Make it Declarative with React

@koba04 / JSConf JP 2019

Toru Kobayashi

- @koba04
  Twitter / GitHub
- Web Developer
  2007〜
- Cybozu
  →Frontend Expert Team
- SmartHR
  →Frontend Advisor

```
ReactVoice.render(
  <>
    <kyoko>こんにちは</kyoko>
    <alex>
        I work as a frontend developer for Cybozu and
        I work as a frontend advisor for SmartHR.
    </alex>
    <alex>My Twitter and GitHub accounts are @koba04</alex>
    <victoria>
        I'm also one of the organizers of React.js meetup in Tokyo and
        a contributor of React.
    </victoria>
    <victoria>I've been working with React for 5years.</victoria>
  </>,
  {}
);
```

Agenda

- Benefits of Declarative Programming for UI
- Custom renderer of React
- Live Demo!

Declarative Programming for UI

Declarative Programming

> In computer science, declarative programming is a programming paradigm—a style of building the structure and elements of computer programs—that expresses **the logic of a computation without describing its control flow**.

> Many languages that apply this style attempt to minimize or eliminate side effects by **describing what the program must accomplish in terms of the problem domain**, rather than describe how to accomplish it as a sequence of the programming language primitives

https://en.wikipedia.org/wiki/Declarative_programming

Why Declarative?

- What Not How
  - How -> Compiler
- Abstraction layer
  - Optimization in the underlying layer
  - Primitive as domain

> The logic of a computation without describing its control flow

DOM manipulation is based on imperative operations

Imperative

```
const view = document.querySelector('.view');
const addButton = document.querySelector('.add-button');

// You have to implement how to update the view
addButton.addEventListener('click', () => {
    view.appendChild(child)
});
```

Declarative

```
const view = document.querySelector('.view');
const addButton = document.querySelector('.add-button');
const state = [];

addButton.addEventListener('click', () => {
    // update the state impleratively
    state.push(child);
    // describe the view declaratively based on the state
    render(state);
});

// describing what the view should display
const render = state => {
    view.innerHTML = state.map(s => `<span>${s}</span>`).join('');
}
```

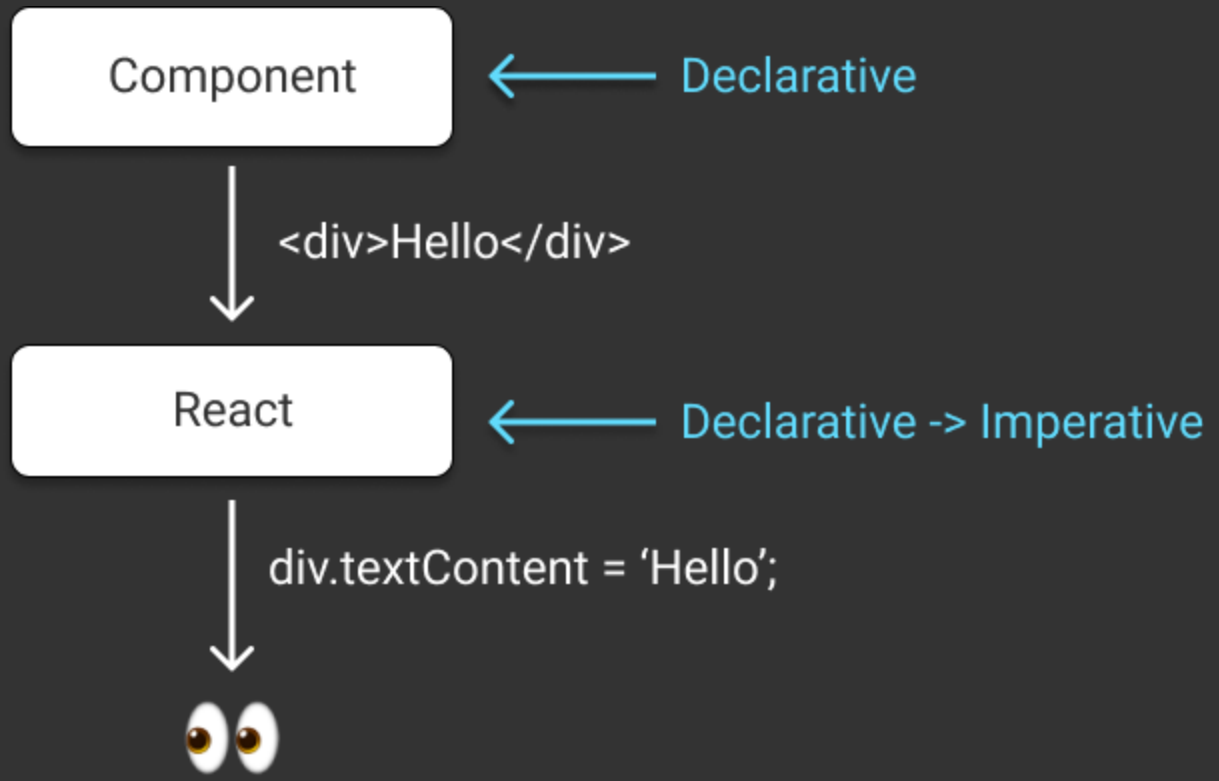
Describing what to update the view

 = View(State)

React updates views efficiently

```
let count = 1;
ReactDOM.render(
    <div>
        <Header />
        <p>{count}</p>
    </div>,
    container
);

count = 2;
ReactDOM.render(
    <div>
        <Header />
        <p>{count}</p>
    </div>,
    container
)
// p.textContent = 2; // React updates the DOM
```

ReactDOM Renderer



> Describing what the program must accomplish in terms of the problem domain

Abstract your application components

- DOM is an implementation detail
- React Component is a primitive of your domain.

Build own domain layers with React

```
const view = document.querySelector('.view');

// describing what the view should display
const App = () => {
    const [items, setItems] = useState([]);
    return (
        <Layout>
            <Header>title</Header>
            <ItemList>
                {items.map(item => <Item key={item.id} item={item} />)}
            </ItemList>
            <AddItemButton
                onAddItem={item => setItems(items.concat(item))}
            />
        </Layout>
    );
}


ReactDOM.render(<App />, view);
```

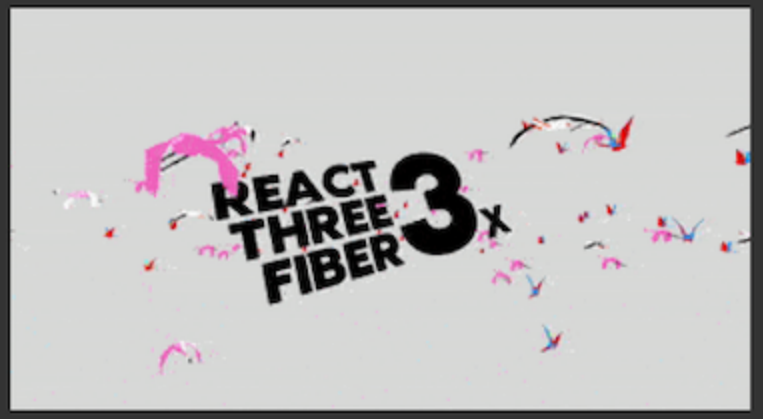
DOM as a Second-class Citizen

Sebastian Markbåge / React Europe 2015

React Custom Renderer

Renderers


```
$ ts-node index.tsx
Hello world!
✨  Done in 2.48s.
```



Source: https://github.com/react-spring/react-three-fiber

Ink

```
import React from 'react';
import {render, Box, Color} from 'ink';

render(
  <Box>
      <Color green>Hello world!</Color>
  </Box>
);
```

ReactKonva

```
ReactKonva.render(
    <Stage width={300} height={300}>
      <Layer>
        <Text text="Hello, world!" fontSize={30} />
        <Star
          x={50}
          y={70}
          innerRadius={20}
          outerRadius={40}
          fill="tomato"
        />
      </Layer>
    </Stage>,
    el
);
```

ReactThreeFiber

```
import React, { useRef } from 'react'
import ReactDOM from 'react-dom'
import { Canvas, useFrame } from 'react-three-fiber'

const Cube = () => {
    const ref = useRef()
    useFrame(() => (ref.current.rotation.x = ref.current.rotation.y += 0.01))
    return (
        <mesh ref={ref}>
            <boxBufferGeometry attach="geometry" args={[1, 1, 1]} />
            <meshNormalMaterial attach="material" />
        </mesh>
    )
}
ReactDOM.render(<Canvas><Cube /></Canvas>, el);
```

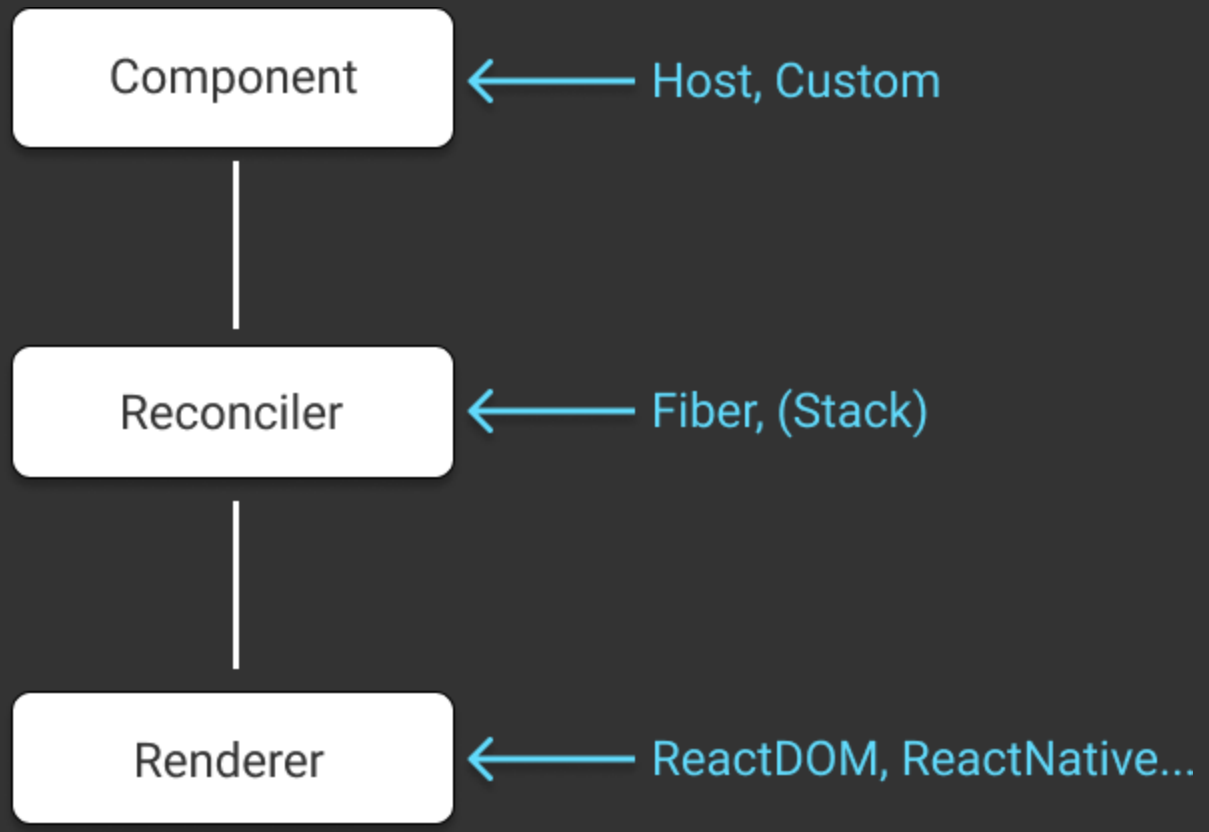
ReactAST

```
import React from 'react';
import {
    renderAst,
    Code,
    ClassDeclaration,
    FunctionDeclaration
} from 'react-ast';

const ast = renderAst(
    <ClassDeclaration name="Hello" superClassName="Array">
        <Code>const hello = 'world'</Code>
        <FunctionDeclaration name="foo">
            <Code>return 'bar'</Code>
        </FunctionDeclaration>
    </ClassDeclaration>
);

console.log(ast);
```

Building a Custom React DOM Renderer

- https://github.com/jquense/react-dom-lite
- https://conf.reactjs.org/event.html?sophiebits

Architecture of React



- https://speakerdeck.com/koba04/algorithms-in-react

react-reconciler

```
npm install react-reconciler
```

packages/react-reconciler

How to use

```
import Reconciler from "react-reconciler";

const renderer = Reconciler(hostconfig);

export const YourReact = {
  render(
    element: React.ReactNode,
    rootContainer: RootContainer,
    callback = () => {}
  ) {
    if (!rootContainer.container) {
      rootContainer.container = {}
      rootContainer.container.fiberRoot = renderer.createContainer(
        container,
        false,
        false
      );
    }
    renderer.updateContainer(element, container.fiberRoot, null, callback);
  }
}
```

HostConfig Interface #1

getPublicInstance, getRootHostContext, getChildHostContext, prepareForCommit, resetAfterCommit, createInstance, appendInitialChild, finalizeInitialChildren, prepareUpdate, shouldSetTextContent, shouldDeprioritizeSubtree, createTextInstance scheduleDeferredCallback, cancelDeferredCallback, setTimeout, clearTimeout, noTimeout, now, isPrimaryRenderer supportsMutation, supportsPersistence, supportsHydration

Mutation(optional)

appendChild, appendChildToContainer, commitTextUpdate, commitMount, commitUpdate, insertBefore, insertInContainerBefore, removeChild, removeChildFromContainer, resetTextContent

HostConfig Interface #2

Persistence(optional)

cloneInstance, createContainerChildSet, appendChildToContainerChildSet, finalizeContainerChildren, replaceContainerChildren

Hydration(optional)

canHydrateInstance, canHydrateTextInstance, getNextHydratableSibling, getFirstHydratableChild, hydrateInstance hydrateTextInstance,didNotMatchHydratedContainerTextInstance, didNotMatchHydratedTextInstance, didNotHydrateContainerInstance, didNotHydrateInstance,didNotFindHydratableContainerInstance, didNotFindHydratableContainerTextInstance, didNotFindHydratableInstance, didNotFindHydratableTextInstance

from @types/react-reconciler

HostConfig of renderers

- ReactDOM
  - packages/react-dom/src/client/ReactDOMHostConfig.js
- ReactNative
  - packages/react-native-renderer/src/ReactNativeHostConfig.js
  - packages/react-native-renderer/src/ReactFabricHostConfig.js
- ReactTestRenderer
  - packages/react-test-renderer/src/ReactTestHostConfig.js
- Ink
  - vadimdemedes/ink/blob/master/src/reconciler.js
- ReactKonva
  - konvajs/react-konva/blob/master/src/ReactKonvaHostConfig.js

HostConfig?

- Side effects for a Host environment
- Define instances
- Define the mode for a renderer
- Hydration logic (if you need)

Side effects for a Host environment

```
ReactDOM.render(
    <ul>
        <li key="a">a</li>
        <li key="b">b</li>
        <li key="c">c</li>
    </ul>,
    container
);

ReactDOM.render(
    <ul>
        <li key="b">b</li>
        <li key="a">a</li>
        <li key="c">c</li>
    </ul>,
    container
)

// React update the DOM like the following
// li.insertBefore(b, a);
```

Side effects for a Host environment

```
export function insertBefore(
  parentInstance: Instance,
  child: Instance | TextInstance,
  beforeChild: Instance | TextInstance
): void {
  // we have to remove a current instance at first
  const index = parentInstance.children.indexOf(child);
  if (index !== -1) {
    parentInstance.children.splice(index, 1);
  }
  // And then, we insert the instance into a new index
  const beforeIndex = parentInstance.children.indexOf(beforeChild);
  parentInstance.children.splice(beforeIndex, 0, child);
}
```

Others

- appendChild, appendInitialChild, appendChildToContainer
- commitTextUpdate, commitMount, commitUpdate
- insertBefore, insertInContainerBefore
- removeChild, removeChildFromContainer, resetTextContent

createInstance, createTextInstance

```
export function createInstance(
  type: Type,
  props: Props,
  rootContainerInstance: Container,
  hostContext: HostContext,
  internalInstanceHandle: OpaqueHandle
): Instance {
  return createYourHostInstance(type, props);
}

export function createTextInstance(
  text: string,
  rootContainerInstance: Container,
  hostContext: HostContext,
  internalInstanceHandle: OpaqueHandle
): TextInstance {
  return createYourTextInstacne(text);
}
```

getPublicInstance

```
export function getPublicInstance(
  instance: Instance
): PublicInstance {
  return convertToPublicInstance(instance);
  // react-dom
  // return instance;
}
```

Define the mode for a renderer

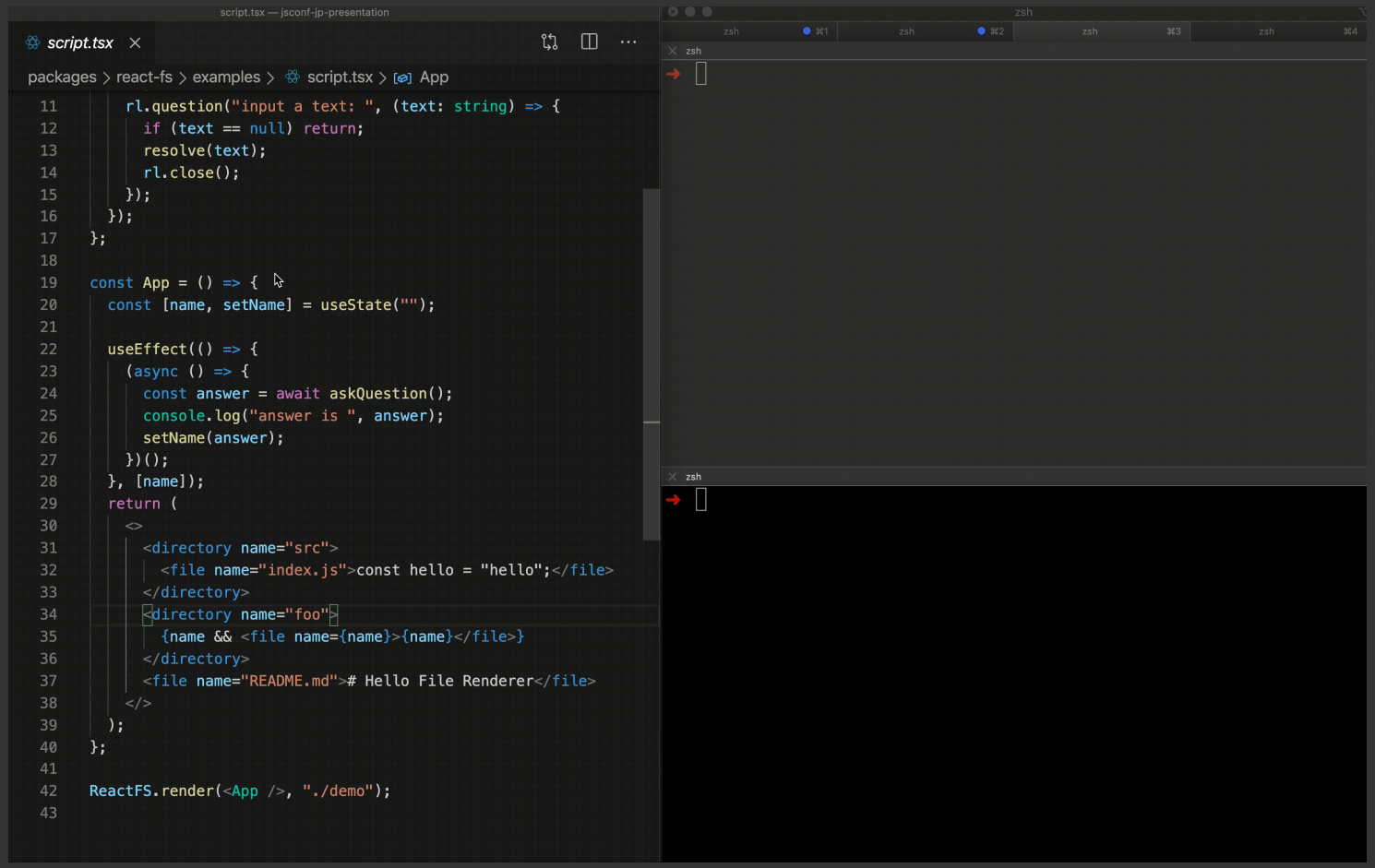
```
export const isPrimaryRenderer = true;
export const supportsMutation = true;
export const supportsPersistence = false;
export const supportsHydration = false;
```

Type Definition for custom host config

```
declare namespace JSX {
  interface IntrinsicElements {
    text: {
      color: string;
      children?: React.ReactNode;
    };
  }
}
```

https://www.typescriptlang.org/docs/handbook/jsx.html#intrinsic-elements

Live Coding




```
    rl.question("input a text: ", (text: string) => {
      if (text == null) return;
      resolve(text);
      rl.close();
    });
  });
};

const App = () => {
  const [name, setName] = useState("");

  useEffect(() => {
    (async () => {
      const answer = await askQuestion();
      console.log("answer is ", answer);
      setName(answer);
    })();
  }, [name]);
  return (
    <>
      <directory name="src">
        <file name="index.js">const hello = "hello";</file>
      </directory>
      <directory name="foo">
        {name && <file name={name}>{name}</file>}
      </directory>
      <file name="README.md"># Hello File Renderer</file>
    </>
  );
};

ReactFS.render(<App />, "./demo");
```

Thank you!!!

- speakerdeck.com/koba04/
- github.com/koba04/jsconf-jp-presentation