Rewrite with React Hooks

Unbug, 2019

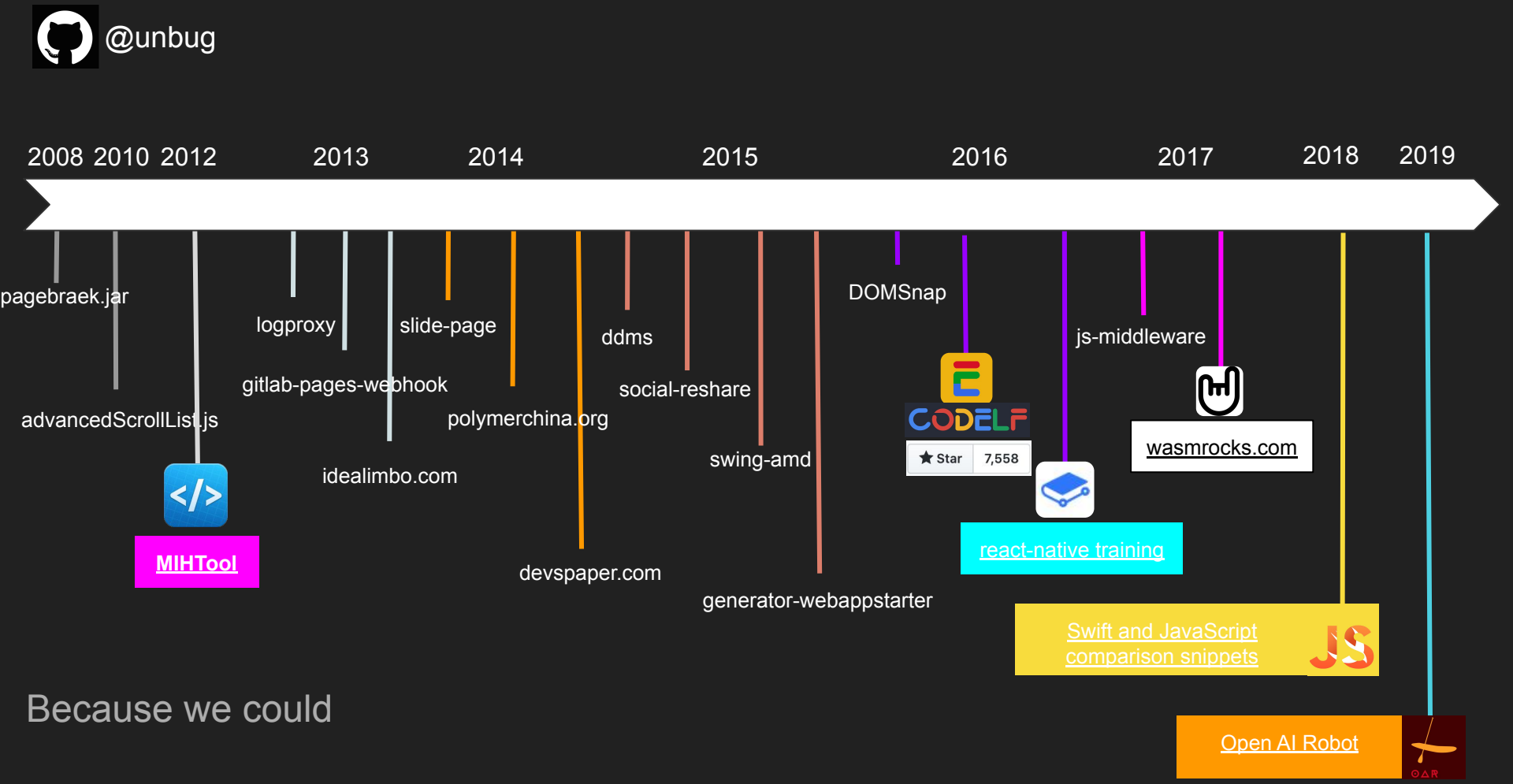

@unbug
2008 2010 2012 2013 2014 2015 2016 2017 2018 2019
pagebraek.jar
advancedScrollList.js
MIHTool
logproxy
gitlab-pages-webhook
idealimbo.com
slide-page
polymerchina.org
devspaper.com
ddms
social-reshare
swing-amd
generator-webappstarter
DOMSnap
CODELF
Star 7,558
react-native training
js-middleware
wasmrocks.com
Swift and JavaScript comparison snippets
JS
Open AI Robot
Because we could

Table of content

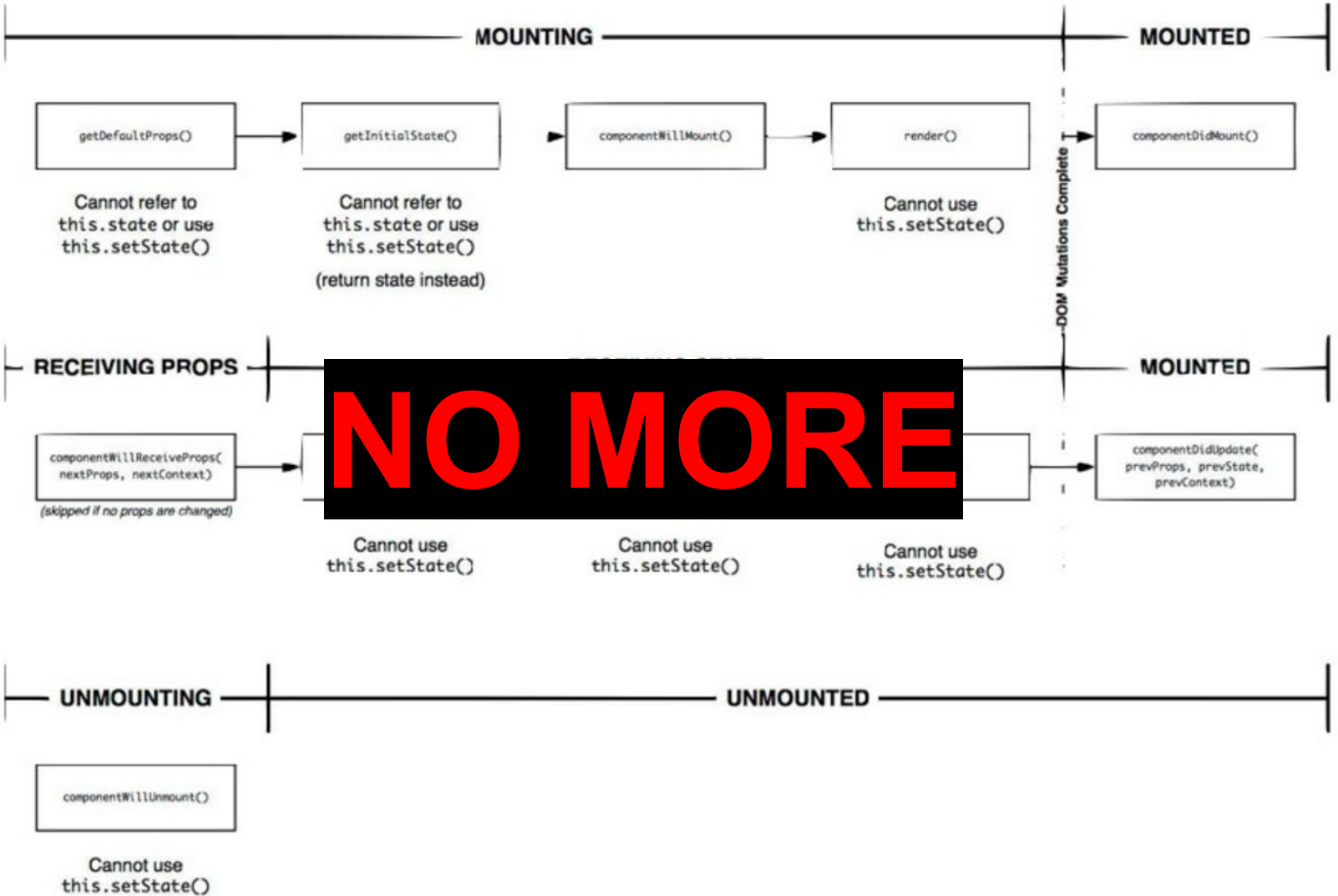
React Hooks gotchas

React Hooks resources

1. Introducing React Hooks.
2. YouTube video—React Today and Tomorrow and 90% Cleaner React With Hooks.
3. React Hooks “Hello World”.
4. All new APIs of React Hooks.
5. Everything you need to know about React Hooks.
6. A Complete Guide to useEffect
7. How Are Function Components Different from Classes?

The good part of React Hooks

1. Manipulate states and interact with component lifecycle methods in your React Functions.
2. Reuse components state/lifecycle logic become possible, and state/lifecycle logic can be tested easily. Reuse components never become so easier.
3. A better Context Providers to avoid “Wrapper Hell”.
4. Avoid frightened by bloated Class Components and no more “xx.bind(this)”, reduce component logic and easy to maintain.
5. Avoid potential performance issues and bugs by making wrong use of component lifecycle method, no more suffering from component lifecycle methods.
6. Integrate with third-party libraries become easier and make a lot of sense.
7. Saving your time from thinking about “state VS props”.
8. Have a better experience with Function Programming and Middleware Programming.

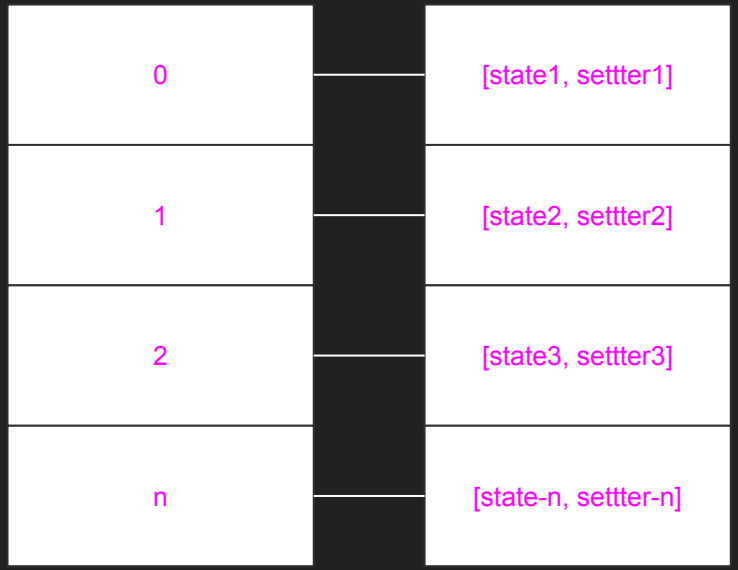

MOUNTING
MOUNTED
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
Cannot refer to this.state or use this.setState()
Cannot refer to this.state or use this.setState()
(return state instead)
Cannot use this.setState()
DOM Mutations Complete
RECEIVING PROPS
MOUNTED
NO MORE
componentWillReceiveProps(nextProps, nextContext)
(skipped if no props are changed)
componentDidUpdate(prevProps, prevState, prevContext)
Cannot use this.setState()
Cannot use this.setState()
Cannot use this.setState()
UNMOUNTING
UNMOUNTED
componentWillUnmount()
Cannot use this.setState()

APIs will be frequently used

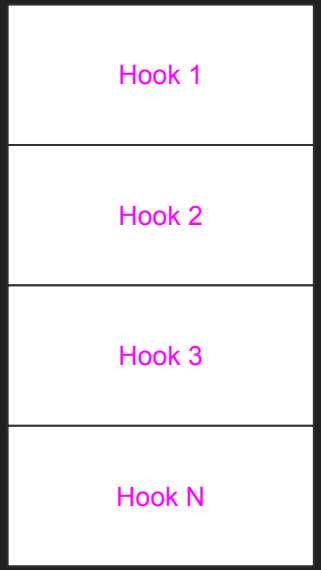
1. useState: manipulate states in your React Functions.
2. useEffect: not just combined the component lifecycle methods of componentDidMount, componentDidUpdate, and componentWillUnmount
3. useContext: return React.createContext as a value, no more “Wrapper Hell”.
4. useReducer: that’s right, ReactJS now ships ReduxJS.
5. useCallback/useMemo: cache expensive calculations to make the render faster.
6. useRef: enhance of React.createRef().
7. useDebugValue: enhance method for React DevTools.

How it works

In array



In order



React Hooks in actions

Requirements

Update ReactJS to 16.8+

```
npm update
```

Install the ESLint Plugin

```
npm install eslint-plugin-react-hooks
```

```
module.exports = {
  //...
  'plugins': [
    'react-hooks'
  ],
  'rules': {
    //...
    'react-hooks/rules-of-hooks': 'error',
  },
  //...
};
```

~#1

The codes before refactor

The challenges

1. Multiple states
2. lifecycle methods(even a getDerivedStateFromProps method)
3. event handlers
4. and react refs

What we gonna do

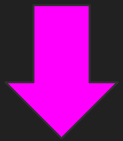
1. Turn the class into a function *export default function SearchBar(props)*
2. Replace all this.props. and this.state. to an empty string.
3. Remove the wrap of render() {//body} function and keep the body codes.
4. Turn all the class methods into pure functions.
5. Refactor class state with React Hooks API useState().
6. Refactor the all the input resize logic and window.addEventListener('resize', this.resizeInput, false) with React Hooks API useEffect()
7. Refactor values created by React.createRef() with React Hooks API useRef(null).

#1~

The codes after refactor

useRef()

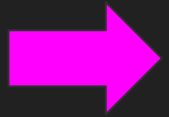
```
32      input = React.createRef();
```



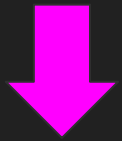
```
32      const inputEl = useRef(null);
```

useState()

```
35   constructor(props) {
36     super(props);
37     this.state = {
38       lang: props && props.searchLang ? props.searchLang : [],
39       prevProps: props,
40       inputSize: 'huge',
41       inputChanged: false
42     }
65   resizeInput = () => {
66     this.setState({inputSize: document.body.offsetWidth < 800 ? '' : 'huge'})
67   }
68
69   handleSearch = () => {
70     this.setState({inputChanged: false});
71     this.props.onSearch(this.input.current.inputRef.value, this.state.lang);
72     this.input.current.inputRef.blur();
73   }
74
75   handleRestLang = () => {
76     this.setState({lang: [], inputChanged: true});
77   }
78
79   handleSelectLang = id => {
80     this.setState({lang: this.state.lang.concat(id).sort(), inputChanged: true});
81   }
82
83   handleDeselectLang = id => {
84     let lang = this.state.lang;
85     lang.splice(this.state.lang.indexOf(id), 1);
86     this.setState({lang: lang.sort(), inputChanged: true});
87   }
```

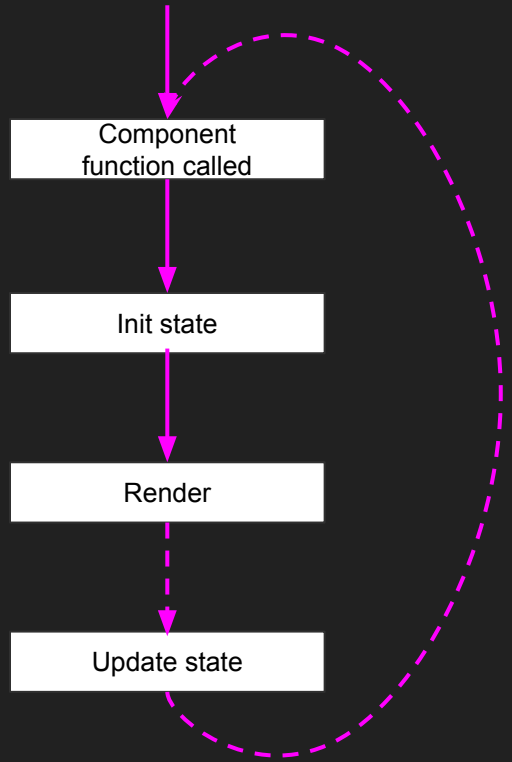


```
34   const [state, setState] = useState({
35     lang: props.searchLang || [],
36     valChanged: false
37   });
38
39   function updateState(vals) {
40     setState(prevState => {
41       return { ...prevState, ...vals };
42     });
43   }
```



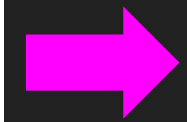
```
34   const [lang, setLang] = useState(props.searchLang || []);
35   const [valChanged, setValChanged] = useState(false);
```

Lifecycle of useState()



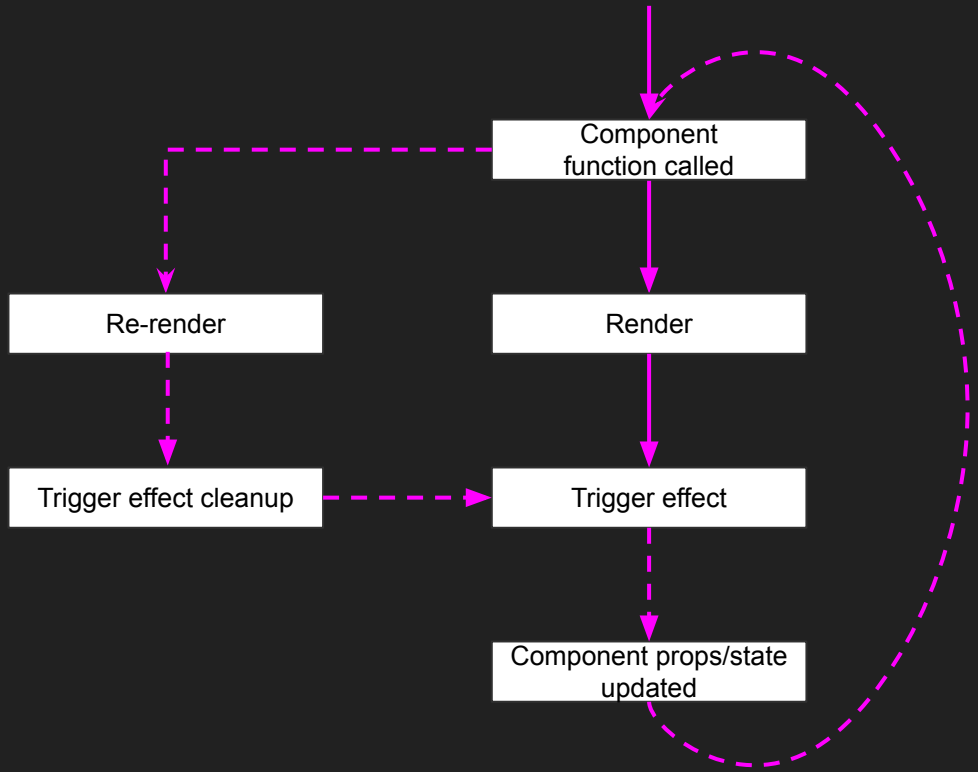
useEffect()

```
35    constructor(props) {
36      super(props);
37      this.state = {
38        lang: props && props.searchLang ? props.searchLang : [],
39        prevProps: props,
40        inputSize: 'huge',
41        inputChanged: false
42      }
43      window.addEventListener('resize', this.resizeInput, false)
44    }
61  componentDidMount() {
62    this.resizeInput();
63  }
64
65  resizeInput = () => {
66    this.setState({inputSize: document.body.offsetWidth < 800 ? '' : 'huge'})
67  }
68
```



```
33  export default function SearchBar(props) {
34    //...
35    const [inputSize, setInputSize] = useState('huge');
36
37    useEffect(() => {
38      resizeInput();
39      window.addEventListener('resize', resizeInput, false);
40      return () => window.removeEventListener('resize', resizeInput, false);
41    }, []);// run an effect and clean it up only once (on mount and unmount),
42
43    function resizeInput() {
44      setInputSize(document.body.offsetWidth < 800 ? '' :'huge');
45    }
46    // ...
47    return (
48      // ...
49      <Input size={inputSize}>{/*...*/}</Input>
50    )
51  }
```

Lifecycle of useEffect()



~#2

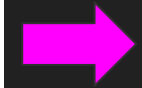
The codes before refactor

The challenges

1. Instance variables (animationName and lastPageLen)
2. Expensive calculation (renderPage()).

useMemo()

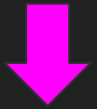
```
renderPage() {
  let pages = [];
  if (notFound(this.props.searchValue)) {
    pages.push(<img style={{maxWidth: '100%'}} src={notFoundImg}/>);
  }
  const pageLen = this.props.variableList.length;
  this.props.variableList.forEach((list, i) => {
    const isLast = i === pageLen - 1 && this.lastPageLen != pageLen;
    const variables = list.map((variable, j) => {
      let style = {}, className = '', duration = (list.length - j) / list.length;
      if (isLast) {
        className = 'animated';
        style = {
          animationName: this.animationName,
          animationDelay: duration + 's',
          animationDuration: Math.min(duration, 0.8) +  Math.random() + 's'
        };
      }
      return <Variable key={Tools.uuid()} variable={variable}
                       onOpenSourceCode={this.props.onOpenSourceCode} style={style} className={className}/>
    });
    if (variables && variables.length) {
      if (pages.length) {
        pages.unshift(<hr/>);
      }
      Array.prototype.unshift.apply(pages, variables)
    }
  });
  this.lastPageLen = pageLen;
  return pages;
}
```



```
const list = useMemo(() => {
  const variableList = props.variableList;
  const pageLen = variableList.length;
  let pages = [];
  if (notFound(props.searchValue)) {
    pages.push(<img style={{ maxWidth: '100%' }} src={notFoundImg} />);
  }
  variableList.forEach((list, i) => {
    const isLast = i === pageLen - 1 && lastPageLen.current != pageLen;
    const variables = list.map((variable, j) => {…
    });
    if (variables && variables.length) {…
    }
  });
  lastPageLen.current = pageLen;
  return pages;
}, [props.variableList]);
```

useRef()

```
50
51      lastPageLen = 0;
52      animationName = Math.random() > 0.5 ? 'zoomInDown' : 'zoomInUp';
53
```



```
7   const animationName = Math.random() > 0.5 ? 'zoomInDown' : 'zoomInUp';
8
9   export default function VariableList(props) {
10    const lastPageLen = useRef();
11    const list = useMemo(() => {
12      const variableList = props.variableList;
13      const pageLen = variableList.length;
14      let pages = [];
15      if (notFound(props.searchValue)) {
16        pages.push(<img style={{ maxWidth: '100%' }} src={notFoundImg} />);
17      }
18      variableList.forEach((list, i) => {
19        const isLast = i === pageLen - 1 && lastPageLen.current != pageLen;
20        const variables = list.map((variable, j) => {…
32        });
33        if (variables && variables.length) {…
38        }
39      });
40      lastPageLen.current = pageLen;
41      return pages;
42    }, [props.variableList]);
```

#2~

The codes after refactor

Integrating with third-party libraries

~#3

The codes before refactor

The challenges

1. Multiple third-party libraries in components
2. Duplicate logic in multiple components

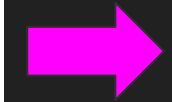
Custom React Hooks

```
componentDidUpdate(prevProps, prevState, snapshot) {
  if (prevProps.sourceCode != this.props.sourceCode || (!this.visiable &&
    this.renderPrettyPrint();
    this.visible = true;
  }
}

componentDidMount() {
  this.renderPrettyPrint();
}

handleClose = () => {
  this.visiable = false;
  this.props.onCloseSourceCode();
}
renderPrettyPrint = () => {
  setTimeout(() => {
    if (this.code.current) {
      this.code.current.classList.remove('prettyprinted');
      setTimeout(() => PR.prettyPrint(
        () => setTimeout(() =>this.renderHighLight(), 1000)
      ), 100);
    }
  }, this.code.current ? 0 : 1000);
}

renderHighLight = () => {
  if (this.mark) {this.mark.unmark()}
  this.mark = new Mark(this.code.current);
  let idx = 0;
  this.mark.mark(this.props.sourceCodeVariable.keyword, {each: el => {
      el.setAttribute('tabindex',idx++);
    }});
}
```



```
import { useEffect, useRef } from 'react';

export default function useCodeHighlighting(watchedProps, keyword) {
  const container = useRef(null);
  const mark = useRef(null);
  useEffect(() => {
    renderPrettyPrint();
  }, [...watchedProps]);

  function renderPrettyPrint() {…
  }

  function renderHighLight() {…
  }

  return container;
}
```

```
import useCodeHighlighting from './hooks/useCodeHighlighting';

export default function SourceCode(props) {
  const codeEl = useCodeHighlighting([props.sourceCode, props.sourceCodeVisible], props.sourceCodeVariable?.keyword);

```

```
import useCodeHighlighting from './hooks/useCodeHighlighting';

export default function Copybook(props) {
  const codeEl = useCodeHighlighting([props.copybookFileContent, props.copybookVisible]);
```

#3~

The codes after refactor

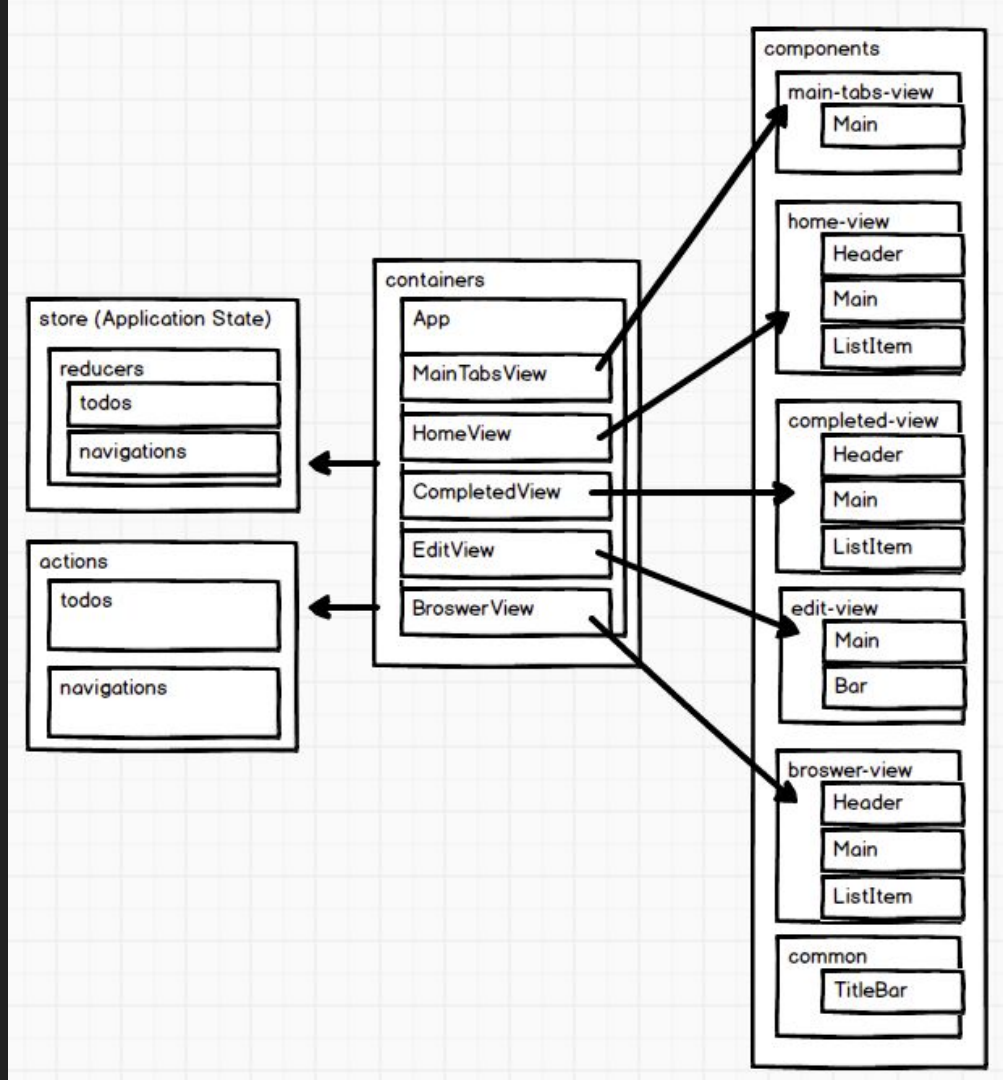
More Custom React Hooks

https://usehooks.com/
https://github.com/gragland/usehooks

useAuth
useEventListener
useWhyDidYouUpdate
useDarkMode
useMedia
useLockBodyScroll
useTheme
useSpring
useHistory
useScript
useKeyPress
useDebounce
useOnScreen
usePrevious
useOnClickOutside
useAnimation
useWindowSize
useHover
useLocalStorage

Refactor a Container component

Container pattern



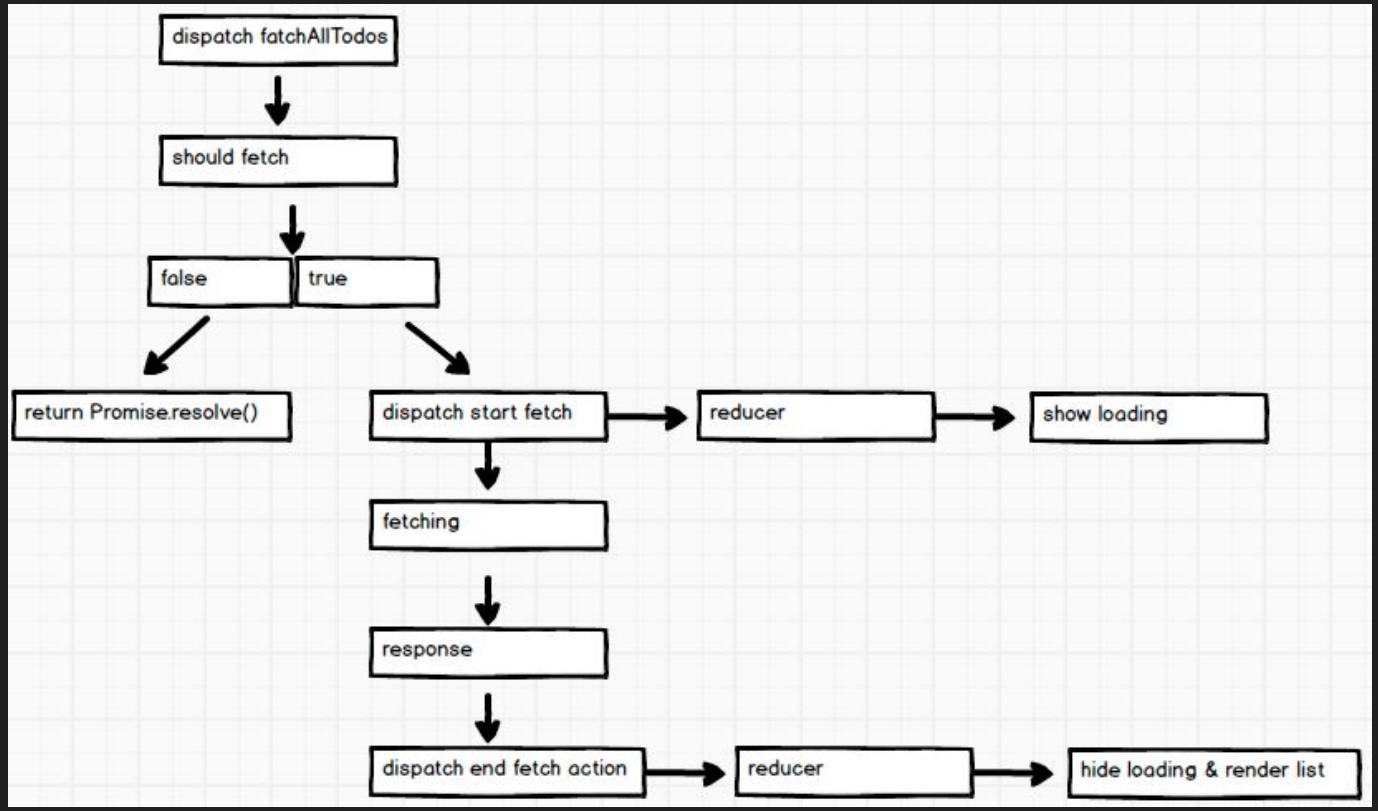
~#4

The codes before refactor

The challenges

1. Too many states
2. Update multiple states in a same place
3. Too much efforts to turn each state property into a useState()

A case of ReduxJS



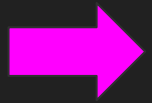
#4~

The codes after refactor

useReducer()

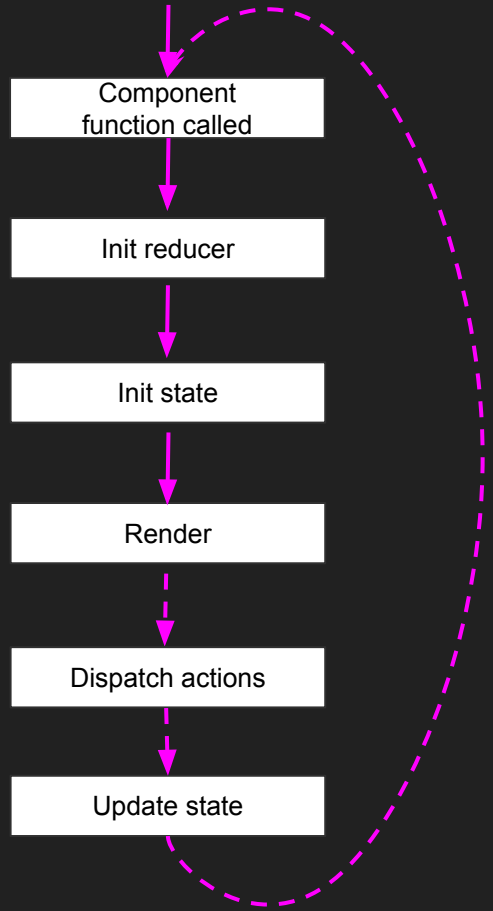
```
21    state = {
22      isZH: false,
23      isError: false,
24      requestingVariable: false,
25      searchValue: SearchCodeModel.searchValue,
26      searchLang: SearchCodeModel.searchLang,
27      page: SearchCodeModel.page,
28      variableList: SearchCodeModel.variableList,
29      suggestion: SearchCodeModel.suggestion,
30      requestingSourceCode: false,
31      sourceCodeVisible: false,
32      sourceCodeVariable: null,
33      sourceCodeRepo: null,
34      requestingCopybook: false,
35      copybookVisible: false,
36      copybookFileList: CopybookModel.fileList,
37      copybookSelectedFile: CopybookModel.selectedFile,
38      copybookFileContent: CopybookModel.fileContent,
39    }
```

```
115         this.setState({
116           isZH: SearchCodeModel.isZH || this.state.isZH,
117           isError: this.checkError(curr),
118           requestingVariable: !mutation.variableList,
119           searchValue: SearchCodeModel.searchValue,
120           searchLang: SearchCodeModel.searchLang,
121           page: SearchCodeModel.page,
122           variableList: SearchCodeModel.variableList,
123           suggestion: SearchCodeModel.suggestion
```



```
2   import React, { useEffect, useReducer, useCallback } from 'react';
3
4   const actionTypes = {
5     UPDATE: 'update',
6   };
7   const initState = {
8     isZH: false,
9     isError: false,
10    variableRequesting: false,
11    //...
12    sourceCodeRepo: null,
13  };
14  function reducer(state, action) {
15    switch (action.type) {
16      case actionTypes.UPDATE:
17        return {
18          ...state,
19          ...action.payload
20        };
21      default:
22        return state;
23    }
24  }
25  export default function MainContainer(props) {
26    const [state, dispatch] = useReducer(reducer, initState);
27
28    // ....
29    const handleSearch = useCallback((val, lang) => {
30      if (val === null || val === undefined || state.variableRequesting) {
31        return;
32      }
33      val = val.trim().replace(/\s+/ig, ' '); // filter spaces
34      if (val.length < 1) {
35        return;
36      }
37      if (val == state.searchValue) {
38        requestVariable(val, lang);
39      } else {
40        setState({ searchLang: lang });
41        setTimeout(() => HashHandler.set(val)); // update window.location.hash
42      }
43    }, [state.searchValue, state.variableRequesting]);
44
45    //...
46    function setState(payload) {
47      dispatch({ type: actionTypes.UPDATE, payload: payload });
48    }
49
50    //...
51  }
```

Lifecycle of useReducer()



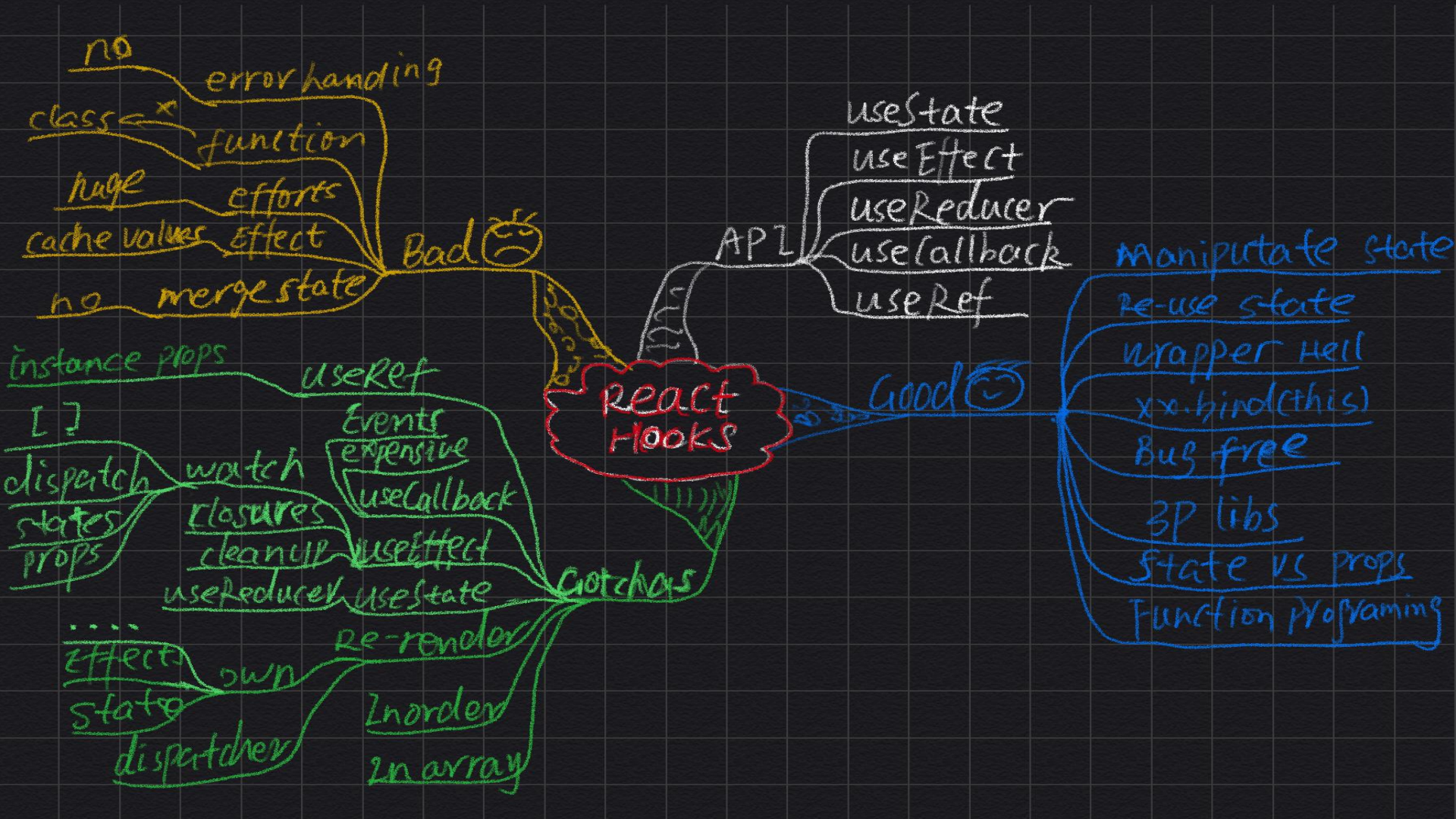
useState() vs useReducer()

Off course, useState({...}) can do the same thing. But useReducer() is easier to define actions to handle complicated logic and keep the component clean, which is making it more scalable. useState() is great for “logicless” component. That’s a big difference use case between useState() and useReducer() .

Summary

The bad part

1. useState() doesn't return a setter and it won't merge new state into the old state automatically, that's sucks then setState() of React Class Components.
2. React effects run on every update. Which make cache local values as instance properties in React Class Components is difficult. We have to be very careful otherwise it will lead to bugs. Unfortunately, the ESLint plugin doesn't cover this kind of case. Such as, we define a local variable let val = null then update the val somewhere in the component, but if an update has been triggered, the val will be reset as null again.
3. A completely rewritten for a large project will cost a huge effort. But if we don't rewrite all the components, reuse new components state logic for old components is impossible, we also can't reuse state logic in a React Class components, duplicate components, and codes will be made.
4. You can turn a React Function into a React Class, or turn a React Class into a React Function. But turn a React Function with React Hooks into React Class is hard, sometimes even impossible. It will be a pain to copy codes from a project to another project.
5. There is no way to handle Error Boundaries with React Hooks right now, which means you won't be able to refactor a React Class that handle errors with componentDidCatchand getDerivedStateFromError . But it is a very common use case, right?


React Hooks
Bad
no error handling
class <-x-> function
huge efforts
cache values Effect
no merge state
API
useState
useEffect
useReducer
useCallback
useRef
Good
Manipulate state
Re-use state
Wrapper Hell
xx.bind(this)
Bug free
3P libs
State VS Props
Function Programing
Gotchas
instance props
useRef
[ ]
dispatch
states
props
watch
Events
expensive
useCallback
closures
cleanup
useEffect
useReducer
useState
....
Effects
states
dispatcher
own
Re-render
1norder
2n array

Thanks

Read it on Medium