introduction to raw React

Steven Brawer
June 10, 2017

introduction	2
headers	2
two React functions	2
first example - Hello, world!	3
second example - nesting elements	4
third example - using functions	5
fourth example - unordered list	7
fifth example - React in classes	8

introduction

Raw React is React without JSX - that is, just using the native React function calls. This document presents an introduction to raw React and gives five examples of increasing complexity. The fifth example shows how to use React in a structured manner in a class hierarchy. This document uses javascript ES6.

headers

To access the React library, insert the following above the *<script>* section.

```
<script type="text/javascript" src = "https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react.js"></script>
<script type="text/javascript" src = "https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react-dom.js"></script>
```

Modify the version if there is a new version. As of this writing, the most recent version is 15.5.4.

two React functions

I present the form of the two React functions that are used in this document. I present them briefly, and then give examples which illustrate their use.The first React function is:

```
let reactElement = React.createElement(tagName, attrsStyles, children);
```

This creates a virtual HTML element **tagName** ("p", "div", etc). The element is stored in React's memory and has not yet been rendered to the DOM. **attrsStyles** is a pure javascript object with attributes and styles. **children** is a child of the element *tagName*. It can be a text string, an element created with *React.createElement*, or an array of such elements.

The other function is

```
ReactDOM.render(reactElement, anElement);
```

where **anElement** is an existing DOM element, such as one obtained with

```
anElement = document.getElementById(....);
```

The render function renders *reactElement* (which, as will be seen, can be quite complex) to the DOM, where it will appear in the browser.

first example - Hello, world!

The following program will print *Hello, world!* in an *<h1>* element in the browser. I leave off the headers to keep things simple.

The html file is (without headers and other obvious elements)

```
<body>
        <div id = "reactAnchor"></div>

        .... headers

        <script>
        const CE = React.createElement;

        let anchor = document.getElementById("reactAnchor");

        let attrsStyles = {style: {
                                color: 'red',
                                border: '1px solid black'
                                }};
        let elem = CE("h1",attrsStyles,"Hello, world!");
        ReactDOM.render(elem, anchor);l

        </script>
</body>
```

The result is (in Safari)

Some comments on this example.

The string "reactAnchor" can be anything - there is nothing magical about "reactAnchor".

We use the abbreviation

```
const CE = React.createElement;
```

for convenience. I think it makes reading the program easier. The *createElement* function in this example creates an **h1** tag, with two styles, and adds the string "Hello, world!" as the child (or *textContent*) of the *h1* tag.

The **render** function renders the element to the DOM as a child of the ** ** whose id is *reactAnchor*.

second example - nesting elements

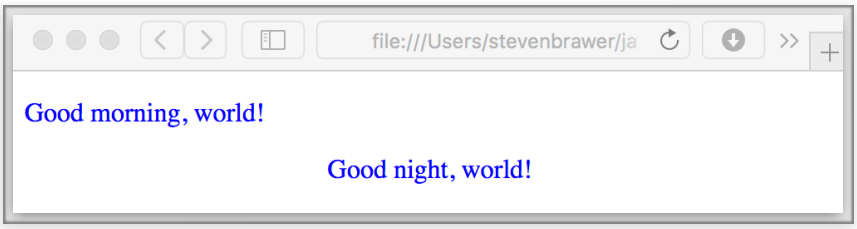
The raw React functions allow for nesting of elements in a straightforward manner. In the following example, I create

```
<div id="reactAnchor">
        <div style = "color: blue">
                <p style= "text-align:left">Good morning, world!</p>
                <p style = "text-align:center">Good night, world!</p>
        </div>
</div>
```

Here is the relevant HTML and javascript

```
<body>
<div id = "reactAnchor"></div>

....headers

<script>

const CE = React.createElement;
let anchor = document.getElementById("reactAnchor");

// work from inside out
        let obj1 = {key: "1",style:{textAlign:"left"}};
        let elem1 = CE("p",obj1,"Good morning, world!");

        let obj2 = {key: "2",style:{textAlign:"center"}};
        let elem2 = CE("p",obj2,"Good night, world!");

let dv = CE("div",{style:{color: "blue"}},[elem1,elem2]);

ReactDOM.render(dv,anchor);
</script>
</body>
```

When executed, the browser looks like



The *key* attribute is discussed below.

Note that *obj1* and *obj2* are different objects. The second argument of *createElement* is a pointer to the object. If we had done, for the second *p* element,

```
obj1 = {key: "2",style:{textAlign:"center"}};
let elem2 = CE("p",obj1,"Good night, world!");
```

then **both** *p* elements would have the same key (not good) and centered text, as both *createElement* functions would be pointing to the same object - namely *obj1*.

In the style object, we have used **textAlign** rather than *text-align*. This is the general rule for all attributes and styles in React functions: **use camelCase**. So, for example, *background-color* becomes **backgroundColor,** *text-indent* becomes **textIndent**, and so forth. One special case: *class* becomes *className.*

In creating the *div* element, note that the third argument of *createElement ("div"...)* is an array

```
[elem1,elem2]
```

This makes both *elem1* and *elem2* children of the *div*, both are on the same level, and they appear in the DOM in the same order as in the array. Whenever an element has multiple children, each child must have a unique **key** attribute. In the above example, we have set the *key* to strings "1" and "2". The keys may be arbitrary strings, so long as they are unique.

The indentation in the example is only to make reading the program easier, and has no interpretation (this is pure javascript, not python).

third example - using functions

So far we have done a lot of work for very little return. However, **raw React allows DOM elements to be constructed with structured code, where the elements may come from different functions or instances of various classes.** Here is the above example, using a different function for each element. Later I will generalize this to using classes.

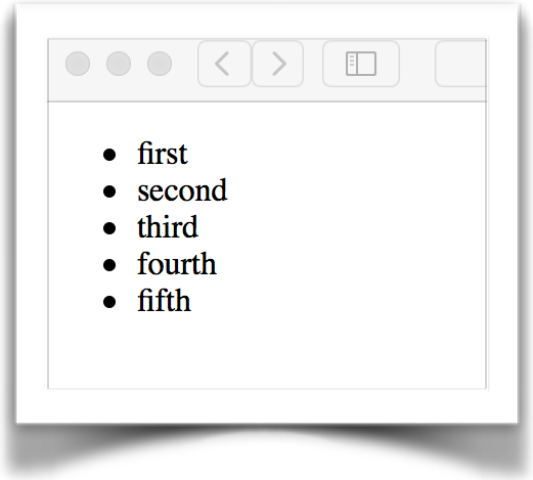
```
<body>
<div id = "reactAnchor"></div>

....headers

<script>

const CE = React.createElement;
let anchor = document.getElementById("reactAnchor");

const newKey = (function(){
                    let key = 0;
                    return function(){ return String(++key);};
                  })();

// create a "p" element
function createP(align, text){
        let obj = {key: newKey(),
                   style: {textAlign: align}};
        return CE("p", obj, text);
};

// create the "div" element
function createDiv(color, children){
        return CE("div", {style:{color: color}}, children);
}

let arr = [
                createP("left",      "Good morning, world!"),
                createP("center", "Good night, world!")
          ];
let dv = createDiv("blue", arr);

ReactDOM.render(dv,anchor);

</script>
</body>
```

In this script, **newKey** is a function which, at each call, returns successively "1", "2", "3",...., unique keys.

In this script, the idividual HTML elements are created in functions and assembled in the main program. One can go even farther, and assemble parts of the nested elements in different functions or class methods. We will see later how this is done for a set of classes in the fifth example.

fourth example - unordered list

This is straightforward. This is what our program will cause to be output to the browser.



Here is the code using raw React. Note that the *<li>* elements are all created in a loop.

```
<body>
<div id = "reactAnchor"></div>

...headers

<script>

const CE = React.createElement;

const newKey = (function(){
                    let key = 0;
                    return function(){ return String(++key);};
                  })();

let anchor = document.getElementById("reactAnchor");

function createLi(text){
        return CE("li",{key: newKey()},text);
};

let listWords= ["first","second","third","fourth","fifth"];
let elems = [];
for(let i=0;i<listWords.length;i++){
        elems.push(createLi(listWords[i]));
}
let ul = CE("ul",{},elems);
```

```
ReactDOM.render(ul,anchor);
</script>
</body>
```

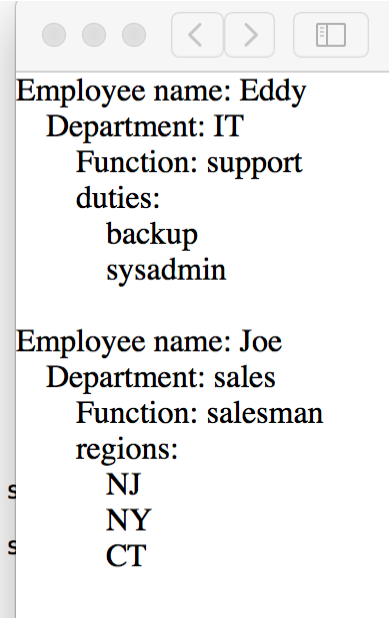
fifth example - React in classes

We ceate a simple class hierarchy, using mixins. The hierarchy is

```
Employee
        Department
                Functions
```

Each of the three categories is represented by a class. The *employee* object has a pointer to a *department* object, which has a pointer to a *functions* object. Each class creates and returns its appropriate HTML element using raw React, much like the third example above, except now with classes.

The program causes the following to be output to the browser.



I'm trying to keep this simple, so minimal styling. Note that each employee has very different functions. This is all handled uniformly by the code below.

```
<style>
* {
        text-align: left;
        margin: 0;
        padding: 0;
}
div{
        display: block;
}
</style>
</head>
<body>
<div id="reactAnchor"></div>

<script type="text/javascript" src = "https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/
react.js"></script>
<script type="text/javascript" src = "https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/
react-dom.js"></script>

<script>

function go(){                  // see window.onload at end of script

const CE = React.createElement;
let employeeList = [];          // array of Employee objects
let anchorElem = document.getElementById("reactAnchor");

const newKey = (function(){
                  let key = 0;
                  return function(){ return String(++key);};
                })();

//==================  CLASSES

class  Employee{
        constructor(name){
                this.name = name;
                this.department;
        };

        // Create and return HTML elements using raw React
        getElement(startIndent, deltaIndent){
                let deptElem =
                   this.department.getElement(startIndent + deltaIndent, deltaIndent);
                let empElem = CE("p",{key:newKey()},"Employee name: " + this.name);
                let obj = {
                        key: newKey(),
                        style: { textIndent: startIndent}  };
```

```
                return CE("div",obj,[empElem, deptElem]);

        };
}; // class Employee

//---------------

class Department {
        constructor(name){
                this.name = name;
                this.functions;
        };

        // Create and return HTML elements using raw React
        getElement(startIndent, deltaIndent){
                let funcElem =
                   this.functions.getElement(startIndent + deltaIndent,deltaIndent);
                let deptElem = CE("p",{key:newKey()}, "Department: " + this.name);
                let obj = {
                        key: newKey(),
                        style: { textIndent: startIndent}  };
                return CE("div",obj,[deptElem, funcElem]);

        };
}; // class Department

//------------------

class  Functions{
        constructor(name, label, tasks){
                this.name = name;
                this.tasks = tasks; // array of strings
                this.label =label;
        };

        // Create and return HTML elements using raw React
        getElement(startIndent, deltaIndent){

                let elem1 = CE("p",{key:newKey()}, "Function: " + this.name);
                let elem2 = CE("p", {key:newKey()}, this.label+":");

                // crete list of regions,duties, whatever
                let listElems = this.createListElements(startIndent + deltaIndent);

                // create div to hold all the elements
                let obj = {
                        key: newKey(),
                        style: { textIndent: startIndent}  };
                return CE("div",obj,[elem1,elem2, listElems]);
```

```
        };

        createListElements(startIndent){
                let arr = [];
                for(let i=0;i<this.tasks.length;i++){
                        // text indent here overrides that of div in getElement()
                        let obj = {key:newKey(),style:{textIndent:startIndent}};
                        arr.push( CE("li",obj,this.tasks[i]) );
                }
                return CE("ul",{key: newKey()},arr);
        };
} // class Functions

//==================  FUNCTIONS

// create class instances for each employee in data
// *** data is at end of script ***
function createEmployeeList(){
        for(let i=0;i<employees.length;i++){
                let emp = employees[i];
                let employeeInst = new Employee(emp.name);

                let dept = emp.department;
                let departmentInst = new Department(dept.name);
                employeeInst.department = departmentInst;

                let funct = dept.functions;
                let duties = funct[funct.label];
                let functionInst = new Functions(funct.name, funct.label, duties);
                departmentInst.functions = functionInst;

                //console.log(employeeInst);

                employeeList.push(employeeInst);
        } // for
};


function render(){
        let elementList = [];
        for(let i=0;i<employeeList.length;i++){
                let employee = employeeList[i];
                let elem = employee.getElement(0,15); // ALL ACTION HERE
                elementList.push(elem);
                elementList.push(CE("br",{key:newKey()}) ); // new line
        }
        // render, creating new <div> to hold all elements
        ReactDOM.render(CE("div",{},elementList), anchorElem);
```

```
}

//================== EXECUTE

createEmployeeList();
render();

} // go

//================= DATA

const employees =
[
        {
        name: "Eddy",
        department: {
                name: "IT",
                functions: {
                        name: "support",
                        label: "duties",
                        duties: ["backup","sysadmin"]
                } // functions
        } // department
        },
        {
        name: "Joe",
        department: {
                name: "sales",
                functions: {
                        name: "salesman",
                        label: "regions",
                        regions: ["NJ","NY","CT"]
                } // functions
        } // department
        }
];

window.onload = go;

</script>
</body>
</head>
```

This creates a set of nested HTML elements, shown below for employee Eddy. The hierarchy is the same for Joe. It can be imagined that each category - Employee, Department, Functions - can have more elements, so this structure makes sense.

```
<div id="reactAnchor">
  <div data-reactroot>
    <div style="text-indent: 0px;">
      <p>Employee name: Eddy</p>
      <div style="text-indent: 15px;">
        <p>Department: IT</p>
        <div style="text-indent: 30px;">
          <p>Function: support</p>
          <p>duties:</p>
          <ul>
            <li style="text-indent: 45px;">backup</li>
            <li style="text-indent: 45px;">sysadmin</li>
          </ul>
        </div>
      </div>
    </div>
    <br>
```

It is also possible to re-write this example so that most elements are not nested. That is, we would have the structure (schematically)

```
<div id = "reactAnchor">
        <div>
                <p style="text-indent: ...."> Employee name:...</p>
                <p ....> Department:....</p>
                <p ....> Function</p>
                <p ....> regions/duties/whatever</p->
                <ul>
                        <li.....>.....</li>
                        <li>.....
                        .....
                </ul>
        </div>
</div>
```

Here, each *<p>* element will have its own text-indent style.

Rewriting to get this structure would be a useful didactic exercise. Note that, for each employee, the *<p>* elements and the single <ul> element (with *<li>* children) would have to be returned by the class instances in a single array, and in the correct order.

It is also possible to modify the gjven version so that the indents are cumulative (that is, an indent is inherited from a parent, and then modified) and not assigned explicitly from method arguments. This would be a much more structured way of formatting. This also would be a useful exercise.