# 使用 React&Redux 开发 现代WEB应用

GrapeCity
Shi Kelong

# About Leyser System

## Leyser System

- *School Management Software*
- *20+ years history, 9+ major version, 3000+ school user.*
- *Version 9 is based on Winform.*
- *Serve 3000 private schools in Japan*

## Leyser Plus

- *Include several single page application (Desktop / Mobile)*
- *Front end technology stack - React, Redux, React-Bootstrap, Webpack.*

# Modern Web Application

- Architecture
- Asset packaging
- Run-time state

# Modern Web Application

- Models as the single source of truth.
- Views observe model changes.
- Minimizing DOM dependent-code.
- Maybe Single-Page Application

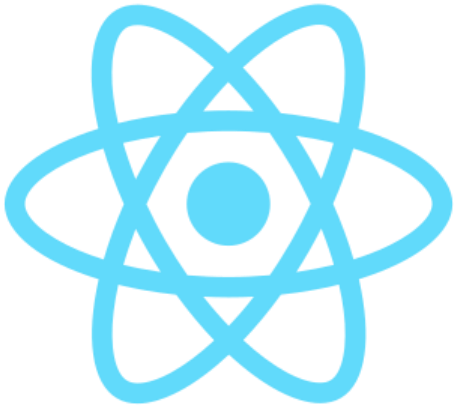
*Quated from http://singlepageappbook.com*

# OUTLINE

- React
- Redux
- Practices

# What Is React?

- A JavaScript **library** for building user interfaces
  - Declarative
  - Component-Based
  - Learn Once, Write Anywhere
- The V in MV* pattern



# React: JSX

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

- JSX just provides syntactic sugar for the React.createElement function.
- Requires compilation process like Babel.

# React: Components and Props

- Components let you split the UI into independent.
- Components let you reusable pieces, and think about each piece in isolation.
- It like Javascript functions. It accept inputs(called props) and return React elements.

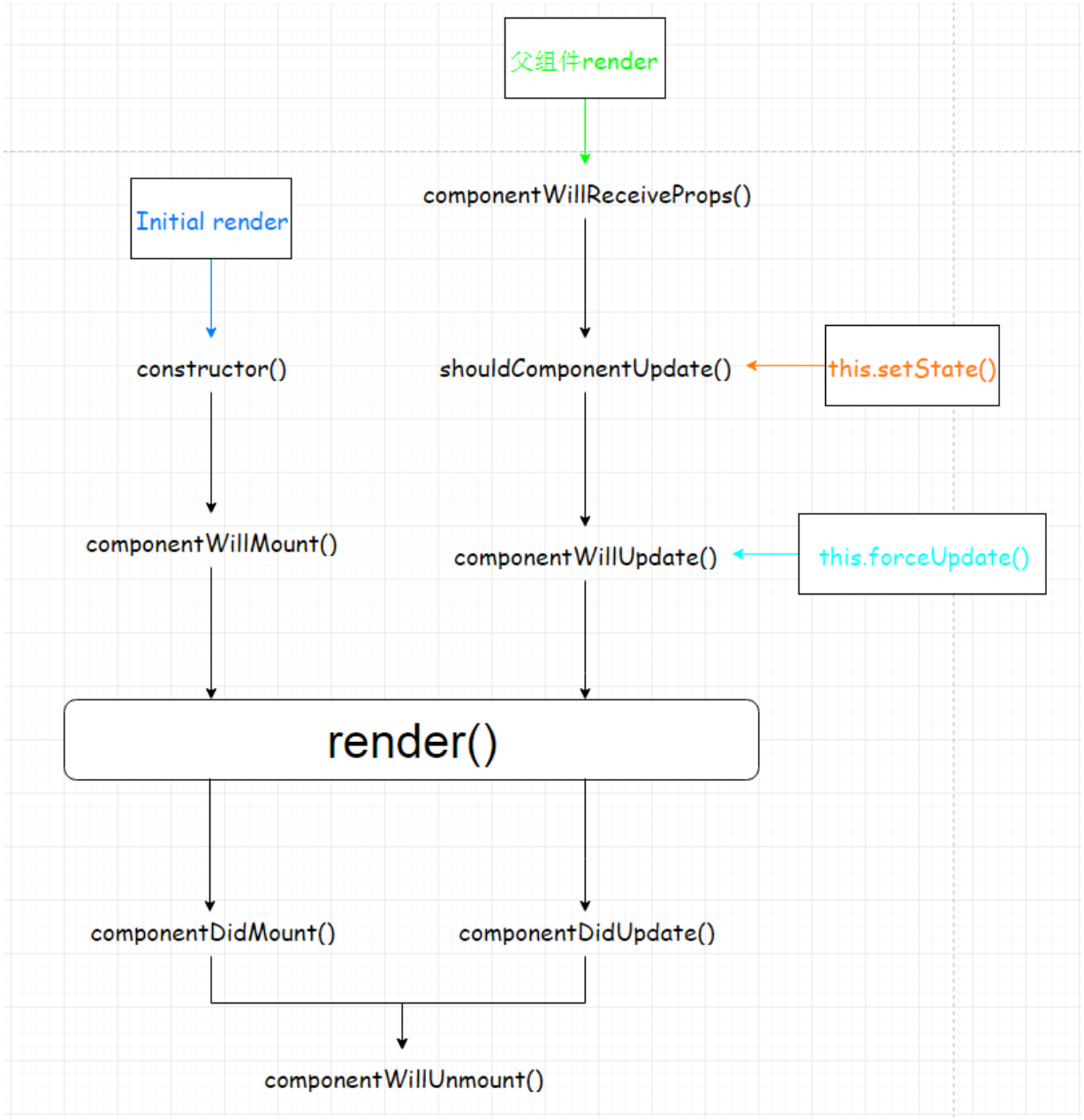
```
class HelloWorld extends React.Component {
  static propTypes = {
    name: React.PropTypes.string.isRequired
  }

  render() {
    return (<div Hello, {this.props.name}. </div>);
  }
}

ReactDOM.render(
    <HelloWorld name="world" />,
    document.getElementById("root")
);
```

# React: Components and Props

- Props are Read-Only
- Anyting can be passed as a prop.
- All React components must act like pure functions with respect to their props.

# React: State and Lifecycle

- State is Component's local data.
- State is update by calling this.setState
- You should not modify state directly.
- State Updates May Be Asynchronous.
- Every call to setState triggers a re-render.




# Component Life-cycle

*Quoted from*
*http://www.jianshu.com/p/4784216b8194*

# React: Container/Presentational Components

- "Container"/ "Smart" components
  - Are concerned with how things work.
  - May contain both presentational and container components, but usually don't have any DOM markup of their own
  - Provide the data and behavior to presentational or other container components.
  - Are usually generated using *higher order components*.

# React: Container/Presentational Components

- "Presentational" / "Dumb" components.
  - Are concerned with how things look.
  - May contain both presentational and container components inside and usually have some DOM markup and styles of their own.
  - Receive data and callbacks exclusively via props.
  - Rarely have their own state (when they do, it's UI state rather than data).

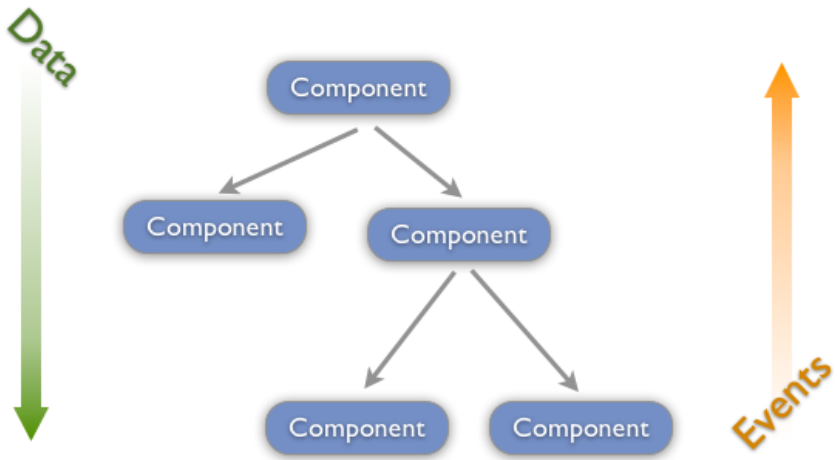
*https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0*

# React: Higher-Order Components

- A higher-order component (HOC) is an advanced technique in React for reusing component logic.
- A higher-order component is just a function that takes an existing component and returns another component that wraps it.
- HOC is a patterns for composition.

# React: Communication between Components

- parent component to child or lower-level component: props, context.
- child to parent or high-level component: callback.
- other: subscribe/publish global event.



# React: Integrating with Other Components

- React Ecosystem can't provide all components or Libraries we need. So we need integrate with other libraries/components.

- We integrate react with *SpreadJS*, *Echarts*, etc.

# React: Integrating with Other Components

- You should initialize the third-party component in suitable life-cycle hooks.
- You should let the third-party component management it' own display and behavior.
- So, Just pass / update your react component's props to the third-party component.
- Don't forget dispose the third-party component' instance.

# What Is Redux?

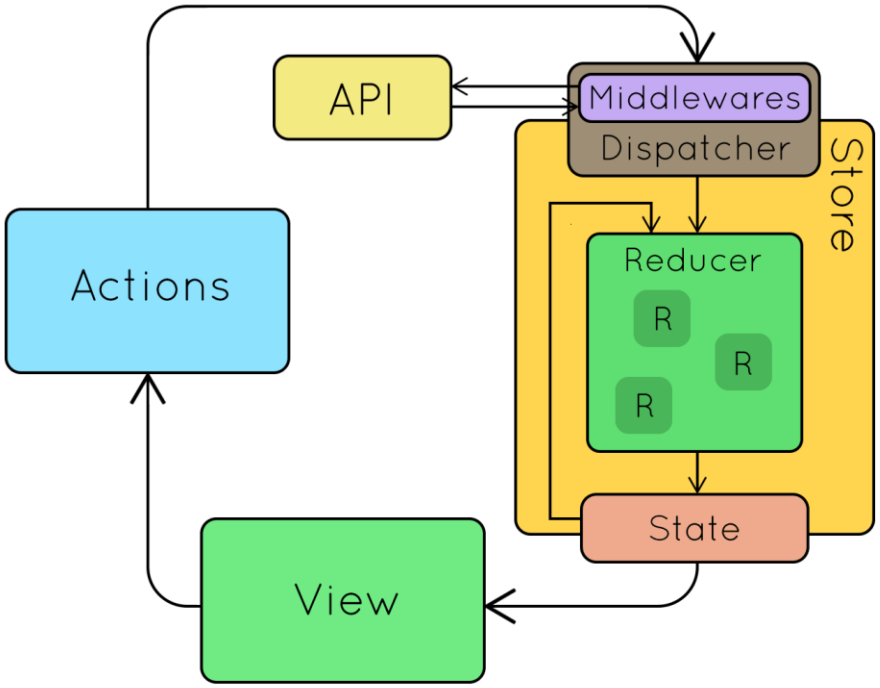
- Redux is a predictable state container for JavaScript apps.
- You can use Redux together with React, or with any other view library.
- Redux evolves the ideas of Flux, but avoids its complexity.

Redux

# Redux : Tree Principles

- Single source of truth: The only way to change the state is to emit an action, an object describing what happened.
- State is read-only: The only way to change the state is to emit an action, an object describing what happened.
- Changes are made with pure functions: To specify how the state tree is transformed by actions, you write pure functions called reducers, which are (state, action) => newState.

# Redux Data Flow



# Redux : Store

- A Redux store contains the current state value. Stores are created using the createStore method.
- Stores have three main methods: dispatch, getState, and subscribe. All subscription callbacks are invoked at the end of every call to dispatch.

# Redux : Actions / Action Creators

- Actions are payloads of information that send data from your application to your store. An action is a plain JS object.
- Action creator is a function that create action. The use of action creators leads to cleaner code and better resuability.

# Redux : Reducers

- All state update logic lives in functions called reducers.

  Since they're just functions, smaller functions can be composed together into larger functions.

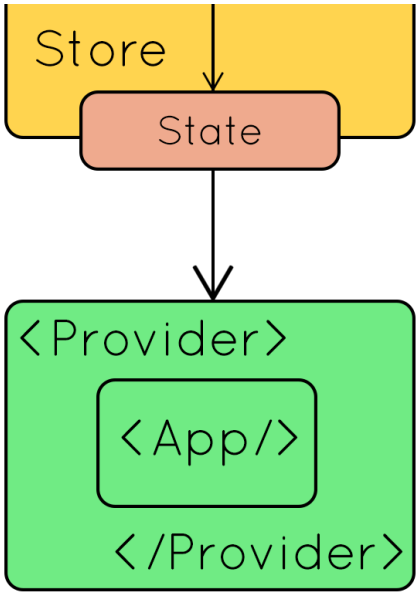
- Reducers are functions like : (state, action) => newState.
- Reducers should be pure functions, with no side effects.
- Reducers need to update data/state immutably.

# Why Redux?

- Centralized state handling
- Predictable state updates
- An Easy way to share data in different component without pass all data as props down from top-level components
- Hot Module Replacement  etc.

# Using Redux with React

```
import {connect} from 'react-redux';

class UserList extends PureComponent {
  ...
}

const mapStateToProps = (state) => {
  return {
    users: state.wages
  };
};

const mapDispatchToProps = {
  getUsers
};

export default connect(mapStateToProps, mapDispatchToProps)(UserList);
```




# PRACTICES

# Code Quality/Style Check

- Eslint
- Stylelint
- SonarQube：SonarJS

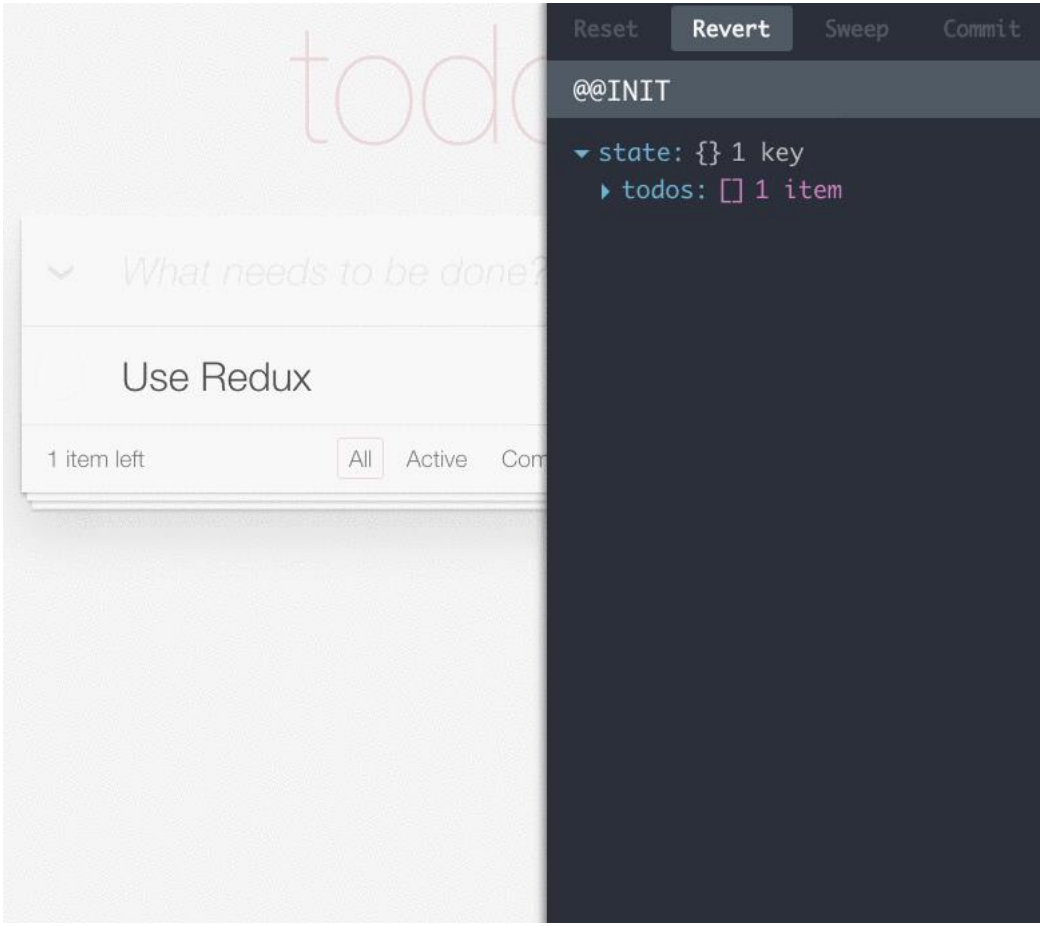
# Dev Tools : react-addons-perf

| Component | Total time (ms) ▼ | Instance count | Total render time (ms) | Average render time (ms) | Render count |
|---|---|---|---|---|---|
| "Modal" | 31.09 | 10 | 6.9 | 0.11 | 62 |
| "LogMonitor" | 21.03 | 1 | 1.2 | 0.12 | 10 |
| "JSONNestedNode" | 9.28 | 32 | 8.99 | 0.28 | 32 |
| "CommonModal" | 8.35 | 5 | 8.32 | 0.29 | 28 |
| "Portal" | 7.82 | 2 | 0.01 | 0 | 6 |
| "LogMonitorEntryList" | 5.79 | 1 | 5.73 | 0.71 | 8 |
| "ChangeUserAccountModal" | 3.82 | 1 | 3.82 | 1.27 | 3 |
| "JSONTree" | 3.62 | 14 | 1.35 | 0.09 | 14 |
| "MessageDetailModal" | 3.36 | 1 | 3.21 | 0.53 | 6 |
| "Dock" | 2.92 | 1 | 2.74 | 0.17 | 16 |
| "JSONArrow" | 2.81 | 32 | 2.81 | 0.08 | 32 |
| "Notifications" | 2.8 | 1 | 2.8 | 0.46 | 6 |
| "JSONObjectNode" | 2.66 | 32 | 2.66 | 0.08 | 32 |
| "JSONNode" | 2.64 | 36 | 2.64 | 0.07 | 36 |

# Dev Tools : redux-devtools

- redux-devtools-log-monitor
- redux-devtools-dock-monitor

# Dev Tools : redux-devtools



# Test React Components

- Test Framework: *Mocha*
- Use *react test utils addons* to test component (Virtual DOM or DOM)
- *Enzyme* provide Jquery-Like api.

# Test React Components

```
import _ from 'lodash';
import React from 'react';
import sinon from 'sinon';
import { expect } from 'chai';
import { mount } from 'enzyme';

import SwitchButton from '$/SwitchButton';

describe('[Common Components] SwitchButton', () => {
  const switchButtonMounted = mount(<SwitchButton />);

  describe('Test Actions', () => {
    it('should call onChange when status changed', () => {
      const callBack = sinon.spy();
      switchButtonMounted.setProps({ onChange: callBack });
      switchButtonMounted.find('input').simulate('change');
      expect(callBack.calledOnce).to.equal(true);
    });
  })
})
```

# Redux: Reducing Boilerplate

- Generating Action Creators
  - utility libraries: *redux-act, redux-actions*
  - simple utility function:

```
const createActionCreator = (type, ...argNames) => (...args) => {
    const action = {type};
    argNames.forEach((arg, index) => {
      action[argNames[index]] = args[index];
    });
    return action;
  };

export const addTodo = createActionCreator('ADD_TODO', 'text');
```

# Redux: Reducing Boilerplate

- Generating Reducers

```
function createReducer(initialState, handlers) {
  return function reducer(state = initialState, action) {
    if (handlers.hasOwnProperty(action.type)) {
      return handlers[action.type](state, action)
    } else {
      return state
    }
  }
}
```

# Redux: Reducing Boilerplate

- Generating Reducers

```
const userInfo = {
  [types.GET_EMPLOYEE_INFO_DETAIL_SUCCESS]: (state, action) => {
    //...
  },
  [types.UPDATE_EMPLOYEE_INFO_SUCCESS]: (state, action) => {
    //...
  }
};

const employeeInfoDetail = createReducer(null, userInfo);
```

# Improving React and Redux Perf with Reselect

```
import { createSelector } from 'reselect'

const getVisibilityFilter = (state) => state.visibilityFilter
const getTodos = (state) => state.todos

export const getVisibleTodos = createSelector(
  [ getVisibilityFilter, getTodos ],
  (visibilityFilter, todos) => {
    switch (visibilityFilter) {
      case 'SHOW_ALL':
        return todos
      case 'SHOW_COMPLETED':
        return todos.filter(t => t.completed)
      case 'SHOW_ACTIVE':
        return todos.filter(t => !t.completed)
    }
  }
)
```

# Improving React and Redux Perf with Reselect

- Selectors can compute derived data, allowing Redux to store the minimal possible state.
- Selectors are efficient. A selector is not recomputed unless one of its arguments change.
- Selectors are composable. They can be used as input to other selectors.

# Webpack : Code Splitting with react-router

- Entry Points: Manually split code using entry configuration.
- Prevent Duplication: Use the *CommonsChunkPlugin* to dedupe and split chunks.
- Dynamic Imports: Split code via inline function calls within modules.

# Webpack : Code Splitting - Dynamic Imports

```
module.exports = {
  name: '保護者グループ設定',
  path: 'guardianGroupManagement',
  getComponent(nextState, cb){
    require.ensure([], (require) => {
      cb(null, require('./components/GuardianGroupManagementPage').default);
    });
  }
};
```

# Webpack : Code Splitting - Prevent Duplication

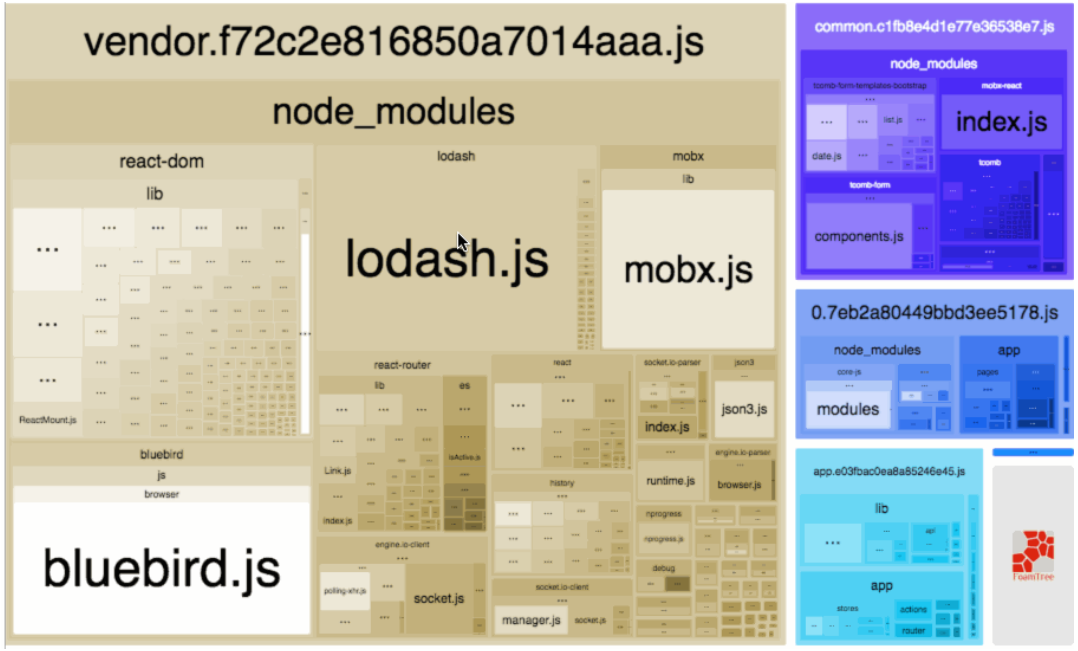
```
plugins: [

  new webpack.optimize.CommonsChunkPlugin({
    name: 'vendor',
    filename: 'vendor.[hash].js',
    minChunks: Infinity
  }),
```

# Webpack : Code Splitting - Bundle Analysis

- *webpack-bundle-analyzer*

```
var BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;

// ...
plugins: [new BundleAnalyzerPlugin()]
// ...
```

# Webpack : Code Splitting - Bundle Analysis



# Import Module from Root - In JS

*npm install babel-plugin-root-import --save-dev*

```
  // Usually
import SomeExample from '../../../some/example.js';
const OtherExample = require('../../../other/example.js');

// With Babel-Root-Importer
import SomeExample from '~/some/example.js';
const OtherExample = require('~/other/example.js');
```

# Import Module from Root - In JS

in *.babelrc* file

```
{
  "plugins": [
    ["babel-plugin-root-import", [{
      "rootPathPrefix": "~",
      "rootPathSuffix": "app"
    }, {
      "rootPathPrefix": "@",
      "rootPathSuffix": "app/assets"
    }, {
      "rootPathPrefix": "$",
      "rootPathSuffix": "app/components"
    }]]
  ]
}
```

# Import Module from Root - In CSS

```
{
  loader: 'postcss-loader',
  options: {
    plugins: function () {
      return [
        //...
        require("postcss-import")({ path: ['./app'] }),
        //...
      ];
    }
  }
}
```

```
@import "assets/css/consts.css";

@import "../../../../../assets/css/consts.css";
```

# i18n

- Generate language resource (JSON) before webpack build.
  - *use glob to find source files path (eg. './app/**/en-US.json')*
- Use utility function get text.
- Other library: *react-intl*

# Summary : Why React ?

- Component-based
- Flexible (only V in MV* )
- Uni-direction data flow
- Efficient (virtual-dom)
- Ecosystem

# Summary : Why Redux ?

- Centralized state handling
- Predictable state update
- Share data in components.

# Q&A

# Thank you ☺