# node-study

## test pagination

React component

## Install

#### Install node module package

$ cd test-pagination  
$ npm install

## Compile and build

$ npm run build

#### You also can use this command

$ webpack

#### Build with press

$ webpack -p

## Develop

$ npm run dev

## Usage

//pagination.test.js  
  
import React from 'react';  
import ReactDom from 'react-dom';  
import Pagination from './components/Pagination';  
  
ReactDom.render(  
 <Pagination   
 pageSize={10}   
 total={100}  
 />,   
document.getElementById('example'));

#### Add react.js/react-dom.js in your html files

<!DOCTYPE>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <title>Test pagination</title>  
 <meta name="description" content="test pagination" />  
 <script src="https://npmcdn.com/react@15.3.1/dist/react.js"></script>  
 <script src="https://npmcdn.com/react-dom@15.3.1/dist/react-dom.js"></script>  
 </head>  
 <body>  
 </body>  
</html>

#### Add the component

<!DOCTYPE>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <title>Test pagination</title>  
 <meta name="description" content="test pagination" />  
 <script src="https://npmcdn.com/react@15.3.1/dist/react.js"></script>  
 <script src="https://npmcdn.com/react-dom@15.3.1/dist/react-dom.js"></script>  
 </head>  
 <body>  
 <div id="example"></div>  
 <script src="http://localhost:3001/js/pagination.test.js"></script>  
 </body>  
</html>

## API

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Description | Type | Default |
| defaultCurrent | default current page | Number | 1 |
| current | current page | Number | undefined |
| total | items total count | Number | 0 |
| defaultPageSize | default items per page | Number | 5 |
| pageSize | items per page | Number | 10 |
| onChange | page change callback | Function([changedTo]) | - |
| pageSelect | show page size select | Bool | false |
| selectOptionsPageSize | specify the sizeChanger selections | Array | [10, 20, 30, 40, 50] |
| className | className of pagination | String | - |
| simplePager | when set, show simple pager | Bool | false |

## 基于React实现的分页组件

本项目是使用ECMAScript 2015的语法,并基于React 框架实现的分页组件.

### 最终效果如下

### 项目完整目录结构如下

├─dist  
│ └─js  
│ pagination.js  
│ pagination.test.js  
├─doc  
│ └─img  
│ pagination.gif   
├─example  
│ index.html  
├─node\_modules  
├─src  
│ │ index.js  
│ │ index.test.js  
│ ├─components  
│ │ Buttons.js  
│ │ Pagination.js  
│ │ Select.js  
│ ├─lib  
│ │ mc-pagination-cal.js  
│ .babelrc  
│ .eslintignore  
│ .eslintrc  
│ package.json  
│ README.md  
│ webpack.config.js

#### 目录结构说明

* dist: 存放编译好的js文件,由webpack配置生成
* doc/img: 存放文档资源文件
* example: 存放demo/实例文件
* node\_modules: node依赖包,包含react/lodash/webpack等
* src: 项目开发目录
* src/components: 存放组件
* src/lib: 其他工具库
* src/index.js: 分页组件导出
* src/index.test.js: 分页组件demo实例
* .babelrc: babel编译工具配置文件
* .eslintignore: ESLint工具过滤器配置文件
* .eslintrc: ESLint工具配置文件
* webpack.config.js: webpack配置文件
* README.md: 项目说明文档
* package.json: 项目配置文件

### 准备

本项目主要基于node.js,由于使用ECMAScript 2015语法,所以需要babel/webpack等工具编译/压缩;用ESLint等工具进行语法检查和校验,所以再进行下一步操作之前,请确保系统含有node.js环境

在本项目中需要

node.js v6.0+  
npm v3.0+

node.js下载地址:

https://nodejs.org/en/

建议进行全局安装webpack工具

$ npm install webpack -g

### 初始化和配置项目

新建项目目录

$ mkdir test-pagination

切换至项目目录下并初始化项目

$ cd test-pagination  
$ npm init

填写项目配置package.json,请注意不可忽略的选项

name: (test-pagination) //项目名称,可忽略  
version: (1.0.0) //项目版本,可忽略  
description: A pagination component base on React // 项目描述,可忽略  
entry point: (index.js) // 默认入口文件,可忽略  
test command: // 测试用命令,可忽略  
git repository: //git仓库,可忽略  
keywords: // 可忽略  
author: Jun // 作者  
license: (ISC) // 开源协议,可忽略

至此完成package.json初始化配置,配置清单如下

{  
 "name": "test-pagination",  
 "version": "1.0.0",  
 "description": "",  
 "main": "index.js",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "keywords": [  
 "pagination",  
 "react"  
 ],  
 "author": "Jun",  
 "license": "ISC"  
}

接下来在package.json配置开发环境

加入React依赖包

"dependencies": {  
 "lodash": "4.16.4",  
 "react": "15.2.1",  
 "react-dom": "15.2.1"  
}

加入webpack工具

"devDependencies": {  
 "extract-text-webpack-plugin": "~0.8.2",  
 "webpack": "^1.7.3",  
 "webpack-dev-server": "^1.16.2"  
}

由于需要使用ES6语法编写,加入babel编译工具

"devDependencies": {  
 "babel-core": "^6.5.2",  
 "babel-eslint": "^4.1.8",  
 "babel-loader": "^6.2.2",  
 "babel-plugin-add-module-exports": "^0.1.2",  
 "babel-plugin-transform-runtime": "^6.15.0",  
 "babel-preset-es2015": "^6.18.0",  
 "babel-preset-es2015-ie": "6.x",  
 "babel-preset-react": "^6.3.13",  
 "babel-preset-stage-1": "^6.16.0",  
 "babel-register": "^6.18.0",  
 "extract-text-webpack-plugin": "~0.8.2",  
 "webpack": "^1.7.3",  
 "webpack-dev-server": "^1.16.2"  
}

加入代码校验工具ESLint

"devDependencies": {  
 "babel-core": "^6.5.2",  
 "babel-eslint": "^4.1.8",  
 "babel-loader": "^6.2.2",  
 "babel-plugin-add-module-exports": "^0.1.2",  
 "babel-plugin-transform-runtime": "^6.15.0",  
 "babel-preset-es2015": "^6.18.0",  
 "babel-preset-es2015-ie": "6.x",  
 "babel-preset-react": "^6.3.13",  
 "babel-preset-stage-1": "^6.16.0",  
 "babel-register": "^6.18.0",  
 "eslint": "^1.10.3",  
 "eslint-config-airbnb": "^5.0.1",  
 "eslint-loader": "^1.6.3",  
 "eslint-plugin-react": "^3.16.1",  
 "extract-text-webpack-plugin": "~0.8.2",  
 "webpack": "^1.7.3",  
 "webpack-dev-server": "^1.16.2"  
}

配置webpack build命令和webpack-dev-server开发环境,可用于实时调试和热部署项目

"scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1",  
 "dev": "webpack-dev-server --devtool eval --progress --colors --open --hot --content-base ./example",  
 "build": "webpack -p --colors"  
}

* 在本项目中会用到的webpack/webpack-dev-server命令说明

webpack

|  |  |
| --- | --- |
| Command | Description |
| webpack | 主命令：执行编译/混合/CSS样式,开发模式,代码未压缩处理,并包含webpack相关编译代码 |
| -p | 主命令：执行编译/压缩/混合/CSS样式,不包含webpack相关编译代码 |
| -w | 执行编译/混合/CSS样式,开发模式,实时监听代码变化,并进行编译/压缩/混合等一系列热部署操作 |
| -p --color | 主命令：执行编译/压缩/混合/CSS样式,不包含webpack相关编译代码,并高亮显示控制台输出结果 |
| -h | 查看更多webpack命令 |

webpack-dev-server

|  |  |
| --- | --- |
| Command | Description |
| webpack-dev-server | 主命令：启动webpack开发调试服务 |
| --devtool eval | 启用开发者模式,编译后代码包含sourcemap等信息,可用于浏览器进行调试 |
| --progress | 显示webpack building进度 |
| --colors | 高亮显示控制台输出结果 |
| --open | 浏览器自动刷新 |
| --hot | webpack服务实时监听 |
| --content-base ./example | webpack服务启动入口html文件目录设置,例如example目录 |

至此package.json配置完成,在之后的开发中如果需要其他配置和依赖包,可按照如上步骤,以下是完整package.json清单

{  
 "name": "test-pagination",  
 "version": "0.0.1",  
 "description": "react pagination",  
 "main": "index.js",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1",  
 "dev": "webpack-dev-server --devtool eval --progress --colors --open --hot --content-base ./example",  
 "build": "webpack -p --colors"  
 },  
 "keywords": [  
 "react",  
 "pagination"  
 ],  
 "author": "Jun",  
 "license": "ISC",  
 "devDependencies": {  
 "babel-core": "^6.5.2",  
 "babel-eslint": "^4.1.8",  
 "babel-loader": "^6.2.2",  
 "babel-plugin-add-module-exports": "^0.1.2",  
 "babel-plugin-transform-runtime": "^6.15.0",  
 "babel-preset-es2015": "^6.18.0",  
 "babel-preset-es2015-ie": "6.x",  
 "babel-preset-react": "^6.3.13",  
 "babel-preset-stage-1": "^6.16.0",  
 "babel-register": "^6.18.0",  
 "eslint": "^1.10.3",  
 "eslint-config-airbnb": "^5.0.1",  
 "eslint-loader": "^1.6.3",  
 "eslint-plugin-react": "^3.16.1",  
 "extract-text-webpack-plugin": "~0.8.2",  
 "webpack": "^1.7.3",  
 "webpack-dev-server": "^1.16.2"  
 },  
 "dependencies": {  
 "lodash": "4.16.4",  
 "react": "15.2.1",  
 "react-dom": "15.2.1"  
 }  
}

安装上面配置好的依赖包(这一步执行完毕,才能继续下面的操作否则ESLint等工具会提示报错)

$ npm install

配置编译工具babel

如果.babelrc文件不存在,则新建,配置清单如下

{  
 "presets": [  
 "es2015",  
 "react"  
 ],  
 "plugins": [  
 "transform-runtime"  
 ]  
}

由于并不需要让babel编译依赖包目录node\_modules,所以需要进行配置,新建.eslintignore文件,配置如下

node\_modules

接下来配置js语法校验工具,按照ECMAScript 2015标准对语法进行检验,在这里我们使用Airbnb的eclint的规则,在前面的package.json中已加入依赖包;新建.eslintrc,配置清单如下

{  
 "env": {  
 "node": true,  
 "es6": true,  
 "browser": true  
 },  
 "parser": "babel-eslint",  
 "extends": "airbnb",  
 "rules": {  
 "no-var": [  
 0  
 ],  
 "no-console": 1,  
 "no-unused-vars":1,  
 "no-param-reassign":1,  
 "react/jsx-no-bind":1  
 }  
}

### webpack配置（项目构建）

如果完成前面的项目初始化配置,接下来可以配置webpack清单,配置明细如下

引入node path模块,用于获取文件路径  
引入ExtractTextPlugin外部加载文件插件  
初始化node依赖包路径

const path = require('path');  
const ExtractTextPlugin = require('extract-text-webpack-plugin');  
const nodeModulesPath = path.resolve(\_\_dirname, 'node\_modules');

* 这里使用ECMAScript 2015不可随意修改的变量类型const,具有块级作用域的作用,可避免var声明的变量存在变量提升和随意修改的问题

在这里使用module.export输出配置

module.exports = {  
 //webpack配置清单  
}

设置devtool属性为false,在webpack打包时不生成sourcemap信息,

module.exports = {  
 devtool: false  
}

设置webpack入口文件,即编译入口文件

entry: {  
 'pagination.test': path.join(\_\_dirname, 'src', 'index.test.js'),// demo测试程序入口文件  
 pagination: path.join(\_\_dirname, 'src', 'index.js'), // 分页组件入口文件  
}

设置输出文件目录以及chunk文件

output: {  
 path: path.join(\_\_dirname, 'dist'), // 输出目录（编译生成文件目录）  
 publicPath: '',  
 filename: 'js/[name].js', // 编译生成的文件,文件名由前面入口文件配置确定  
 chunkFilename: 'js/[id].chunk.js',  
}

模块和插件配置

配置ESLint预加载,用于语法检查

module: {  
 preLoaders: [  
 {  
 // ESlint loader  
 test: /\.(js|jsx)$/,  
 loader: 'eslint-loader',  
 include: [path.resolve(\_\_dirname, 'src')],  
 exclude: [nodeModulesPath],  
 },  
 ]  
}

配置加载模块插件,在本项目中仅编译js所以仅加载js的编译工具,同时排除node依赖包的编译,且使用babel;在之后会补充css/sass模块插件

module: {  
 preLoaders: [  
 {  
 // ESlint loader  
 test: /\.(js|jsx)$/,  
 loader: 'eslint-loader',  
 include: [path.resolve(\_\_dirname, 'src')],  
 exclude: [nodeModulesPath],  
 },  
 ],  
 loaders: [  
 {   
 test: /\.js?$/,  
 exclude: /node\_modules/,  
 loader: 'babel',  
 },  
 ]  
}

为了便于更好的扩展性,且同时编译react/react-dom,生成的文件会很大,比较消耗资源,在页面中加载数MB的js文件并不理想,所以在这里进行如下配置,可以将react/react-dom通过CDN依赖等外部引入的方式加载至页面

externals: { // 指定采用外部 CDN 依赖的资源,不被webpack打包  
 react: 'React',  
 'react-dom': 'ReactDOM',  
}

webpack-dev-server也可以在这里配置,包括服务监听端口号

devServer: {  
 hot: true,  
 inline: true, // webpack-dev-server有两种模式,默认是false,即在页面中加入frame标签构建调试页面;若为true则是在完整页面中构建调试页面  
 progress: true,  
 port: '3001',  
}

加载ESLint配置文件,由于上面进行ESLint模块预加载,在这里需要加入ESLint配置文件

eslint: {  
 configFile: '.eslintrc',  
}

完整webpack清单如下

const path = require('path');  
const ExtractTextPlugin = require('extract-text-webpack-plugin');  
const nodeModulesPath = path.resolve(\_\_dirname, 'node\_modules');  
  
module.exports = {  
 devtool: false,  
 entry: {  
 'pagination.test': path.join(\_\_dirname, 'src', 'index.test.js'),  
 pagination: path.join(\_\_dirname, 'src', 'index.js'),  
 },  
  
 output: {  
 path: path.join(\_\_dirname, 'dist'),  
 publicPath: '',  
 filename: 'js/[name].js',  
 chunkFilename: 'js/[id].chunk.js',  
 },  
 module: {  
 preLoaders: [  
 {  
 // ESLint loader  
 test: /\.(js|jsx)$/,  
 loader: 'eslint-loader',  
 include: [path.resolve(\_\_dirname, 'src')],  
 exclude: [nodeModulesPath],  
 },  
 ],  
 loaders: [  
 {  
 test: /\.js?$/,  
 exclude: /node\_modules/,  
 loader: 'babel',  
 },  
 ],  
 },  
 externals: { // 指定采用外部 CDN 依赖的资源,不被webpack打包  
 react: 'React',  
 'react-dom': 'ReactDOM',  
 },  
 devServer: {  
 hot: true,  
 inline: true,  
 progress: true,  
 port: '3001',  
 },  
 eslint: {  
 configFile: '.eslintrc',  
 },  
};

配置好webpack清单,基本上可以执行webpack相关命令了

### 组件具体实现

本项目实现的分页组件由多个子组件组成,目前仅实现了基础子组件,之后会不断完善

|  |  |
| --- | --- |
| Component | Description |
| Buttons | 分页按钮,包括每页按钮,下一页按钮,多页跳转按钮 |
| Pagination | 分页组件主结构 |
| Select | 分页组件选择每页显示的数目 |

#### 在src/components目录中新建如下文件

Buttons.js // 分页按钮组件  
Pagination.js // 分页组件主结构  
Select.js // 下拉选择数目组件

#### 实现Buttons组件

引入react.js

import React from 'react';

* ECMAScript 2015标准,规定了在js文件中可以使用import/from关键字引入其他目录的js模块文件

定义Buttons类,并继承React Component父类

class Buttons extends React.Component {  
  
}

* ECMAScript 2015标准,规定了class的用法与标准

定义Buttons类的构造方法,构造函数,在创建组件的时候调用一次,用来在之后的方法中引用父类(React.Component)的this对象

class Buttons extends React.Component {  
 constructor(props) {  
 super(props);  
 }  
}

在上面的实现效果图中,分页按钮包括含有分页编号的按钮/上一页/下一页/向前几页跳转/向后几页跳转等.因此需要定义Buttons props如下默认属性

|  |  |  |  |
| --- | --- | --- | --- |
| PropTypes | Description | Type | Default |
| pageNumber | 分页编号 | number |  |
| active | 是否是当前选中的分页 | number | false |
| className | 按钮class属性 | number |  |

代码如下：

Buttons.propTypes = {  
 pageNumber: React.PropTypes.number,  
 active: React.PropTypes.bool,  
 className: React.PropTypes.string,  
};  
  
Buttons.defaultProps = {  
 active: false,  
};

实现render方法,react.js渲染组件时执行的实现方法

render() {  
  
}

初始化props对象,props中包含上面定义的属性,包括react封装好的属性

render() {  
 const props = this.props;  
}

按钮组件需要继承父组件的属性包括className集合,即多个class值,所以这里初始化父组件父组件class值

render() {  
 const props = this.props;  
 const prefix = `${props.rootClassNamePrefix}-btn`; // 继承父组件class属性前缀  
 let tempClassName = `${prefix}`;  
}

组装className属性

render() {  
 const props = this.props;  
 const prefix = `${props.rootClassNamePrefix}-btn`;  
 let tempClassName = `${prefix}`;  
 if (props.pageNumber) {  
 tempClassName += ` ${prefix}-${props.pageNumber}`;  
 }  
 if (props.active) {  
 tempClassName += ` ${prefix}-active`; // 设置active标识  
 }  
 if (props.className) {  
 tempClassName += ` ${props.className}`;  
 }  
}

返回组件标签值

render() {  
 const props = this.props;  
 const prefix = `${props.rootClassNamePrefix}-btn`;  
 let tempClassName = `${prefix}`;  
 if (props.pageNumber) {  
 tempClassName += ` ${prefix}-${props.pageNumber}`;  
 }  
 if (props.active) {  
 tempClassName += ` ${prefix}-active`;  
 }  
 if (props.className) {  
 tempClassName += ` ${props.className}`;  
 }  
  
 return (  
 <li title={props.title} // 设置html title属性  
 className={tempClassName} //   
 onClick={props.onClick}> // 对按钮设置点击事件属性  
 <a>{props.btnContent}</a>  
 </li>  
 );  
 }

至此按钮组件完成,接下来需要输出Buttons类供其他组件复用

export default Buttons;

这里使用ECMAScript 2015规范中的export,即暴露供外部调用的class/function/变量等,其他类如果需要使用,只需按如下方式,使用import/from等关键字

import Buttons from './Buttons';

在Pagination.js中引入react.js

import React from 'react';

再按照上文所说引入Buttons组件

定义父组件Pagination类,并继承React.Component

class Pagination extends React.Component {  
  
}

定义构造方法并继承React.Component的this对象

class Pagination extends React.Component {  
 constructor(props) {  
 super(props);  
 }  
}

定义Pagination props属性(对外暴露)

|  |  |  |  |
| --- | --- | --- | --- |
| PropTypes | Description | Type | Default |
| current | 当前页编号 | number |  |
| defaultCurrent | 默认当前页 | number | 1 |
| defaultPageSize | 默认分页每页显示数目 | number | 5 |
| total | 数据总数 | number | 0 |
| pageSize | 分页每页显示数目 | number |  |
| classNamePrefix | className属性前缀 | string | mc-pagination |
| onChange | 页面是否变化 | func |  |
| displayLength | 显示按钮数量 | number | 5 |
| simplePager | 是否手动跳转指定页面 | bool | false |
| pageSelect | 是否选择分页每页显示的数目 | bool | false |
| selectOptionsPageSize | 分页显示数目选项 | array | (默认值取子组件) |

代码如下

Pagination.propTypes = {  
 current: React.PropTypes.number,  
 defaultCurrent: React.PropTypes.number,  
 defaultPageSize: React.PropTypes.number,  
 total: React.PropTypes.number,  
 pageSize: React.PropTypes.number,  
 classNamePrefix: React.PropTypes.string,  
 onChange: React.PropTypes.func,  
 displayLength: React.PropTypes.number,  
 simplePager: React.PropTypes.bool,  
 pageSelect: React.PropTypes.bool,  
 selectOptionsPageSize: React.PropTypes.arrayOf(React.PropTypes.number),  
};  
  
Pagination.defaultProps = {  
 defaultCurrent: 1,  
 defaultPageSize: 5,  
 total: 0,  
 classNamePrefix: 'mc-pagination',  
 onChange: temp,  
 displayLength: 5,  
 simplePager: false,  
 pageSelect: false,  
};

#### 本项目实现的分页算法如下

在实际效果图中,分页组件由四个部分组成,如图所示

上一页/下一页按钮(蓝色框)

第一页/最后一页按钮(绿色框)

向前跳转更多页/向后跳转更多页(紫色框)

页码按钮(红色框)

* 首先定义数组容器,用于存放分页按钮
* const pageList = [];
* 第一页始终保持静态,但是当点击触发时该按钮状态变为active,即先默认初始化第一页的按钮
* pageList.push(<Buttons  
   rootClassNamePrefix={props.classNamePrefix}  
   title={1}  
   key={1}  
   onClick={this.\_handleChange.bind(this, 1)}  
   btnContent={1}  
   pageNumber={1}  
  />);
* 这里是react jsx语法的写法,配置Buttons组件即可,详细请看下面
* 最后一页即为总页数,总页数由总数目决定,算法如下
* this.props.total / pageSize

在这里是需要取整页数,且页码计算是从0开始,所以调整如下

Math.floor((this.props.total - 1) / pageSize) + 1;

* props.displayLength属性控制显示页码按钮数目,默认设置是5,即显示5个页码按钮
* 接下来需要确定如何动态控制页码按钮,如效果图所示.设定两个锚点值,左锚与右锚.其中右锚由左锚加上props.displayLength再减去1得到,如下  
  假设当前分页组件的页码状态如下
* 1 ... 6(锚) 7 8 9 10(锚) ... 200

点击第10页,如下

1 ... 10(锚) 11 12 13 14(锚) ... 200

点击第11页至第13页锚不发生改变

1 ... 10(锚) 11 12 13 14(锚) ... 200

点击向前或向后跳转回到第1页或最后一页,如下

1 2(锚) 3 4 5 6(锚) ... 200

1 ... 195(锚) 196 197 198 199(锚) 200

* 由上可知,初始化如下参数
* const anchor = this.state.leftAnchor; // 起始锚点,不可修改  
  const length = this.\_calcTotalPage(); // 总长度(总页数),不可修改  
  const dl = this.props.displayLength; // 步长(页码按钮数量),不可修改  
  let start = 2; // 起始变化值(左锚点)  
  let end = start + dl - 1; // 结束变化值(右锚点)
* 执行状态判断  
  ```  
  // n 当前页码,如果当前页码小于等于0,则赋初始值1,即回到第一页  
  if (n <= 0) {  
  n = 1;  
  }

// 如果当前页码大于等于最后一页,则赋length给n,即回到最后一页  
if (n >= length) {  
n = length;  
}

// 如果当前页码大于起始锚点(上一个状态的左锚点),则赋anchor给start,否则赋n给start,即确定左锚点  
if (n >= anchor) {  
start = anchor;  
} else {  
start = n;  
}

// 右锚点就是左锚点加步长  
end = start + dl - 1;

// 此时存在右锚点小于当前页码值的情况,因此重新确定左右锚点  
if (end <= n) {  
start = n;  
end = start + dl - 1;  
}

// 此时起始锚点值(左锚点)被改变,存在小于1的情况,  
// 因此重新确定锚点,即回到第1页  
if (start <= 1) {  
start = 2;  
end = start + dl - 1;  
}

// 如果起始锚点不存在小于1的情况,那么锚点结束位置存在大于总长度的情况,  
// 因此赋length - 1 给end,同时重新确定锚点  
if (end >= length - 1) {  
end = length - 1;  
start = end - dl + 1;  
if (start <= 1) {  
start = 2;  
}  
}

> 至此页面计算完毕,接下来初始化`Pagination`组件`props`属性和状态处理  
  
根据react核心基本原理,当状态(state)发生改变时,立刻刷新组件,重新渲染dom元素.因此上面的算法实现的分页按钮点击事件操作都会用当前页码值改变组件状态,刷新组件.因此在这里做初始化state.current属性

class Pagination extends React.Component {  
constructor(props) {  
super(props);

let current = props.defaultCurrent;  
  
this.state = {  
 current,  
}

}  
}

同时需要监听锚点状态,因此也需要初始化

class Pagination extends React.Component {  
constructor(props) {  
super(props);

const start = 2;  
const end = start + props.displayLength - 1;  
  
let current = props.defaultCurrent;  
  
this.state = {  
 current,  
 leftAnchor: start,  
 rightAnchor: end,  
};

}  
}

接下来实现钩子函数用来改变事件状态

\_handleChange(n) {  
const tempAnchor = this.\_calcPage(n); // 前面实现的\_calcPage动态页码按钮计算函数

this.setState({  
 current: tempAnchor.n,  
 \_current: tempAnchor.n,  
 leftAnchor: tempAnchor.start,  
 rightAnchor: tempAnchor.end,  
});  
  
return this.state.current;

}

至此钩子函数实现,由以上算法和原理以及钩子函数,可以依次实现下一页/上一页/向前向后跳转按钮事件,例如

// 是否有上一页  
\_hasPrev() {  
return this.state.current > 1; // this.state.current当前页(当前状态)  
}

// 是否有下一页  
\_hasNext() {  
return this.state.current < this.\_calcTotalPage(); // 由以上逻辑实现的,\_calcTotalPage计算总页数函数  
}

// 上一页  
\_prev() {  
if (this.\_hasPrev()) {  
this.\_handleChange(this.state.current - 1);  
}  
}

// 下一页  
\_next() {  
if (this.\_hasNext()) {  
this.\_handleChange(this.state.current + 1);  
}  
}

// 向前/向后跳转displayLength长度的页面  
\_leftMore() {  
return this.\_handleChange((this.state.current - this.props.displayLength) <= 0 ?  
1 : (this.state.current - this.props.displayLength));  
}

\_rightMore() {  
const totalPage = this.\_calcTotalPage();  
return this.\_handleChange((this.state.current + this.props.displayLength) >= totalPage ?  
totalPage : (this.state.current + this.props.displayLength));  
}

至此分页逻辑基本实现,但是需要做调整  
  
\* 方法绑定父类`this`对象

class Pagination extends React.Component {  
constructor(props) {  
super(props);

const start = 2;  
const end = start + props.displayLength - 1;  
  
let current = props.defaultCurrent;  
  
this.state = {  
 current,  
 leftAnchor: start,  
 rightAnchor: end,  
};  
  
[  
 'render',  
 '\_handleChange',  
 '\_isValid',  
 '\_leftMore',  
 '\_rightMore',  
 '\_hasPrev',  
 '\_hasNext',  
 '\_prev',  
 '\_next',  
].forEach((method) => this[method] = this[method].bind(this));

}  
}

\* 实现开放API接口  
  
到这里,`Pagination`组件暂时仅仅只能被react-dom渲染至页面,不能当作子组件复用,例如  
  
在`index.test.js`中编写如下代码,以渲染组件

import React from 'react';  
import ReactDom from 'react-dom';  
import Pagination from './components/Pagination';

ReactDom.render(, document.getElementById('example'));

在`example/index.html`中引用

> 本实例按照之前配置的webpack-dev-server启动,在浏览器中输入如下地址,即可查看demo和调试

<http://localhost:3001/>

如上所示,即使配置`current`也不起任何作用,无法满足复用,因此需要使用`react`组件生命周期`API`  
  
`props`是父组件传递给子组件的.父组件发生`render`的时候子组件就会调用`componentWillReceiveProps`(不管`props`有没有更新,也不管父子组件之间有没有数据交换)

componentWillReceiveProps(nextProps) {

}

在`componentWillReceiveProps`方法内处理当前页状态和锚点状态变化

componentWillReceiveProps(nextProps) {

// 如果设置current属性,则先做页面分配计算  
const n = nextProps.current;  
const tempAnchor = this.\_calcPage(n);

if ('current' in nextProps) { // 改变页面状态  
this.setState({  
current: tempAnchor.n,  
\_current: tempAnchor.n,  
leftAnchor: tempAnchor.start,  
rightAnchor: tempAnchor.end,  
});  
}  
}

同时需要在构造函数中做初始化处理

class Pagination extends React.Component {  
constructor(props) {  
super(props);

// ...  
  
let current = props.defaultCurrent;  
  
if ('current' in props) {  
 current = props.current;  
}  
  
// ...

}  
}

需要补充页面校验函数

\_isValid(num) {  
return typeof num === 'number' && num >= 1 && num !== this.state.current;  
}

改进钩子函数`\_handleChange`

\_handleChange(n) {  
const tempAnchor = this.\_calcPage(n);  
if (this.\_isValid(n)) {  
if (!('current' in this.props)) {  
this.setState({  
current: tempAnchor.n,  
\_current: tempAnchor.n,  
leftAnchor: tempAnchor.start,  
rightAnchor: tempAnchor.end,  
});  
}

const pageSize = this.state.pageSize;  
this.props.onChange(n, pageSize);  
  
return n;

}

return this.state.current;  
}

实现`pageSize`选择组件,同上`pageSize`改变引发状态改变,即重新渲染组件,所以做如下设置

//构造函数初始化pageSize  
class Pagination extends React.Component {  
constructor(props) {  
super(props);

// ...  
  
let pageSize = props.defaultPageSize;  
if ('pageSize' in props) {  
 pageSize = props.pageSize;  
}  
  
this.state = {  
  
 // ...  
  
 pageSize,  
  
 // ...  
  
};  
  
// ...

}  
}

同上在生命周期函数中需要对`pageSize`状态发生变化做处理,重新计算页面分配,改变当前页面状态

componentWillReceiveProps(nextProps) {

// ...

if ('pageSize' in nextProps) {  
const newState = {};  
let current = this.state.current;  
const newCurrent = this.\_calcTotalPage(nextProps.pageSize);  
current = current > newCurrent ? newCurrent : current;  
const tempAnchor2 = this.\_calcPage(current);  
if (!('current' in nextProps)) {  
newState.current = tempAnchor2.n;  
newState.\_current = tempAnchor2.n;  
newState.leftAnchor = tempAnchor2.start;  
newState.rightAnchor = tempAnchor2.end;  
}  
newState.pageSize = nextProps.pageSize;  
this.setState(newState);  
}  
}

在Select.js中引入react

import React from 'react';

定义`Select`类,继承`React.Component`,并定义构造函数继承父类`this`对象

class Select extends React.Component {  
constructor(props) {  
super(props);  
}  
}

定义对外暴露属性,并设置默认值  
  
| PropTypes | Description |Type | Default |  
|------------------|------------------------------------|-------------------|----------------|  
| pageSize | 初始每页显示数目 | number | |  
| changeSize | select钩子函数用于pageSize状态改变 | func | |  
| selectOptionsPageSize | 每页显示数目选项 | array(number) | [10, 20, 30, 40, 50] |  
  
  
代码如下

Select.propTypes = {  
pageSize: React.PropTypes.number,  
changeSize: React.PropTypes.func,  
selectOptionsPageSize: React.PropTypes.arrayOf(React.PropTypes.number),  
};

Select.defaultProps = {  
selectOptionsPageSize: [10, 20, 30, 40, 50],  
};

实现下拉框选择钩子函数监听选择事件改变`Pagination` `pageSize`状态

\_changeSize(event) {  
const value = event.target.value;  
this.props.changeSize(Number(value));  
}

> 这里由父组件传递`changeSize`方法  
  
实现render方法,返回并输出Select组件

render() {  
const props = this.props;  
const pageSize = props.pageSize || props.selectOptionsPageSize[0]; //设置select初始默认值  
const options = props.selectOptionsPageSize.map((o, i) => (  
{o}  
)); //根据selectOptionsPageSize组装options  
return (  
  
{options}  
  
);  
}

// ...

export default Select;

为实现方法绑定`this`对象

[  
'render',  
'\_changeSize',].forEach((method) => this[method] = this[method].bind(this));

> 至此`Select`组件实现  
  
通过一系列调整,接下来可以实现`Select`组件的`\_changePageSize`方法

\_changePageSize(size) {  
let current = this.state.current;  
const newCurrent = this.\_calcTotalPage(size);  
current = current > newCurrent ? newCurrent : current;  
const tempAnchor = this.\_calcPage(current);  
if (typeof size === 'number') {  
if (!('pageSize' in this.props)) {  
this.setState({  
pageSize: size,  
});  
}

if (!('current' in this.props)) {  
 this.setState({  
 current: tempAnchor.n,  
 \_current: tempAnchor.n,  
 leftAnchor: tempAnchor.start,  
 rightAnchor: tempAnchor.end,  
 });  
}

}

return size;  
}

为`\_changePageSize`方法绑定`this`对象

class Pagination extends React.Component {  
constructor(props) {  
super(props);  
[  
// ...  
'\_changePageSize',  
].forEach((method) => this[method] = this[method].bind(this));  
}  
}

在`Pagination`组件中引入`Select`组件

import Select from './Select';

实现`pageSelect`,即当`pageSelect`为`true`时显示分页展示数量选择框,如图所示  
  
<img src="doc/img/desc2.jpg">  
  
代码如下

render() {

// ...

let pageSelect; // Select 组件

// ...

if (props.pageSelect) {  
pageSelect = (  
  
);  
}

// ...

}

通过钩子函数实现手动输入指定页面并按回车键跳转至指定页面

\_handleKeyEnter(event) {  
const value = event.target.value;  
let tempValue;  
if (isNaN(Number(value))) {  
tempValue = this.state.current;  
} else {  
tempValue = Number(value);  
}  
if (event.keyCode === 13) {  
this.\_handleChange(tempValue);  
}  
}

为`\_handleKeyEnter`函数绑定`this`对象

class Pagination extends React.Component {  
constructor(props) {  
super(props);  
[  
// ...  
'\_handleKeyEnter',  
].forEach((method) => this[method] = this[method].bind(this));  
}  
}

在`render`方法中添加手动输入跳转指定页面的`input`组件,且当`simplePager`属性为`true`时做渲染

render() {

// ...

let simplePager; // input手动输入框组件

// ...

if (props.simplePager) {  
simplePager = (跳至  
页  
);  
}  
}

> 至此,组件基本逻辑改进完成,接下来时组装和最后的输出组件  
  
组装分页页码按钮/下一页/上一页/向前跳转/向后跳转按钮

render() {  
const props = this.props;  
const pageList = [];  
const totalPage = this.\_calcTotalPage(); //获取总页数  
const { current, pageSize } = this.state;

// ...

// 第一页按钮,并设置onClick属性和钩子事件函数  
pageList.push();

// 向前跳转按钮,且当左锚点大于2(第2页)时,才渲染  
if (this.state.leftAnchor > 2) {  
pageList.push();  
}

// 根据锚点循环组装页码按钮  
for (let i = this.state.leftAnchor; i <= this.state.rightAnchor; i++) {  
const isActive = this.state.current === i;  
pageList.push(  
);  
}

//向后跳转,且当右锚点小于(totalPage - 1)(最后一页减一)时,才渲染  
if (this.state.rightAnchor < (totalPage - 1)) {  
pageList.push();  
}

// 最后一页按钮  
pageList.push();

}

输出组件,这里用`svg`生成上一页下一页按钮样式

render() {

// ...

return (  
// 设置组件class属性

// 上一页按钮  
 <Buttons  
 rootClassNamePrefix={props.classNamePrefix}  
 title={`上一页`}  
 onClick={this.\_prev}  
  
 // 在这里使用svg,包括svg的样式  
 btnContent={<svg viewBox={`0 0 24 24`}  
 style={{  
 display: 'inline-block',  
 color: (this.\_hasPrev() ? 'rgba(0, 0, 0, 0.870588)' : '#ccc',  
 fill: 'currentcolor',  
 height: '24px',  
 width: '24px',  
 userSelect: 'none',  
 transition: 'all 450ms cubic-bezier(0.23, 1, 0.32, 1) 0ms',  
 }}  
 >  
 <path  
 d={'M15.41 7.41L14 6l-6 6 6 6 1.41-1.41L10.83 12z'}  
 ></path>  
 </svg>}  
 className={`${props.classNamePrefix}-btn-prev ${this.\_hasPrev() ? '' :  
 `${props.classNamePrefix}-btn-disabled`}`}  
 />  
 {pageList}  
  
 // 下一页按钮  
 <Buttons  
 rootClassNamePrefix={props.classNamePrefix}  
 title={`下一页`}  
 onClick={this.\_next}  
 btnContent={<svg  
 viewBox={`0 0 24 24`}  
 style={{  
 display: 'inline-block',  
 color: (this.\_hasNext() ? 'rgba(0, 0, 0, 0.870588)' : '#ccc'),  
 fill: 'currentcolor',  
 height: '24px',  
 width: '24px',  
 userSelect: 'none',  
 transition: 'all 450ms cubic-bezier(0.23, 1, 0.32, 1) 0ms',  
 }}  
 >  
 <path  
 d={`M10 6L8.59 7.41 13.17 12l-4.58 4.59L10 18l6-6z`}  
 >  
 </path></svg>}  
 className={`${props.classNamePrefix}-btn-next ${this.\_hasNext() ? '' :  
 `${props.classNamePrefix}-next-btn ${props.classNamePrefix}-btn-disabled`}`}  
 />  
  
 // 页面选择组件  
 {pageSelect}  
  
 // 手动输入页面组件  
 {simplePager}  
</ul>

);  
}

// ...

// 输出Pagination组件  
export default Pagination;

> 到这里分页组件所有逻辑均已实现,接下来是用法以及开发环境的调试  
  
#### 用法  
  
> 为了便于组件的复用,且`Pagination`组件使用了多个子组件,因此对`Pagination`组件做最终出口文件处理  
  
在`src/index.js`中做如下处理

import Pagination from './components/Pagination';

export default {  
Pagination,  
};

接下来用webpack编译出可以供外部使用的完整`Pagination.js`组件  
  
切换至项目目录下,执行如下命令

$ npm run build

或者直接执行`webpack`命令

$ webpack

webpack编译过程如下  
  
<img src="doc/img/webpack-build.jpg">  
  
> 出现`warning`是因为`ESLint`在校验过程中发现存在符合规则但可以忽略的警告,可以在`.eslintrc`中编写过滤规则,过滤不需要的规则,关于ESLint规则,下面会作介绍.  
  
可以使用如下命令对`Pagination.js`进行压缩处理

$ webpack -p

最终生成的文件路径(可在webpack.config.js清单中修改)如下

dist\js\

在其他页面中使用`Pagination`组件,并进行开发调试  
  
如果按照上面的步骤配置了`webpack-dev-server`,那么即可执行如下命令启动webpack-dev-server,否则请按照上面的步骤进行配置

$ npm run dev

启动成功后浏览器与`webpack-dev-server`服务建立实时通信,`webpack-dev-server`实时监听项目js文件变动,并进行编译/压缩/混合等等一系列操作,完成操作后进行热部署,通知浏览器自动刷新页面,即可进行实时调试.打开`chrome`浏览器的调试工具可以看到控制台实时反馈`webpack-dev-server`服务操作过程,如图所示  
  
<img src="doc/img/chrome-debug.jpg">  
  
`Pagination`组件在外部引用中的用法  
  
在`index.test.js`中按照如下方式编写

import React from 'react'; // 引入react,用于创建Test类(组件)  
import ReactDom from 'react-dom'; // 引入react-dom,用于渲染组件  
import Pagination from './components/Pagination'; // 引入Pagination组件(尚未编译)

class Test extends React.Component {

//初始化构造函数  
constructor(props) {  
super(props);

// 初始化状态  
this.state = {  
 current: 2,  
};  
  
// 为方法绑定this对象  
[  
 'render',  
 'onChange',  
].forEach((method) => this[method] = this[method].bind(this));

}

// 设置onChange方法用于改变子组件当前页面状态  
onChange(page) {  
this.setState({  
current: page,  
});  
}

// 输出组件,并填写配置信息  
render() {  
return ();  
}  
}

ReactDom.render( // 渲染组件  
,  
document.getElementById('example') // 通过ID属性获取html中的存放组件的容器  
);

接下来需要编写测试用的入口html  
  
在`example/index.html`编写如下代码

> 请注意,此时是运行在服务端的开发模式资源文件(pagination.test.js)存在于服务端缓存中需要使用服务端动态地址(url)  
  
运行效果图如下  
  
<img src="doc/img/chrome-debug2.jpg">  
  
#### 关于ESLint  
  
`ESLint` 是一个开源的 `JavaScript` 代码检查工具.代码检查是一种静态的分析,常用于寻找有问题的模式或者代码,并且不依赖于具体的编码风格。对大多数编程语言来说都会有代码检查,一般来说编译程序会内置检查工具。  
  
`JavaScript` 是一个动态的弱类型语言,在开发中比较容易出错。且是实时编译,为了改变开发方式,提升开发效率,需要一种语法检查工具.  
  
`ESLint` 为了让开发者可以创建自己的检测规则,所有规则都被设计成可插入的。`ESLint` 的默认规则与其他的插件无太大区别,规则本身和测试可以依赖于同样的模式。在项目中可以使用`ESLint` 内置规则,也可以在使用过程中自定义规则。  
  
`ESLint`是基于`Node.js`,使用node.js安装部署。  
  
所有规则都是可拔插的  
  
\* 内置规则和自定义规则共用一套规则 API  
\* 内置的格式化方法和自定义的格式化方法共用一套格式化 API  
\* 额外的规则和格式化方法能够在运行时指定  
\* 规则和对应的格式化方法并不强制捆绑使用  
  
每条规则:  
  
\* 各自独立  
\* 可以开启或关闭  
\* 可以将结果设置成警告或者错误  
\* ESLint 规则可根据需要自由定制  
\* 所有内置规则都是泛化的  
\* 在本项目中使用Airbnb规则  
  
关于`eslint-config-airbnb`规则  
  
`eslint-config-airbnb`规则是Airbnb公司开源的基于`ESLint`的规则,使用起来较为方便,且可以更好的保持代码风格一致性,可读性,可维护性  
  
#### 关于ECMAScript  
  
ECMAScript 的第六版修订,于 2015 年完成标准化.这个标准被部分实现于大部分现代浏览器.  
  
在本项目使用了ECMAScript 2015规范中的语法,并使用babel工具进行编译,以便于在大部分浏览器中能够稳定运行.  
  
\* 关于babel  
  
Babel是一个广泛使用的转码器,可以将ECMAScript 2015(ECMAScript 6)代码转为ECMAScript 5代码,从而在现有环境执行.  
  
\* 在本项目中用到的一些语法特性  
  
>`const\let` 用法  
  
> 不同于`var`,`const/let`更加严格,具有块级作用域性质;可以避免`var`变量提升,局部变量不可控等一系列问题,例如  
 ```  
 // ES6  
 let sum=0;  
 for(let i=0; i<5;i++){  
 sum+=i;  
 }  
 console.log('sum = '+sum);  
 console.log('i = '+i)  
  
 // error  
  
 // Uncaught SyntaxError: Identifier 'sum' has already been declared  
 ```  
  
 ```  
 // ES6  
 const sum = 1;  
 console.log(sum);  
  
 // error  
 // Uncaught SyntaxError: Identifier 'sum' has already been declared  
 ```  
> babel编译后   
 ```  
 // ES5  
 var sum=0;  
 for(var i=0; i<5;i++){  
 sum+=i;  
 }  
 console.log('sum = '+sum);  
 console.log('i = '+i)  
  
 // sum = 10  
 // i = 5  
 ```  
  
> 箭头函数用法  
 ```  
 // ES6  
 let f = v => v;  
 ```  
> babel编译后   
 ```  
 "use strict";  
  
 var f = function f(v) {  
 return v;  
 };  
 ```  
  
> class\extends\constructor\super用法  
   
> JavaScript语言的传统方法是通过构造函数,定义并生成新对象,例如  
 ```  
 function test(value){  
 this.value = value  
 }  
 test.prototype.testFunc = function () {  
 console.log(this.value);  
 }  
 var t = new test(1);  
 ```  
> 使用class  
 ```  
 class test{  
 constructor(value){  
 this.value = value  
 }  
 testFunc = function () {  
 console.log(this.value);  
 }  
 }  
 ```   
> `class`之间可以通过`extends`关键字实现继承,这比ES5的通过修改原型链实现继承,要清晰和方便很多,例如在本项目中继承react的Component类  
 ```  
 class Pagination extends React.Component {  
   
 }  
 ```   
> 用`super`继承父类`this`,但是创造父类的实例对象`this`(所以必须先调用`super`方法),然后再用子类构造函数修改`this`  
  
> import/export模块功能用法  
  
> 模块功能主要由两个命令构成:export和import。export命令用于规定模块的对外接口,import命令用于输入他模块提供的功能  
  
> 一个模块就是一个独立的文件,该文件内部的所有变量,外部无法获取.如果希望外部能够读取模块内部的某个变量/方法/类等,就必须使用export关键字输出  
 ```  
 var firstName = 'Michael';  
 var lastName = 'Jackson';  
 var year = 1958;  
  
 export {firstName, lastName, year};  
 ```  
> 指定模块输出`export default`   
  
> 使用export命令定义了模块的对外接口以后,其他 JS 文件就可以通过import命令加载这个模块.  
 ```  
 import { stat, exists, readFile } from 'fs'; // 引入node 内置fs模块  
 ```  
  
> import命令接受一对大括号,里面指定要从其他模块导入的变量名.大括号里面的变量名,必须与被导入模块对外接口的名称相同  
  
> 如果想为输入的变量重新取一个名字,import命令要使用as关键字,将输入的变量重命名  
 ```  
 import { rf as readFile } from 'fs';  
 ```  
  
#### 关于React  
React是一个为数据提供渲染, HTML的视图的开源 JavaScript 库.React视图通常采用包含以自定义HTML 标记规定的其他组件的组件渲染.React 为开发者提供了一种子组件不能直接影响外层组件 ("data flows down") 的模型,数据改变时对HTML文档的有效更新,和现代单页应用中组件之间干净的分离   
  
React提出了虚拟DOM的概念(virtual DOM)即React组件并不是真实的DOM节点，而是存在于内存之中的一种数据结构.只有当它插入文档以后,才会变成真实的DOM.根据React的设计,所有的DOM变动,都先在虚拟DOM上发生,然后再将实际发生变动的部分,反映在真实DOM上,这种算法叫做DOM diff,它可以极大提高网页的性能表现  
  
在本项目中的用法  
  
\* HTML模板  
  
使用React定义的JSX语法实现HTML模板,例如

class Test extends React.Component {  
render() {  
return (){  
  
Hello world!  
  
}  
}  
}

\* PropTypes用法  
组件的属性可以接受任意值,字符串\对象\函数等等都可以.有时,我们需要一种机制,验证别人使用组件时,提供的参数是否符合要求.  
组件类的PropTypes属性,就是用来验证组件实例的属性是否符合要求

Test.propTypes = {  
testA: React.PropTypes.number,  
testB: React.PropTypes.string,  
testC: React.PropTypes.bool,  
};

> 请注意在这里需要在Test类外部定义  
  
\* this.props用法  
React组件基本属性可以获取html标签的所有属性,同时可以用于获取组件的自定义PropTypes属性  
  
\* this.state状态用法  
组件免不了要与用户互动，React 的一大创新，就是将组件看成是一个状态机，一开始有一个初始状态，然后用户互动，导致状态变化

class Test extends React.Component {  
constructor(props) {  
super(props);

this.state = {  
 testA:1  
}  
  
handleClick.bind(this);

}  
handleClick: function(event) {  
this.setState({testA: 2});  
},  
render: function() {  
return (  
  
Hello world!{this.state.testA}  
  
  
Click here.  
  
);  
}  
});

由于this.props 和 this.state 都用于描述组件的特性,可能会产生混淆.一个简单的区分方法是,this.props 表示那些一旦定义,就不再改变的特性,而 this.state 是会随着用户互动而产生变化的特性.  
  
## 总结  
  
本文主要介绍了分页组件的实现(开发过程),包括开发环境搭建,开发工具的配置和使用,具体实现过程以及简单的用法介绍等.  
  
本项目主要用于学习和研究,在整个过程中收获颇丰,从基础知识到功能设计与完善以及最后的文档编写都是反复巩固和学习的过程,尤其是对Javascript语法(ECMAScript 2015)更加熟练一些,同时对React的理解也更进一步;在设计思想上更加感觉站在了一个全新的角度.  
  
当然依然有很多很多不足,比如说算法还是需要改进,从体验的角度来说,多少有些瑕疵;代码结构和习惯还是需要改进;对react的设计思想多少还是需要更加一步理解,不能单纯的站在传统dom结构的操作的思维方式  
  
下面是需要待完善的功能点  
  
\* 样式：接下来需要加入sass更加动态灵活高效控制组件css样式  
\* 实现更加完整的分页功能组件,包括引入fetch这种新的ajax模式  
\* 增加更多配置选项,增强复用性和扩展性  
  
## 开发工具及环境

IDE：Visual Studio Code  
plugin：ESLint/Webpack/Babel  
node.js：6.x +  
browser: chrome v50+  
```

## 参考

* [React英文官网](https://facebook.github.io/react/)
* [React中文官网](http://reactjs.cn/react/docs/getting-started-zh-CN.html)
* [React Github](https://github.com/facebook/react)
* [ECMAScript 6 入门](http://es6.ruanyifeng.com/)
* [ECMAScript 2015](http://www.ecma-international.org/ecma-262/6.0/)
* [Babel](https://babeljs.io)
* [Babel 中文](http://babeljs.cn/)
* [Babel 在线工具](https://babeljs.io/repl/)
* [ESLint](http://eslint.org/)
* [ESLint Github](https://github.com/eslint/eslint)
* [ESLint 中文](http://eslint.cn)
* [Webpack](https://webpack.github.io/)
* [Webpack Github](https://github.com/webpack/webpack)
* [CNode 社区](https://cnodejs.org)
* [react-component](https://github.com/react-component)
* [ant design](https://ant.design/)
* [airbnb javascript](https://github.com/airbnb/javascript)