Chapter 17: Performance

Section 17.1: Performance measurement with ReactJS

You can't improve something you can't measure. To improve the performance of React components, you should be able to measure it. ReactJS provides with *addon* tools to measure performance. Import the `react-addons-perf` module to measure the performance

```
import Perf from 'react-addons-perf' // ES6
var Perf = require('react-addons-perf') // ES5 with npm
var Perf = React.addons.Perf; // ES5 with react-with-addons.js
```

You can use below methods from the imported `Perf` module:

- Perf.printInclusive()
- Perf.printExclusive()
- Perf.printWasted()
- Perf.printOperations()
- Perf.printDOM()

The most important one which you will need most of the time is `Perf.printWasted()` which gives you the tabular representation of your individual component's wasted time

(index)	Owner > component	Wasted time (ms)	Instances
0	"Todos > TodoItem"	102.76999999977124	1000

Total time: 132.71 ms react-with-addons.js:9900

You can note the **Wasted time** column in the table and improve Component's performance using **Tips & Tricks** section above

Refer the React Official Guide and excellent article by Benchling Engg. on React Performance

Section 17.2: React's diff algorithm

Generating the minimum number of operations to transform one tree into another have a complexity in the order of O(n^3) where n is the number of nodes in the tree. React relies on two assumptions to solve this problem in a linear time - O(n)

1. Two components of the same class will generate similar trees and tw components of different classes will generate different trees.

2. It is possible to provide a unique key for elements that is stable across different renders.

In order to decide if two nodes are different, React differentiates 3 cases

1. Two nodes are different, if they have different types.

- for example, `<div>...</div>` is different from `<span>...</span>`

2. Whenever two nodes have different keys

- for example, `<div key="1">...</div>` is different from `<div key="2">...</div>`

Moreover, **what follows is crucial and extremely important to understand** if you want to optimise performance

> If they [two nodes] are not of the same type, React is not going to even try at matching what they render. It is just going to remove the first one from the DOM and insert the second one.

Here's why

> It is very unlikely that a element is going to generate a DOM that is going to look like what a would generate. Instead of spending time trying to match those two structures, React just re-builds the tree from scratch.

Section 17.3: The Basics - HTML DOM vs Virtual DOM

HTML DOM is Expensive

Each web page is represented internally as a tree of objects. This representation is called *Document Object Model*. Moreover, it is a language-neutral interface that allows programming languages (such as JavaScript) to access the HTML elements.

In other words

> The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

However, those **DOM operations** are extremely **expensive**.

Virtual DOM is a Solution

So React's team came up with the idea to abstract the *HTML DOM* and create its own *Virtual DOM* in order to compute the minimum number of operations we need to apply on the *HTML DOM* to replicate current state of our application.

The Virtual DOM saves time from unnecessary DOM modifications.

How Exactly?

At each point of time, React has the application state represented as a `Virtual DOM`. Whenever application state changes, these are the steps that React performs in order to optimise performance

1. Generate a new *Virtual DOM* that represents the new state of our application
2. Compare the old Virtual DOM (which represents the current HTML DOM) vs the new Virtual DOM
3. Based on 2. find the minimum number of operations to transform the old Virtual DOM (which represents the current HTML DOM) into the new Virtual DOM

- to learn more about that - read React's Diff Algorithm

4. After those operations are found, they are mapped into their equivalent *HTML DOM* operations

- remember, the *Virtual DOM* is only an abstraction of the *HTML DOM* and there is a isomorphic relation between them

5. Now the minimum number of operations that have been found and transferred to their equivalent *HTML DOM* operations are now applied directly onto the application's *HTML DOM*, which saves time from modifying the *HTML DOM* unnecessarily.

Note: Operations applied on the Virtual DOM are cheap, because the Virtual DOM is a JavaScript Object.

Section 17.4: Tips & Tricks

When two nodes are not of the same type, React doesn't try to match them - it just removes the first node from the DOM and inserts the second one. This is why the first tip says

1. If you see yourself alternating between two components classes with very similar output, you may want to make it the same class.

2. Use shouldComponentUpdate to prevent component from rerender, if you know it is not going to change, for example

```
shouldComponentUpdate: function(nextProps, nextState) {
  return nextProps.id !== this.props.id;
}
```