => false! not the same object

// Internals are left alone:
```

```
let company: {
  name: "Foo Corp",
  people: [
    {name: "Joe"},
    {name: "Alice"}
  ]
}
let newCompany = {...company};
newCompany === company // => false! not the same object
newCompany.people === company.people // => true!
```

When used as shown above, the spread operator makes it easy to create a new object or array that contains the exact same contents as another one. This is useful for creating a copy of an object/array, and then overwriting specific properties that you need to change:

```
let liz = {
  name: "Liz",
  age: 32,
  location: {
    city: "Portland",
    state: "Oregon"
  },
  pets: [
    {type: "cat", name: "Redux"}
  ]
}

// Make Liz one year older, while leaving everything
// else the same:
let olderLiz = {
  ...liz,
  age: 33
}
```

The spread operator for objects is a stage 3 draft, which means it’s not officially part of JS yet. You’ll need to use a transpiler like Babel to use it in your code. If you use Create React App, you can already use it.

Recipes for Updating State

These examples are written in the context of returning state from a Redux reducer: pretend that the `state = {whatever}` at the top is the state that was passed in to the reducer, and then the updated version is returned underneath.

Applying to React and setState

To apply these examples to plain React state, you just need to tweak a couple things:

```
return {
  ...state,
  updates here
}

// becomes:
this.setState({
  ...this.state,
  updates here
})
```

Here are some common immutable update operations:

Updating an Object

```
state = {
  clicks: 0,
  count: 0
}
```

```
return {
  ...state,
  clicks: state.clicks + 1,
  count: state.count - 1
}
```

Updating a Nested Object

```
state = {
  house: {
    name: "Ravenclaw",
    points: 17
  }
}

// Two points for Ravenclaw
return {
  ...state,
  house: {
    ...state.house,
    score: state.house.points + 2
  }
}
```

Updating an Object by Key

```
state = {
  houses: {
    gryffindor: {
      score: 15
    },
    ravenclaw: {
      score: 18
    },
    hufflepuff: {
      score: 7
    },
    slytherin: {
      score: 5
    }
  }
}

// Add 3 points to Ravenclaw,
// when the name is stored in a variable
key = "ravenclaw";
return {
  ...state,
  houses: {
    ...state.houses,
    [key]: {
      ...state.houses[key],
      score: state.houses[key] + 3
    }
  }
}
```

Add an element to the beginning of an array

```
array = [1, 2, 3];
newItem = 0;
return [
  newItem,
  ...array
];
```

Add an element to the end of an array

```
array = [1, 2, 3];
newItem = 4;
return [
  ...array,
  newItem
];
```

Add an element in the middle of an array

Pro tip: Write unit tests for these things. It’s easy to make off-by-one errors.

```
array = [1, 2, 3, 5, 6];
newItem = 4;
return [                // array is new
  ...array.slice(0, 3), // first X items unchanged
  newItem,
  ...array.slice(3)     // last Y items unchanged
];
```

Change an element in the middle of an array

This is the same pattern as adding an item, except that the indexes are different.

Pro tip: Write unit tests for these things. It’s easy to make off-by-one errors.

```
array = [1, 2, "X", 4];
newItem = 3;
return [                // array is new
  ...array.slice(0, 2), // first X items unchanged
  newItem,
  ...array.slice(3)     // last Y items unchanged
];
```