Demo地址:  [https://github.com/huyongdi/hyd-ui]( https://github.com/huyongdi/hyd-ui)

### yarn init -y

通过默认设置新建出package.json文件

{

    "name": "ty-ui",

    "version": "1.0.0",

    "main": "index.js",

    "repository": "http://10.20.8.45/huyongdi/ty-ui.git",

    "author": "huyongdi <huyongdi@skycomm.com.cn>",

    "license": "MIT"

}

### 安装storybook 方便进行插件预览

npx - p @storybook/cli sb init  --type webpack\_react

会产生3个目录：

.storebook: storebook 的一些配置

Stories: 项目静态文档,负责 demo 演示

node\_modules: 包文件

运行yarn storybook 能在6006端口打开storybook默认界面，

运行yarn build-storybook 能在根目录下构建一个目录storybook-static （预览页面dist）

参考：<https://storybook.js.org/docs/guides/guide-react/>

### 新建测试组件

根目录下新建components目录（参考antd, 放置所有组件）

①.components 下新建button组件目录：button

button目录下建index.js

import React, { Component } from 'react';

import PropTypes from 'prop-types';

import './style/index.less';

export default class Button extends Component {

    render() {

        return <button className="btn">{this.props.text}</button>

    }

}

Button.propTypes = {

    text: PropTypes.any

}

button目录下建style目录，用来写样式

style目录下新建css.js (import './index.css')

style目录下新建index.js (import './index.less')

style目录下新建index.less (.btn { color : red } )

②.复制button目录成一个button1

修改index.js ( Button 改为Button1 className="btn1" )

修改index.less .btn1 { color : blue}

③.components 下新建index.js

export { default as Button } from "./button";

export { default as Button1 } from "./button1";

### 安装依赖

* @babel/preset-env：babel-preset-env 功能类似 babel-preset-latest，优点是它会根据目标环境选择不支持的新特性来转译。[babel-preset-latest](http://babeljs.io/docs/plugins/preset-latest/)： 支持现有所有ECMAScript版本的新特性，包括处于stage 4里的特性
* @babel/cli：babel自带的cli命令
* @babel/preset-react：适应react语法
* @babel/plugin-proposal-object-rest-spread ：让对象能使用spread操作符

var obj1 = {

name: "james",

age: 27

}

var obj2 = {...obj1};

* cross-env：使指令跨平台运行
* @babel/plugin-proposal-class-properties：解析类的属性
* webpack-cli : webpack4之后必须，不再自带
* uglifyjs-webpack-plugin ：压缩JS
* optimize-css-assets-webpack-plugin:压缩CSS
* cssnano：PostCSS的CSS优化和分解插件。cssnano采用格式很好的CSS，并通过许多优化，以确保最终的生产环境尽可能小
* mini-css-extract-plugin：webpack4新加 将CSS提取为独立的文件的插件，对每个包含css的js文件都会创建一个CSS文件，支持按需加载css和sourceMap
* progress-bar-webpack-plugin： 编译进度条插件
* webpack-bundle-analyzer ：打包体积展示
* autoprefixer：通过浏览器自动加CSS前缀

### 5. 新建.babelrc.js文件

'use strict';

const outputModule = process.env.OUTPUT\_MODULE;

const options = {

    presets: [

        ['@babel/preset-env', { modules: outputModule || false }], '@babel/preset-react'

    ],

    plugins: [

        '@babel/proposal-object-rest-spread', '@babel/proposal-class-properties'

    ],

};

module.exports = options;

### 6. 新建postcss.config.js

module.exports = {

    plugins: [

        require("autoprefixer")({

            browsers: [

                "last 2 versions",

                "Firefox ESR",

                "not ie < 9",

                "ff >= 30",

                "chrome >= 34",

                "safari >= 6",

                "opera >= 12.1"

            ]

        })

    ]

};

### 7. 新建webpack.config.js文件

/\*\*

\* @name UMD 模块 打包

\* @description 输出目录 [dist]

\* @description 文件名 [hyd-ui]

\* CMD Node.js 环境

\* AMD 浏览器环境

\* UMD 两种环境都可以执行

\*/

const fs = require("fs");

const path = require("path");

const webpack = require("webpack");

const UglifyJsPlugin = require("uglifyjs-webpack-plugin");  //压缩

const OptimizeCSSAssetsPlugin = require("optimize-css-assets-webpack-plugin"); //压缩单独的CSS

const MiniCssExtractPlugin = require('mini-css-extract-plugin'); //webpack4新出，单独打包CSS

const ProgressBarPlugin = require('progress-bar-webpack-plugin'); //编译进度条插件

const { BundleAnalyzerPlugin } = require("webpack-bundle-analyzer")  //打包体积展示

const { name } = require("./package.json");

const config = {

    mode: "production",

    entry: {

        [name]: ["./components/index.js"]

    },

    //umd 模式打包

    output: {

        library: name,

        libraryTarget: "umd",

        umdNamedDefine: true, // 是否将模块名称作为 AMD 输出的命名空间

        path: path.join(process.cwd(), "dist"), //当前Node.js进程执行时的工作目录与dist连接

        filename: "[name].min.js"

    },

    //react 和 react-dom 不打包

    externals: {

        react: {

            root: "React",

            commonjs2: "react",

            commonjs: "react",

            amd: "react"

        },

        "react-dom": {

            root: "ReactDOM",

            commonjs2: "react-dom",

            commonjs: "react-dom",

            amd: "react-dom"

        }

    },

    resolve: {

        enforceExtension: false,  //导入语句去掉文件后缀

        extensions: [".js", ".jsx", ".json", ".less", ".css"]

    },

    module: {

        rules: [

            {

                test: /\.js[x]?$/,

                use: [

                    {

                        loader: "babel-loader"

                    }

                ],

                exclude: "/node\_modules/",

                include: [path.resolve("components")]  //当前工作目录的绝对路径

            },

            {

                test: /\.(le|c)ss$/,

                use: [

                    MiniCssExtractPlugin.loader,

                    "css-loader",

                    { loader: "postcss-loader", options: { sourceMap: false } },

                    {

                        loader: "less-loader",

                        options: {

                            sourceMap: false

                        }

                    }

                ]

            },

            {

                test: /\.(jpg|jpeg|png|gif|cur|ico)$/,

                use: [

                    {

                        loader: "file-loader",

                        options: {

                            name: "images/[name][hash:8].[ext]" //遇到图片  生成一个images文件夹  名字.后缀的图片

                        }

                    }

                ]

            }

        ]

    },

    optimization: {

        minimizer: [

            new UglifyJsPlugin({

                cache: true,  //是否启用文件缓存，默认缓存在node\_modules/.cache/uglifyjs-webpack-plugin.目录

                parallel: true, //使用多进程并行运行来提高构建速度

                uglifyOptions: {

                    compress: {

                        // warnings: false,

                        drop\_debugger: true,

                        drop\_console: false

                    },

                }

            }),

            new OptimizeCSSAssetsPlugin({

                // 压缩css  与 ExtractTextPlugin 配合使用

                cssProcessor: require("cssnano"),

                cssProcessorOptions: { discardComments: { removeAll: true } }, // 移除所有注释

                canPrint: true //是否向控制台打印消息

            })

        ],

        // noEmitOnErrors: true //取代 new webpack.NoEmitOnErrorsPlugin()，编译错误时不打印输出资源。

    },

    plugins: [

        new ProgressBarPlugin(),

        new MiniCssExtractPlugin({

            filename: "[name].min.css"

        }),

        //moment 2.18 会将所有本地化内容和核心功能一起打包。你可使用 IgnorePlugin 在打包时忽略本地化内容:

        new webpack.IgnorePlugin(/^\.\/locale$/, /moment$/),

        // new BundleAnalyzerPlugin(),

    ]

};

module.exports = config;

### 8. 利用gulp打包less文件

gulp：

* gulp-concat:合并文件
* gulp-less：less文件处理
* gulp-autoprefixer：浏览器兼容
* gulp-cssnano：压缩CSS
* gulp-filesize：显示文件大小
* 注意gulp3和4的区别：**gulp.series**和**gulp.paralle**l

<https://blog.csdn.net/qq_31975963/article/details/83034450>

* less less-loader

新建gulp文件

/\*\*

\* @name gulpfile.js

\* @description 打包项目css依赖

\*/

const path = require("path");

const gulp = require("gulp");

const concat = require("gulp-concat"); //合并文件，减少网络请求。

const less = require("gulp-less");

const autoprefixer = require("gulp-autoprefixer");

const cssnano = require("gulp-cssnano"); //压缩

const size = require("gulp-filesize"); //显示文件大小

const { name } = require("./package.json");

const browserList = [

    "last 2 versions",

    "Firefox ESR",

    "not ie < 9"

];

const DIR = {  //生成绝对路径

    less: path.resolve(\_\_dirname, "components/\*\*/\*.less"),

    buildSrc: [

        path.resolve(\_\_dirname, "components/\*\*/index.less")

    ],

    lib: path.resolve(\_\_dirname, "lib"),

    es: path.resolve(\_\_dirname, "es"),

    dist: path.resolve(\_\_dirname, "dist")

};

gulp.task("copyLess", () => {

    return gulp

        .src(DIR.less)

        .pipe(gulp.dest(DIR.lib))

        .pipe(gulp.dest(DIR.es));

});

gulp.task("copyCss", () => {

    return gulp

        .src(DIR.buildSrc)

        //编译.less文件为.css文件

        .pipe(

            less({

                outputStyle: "compressed"

            })

        )

        .pipe(autoprefixer({ browsers: browserList }))

        .pipe(size())

        .pipe(cssnano())

        .pipe(gulp.dest(DIR.lib))

        .pipe(gulp.dest(DIR.es));

});

gulp.task("dist", () => {

    return gulp

        .src(DIR.buildSrc)

        .pipe(

            less({

                outputStyle: "compressed"

            })

        )

        .pipe(autoprefixer({ browsers: browserList }))

        .pipe(concat(`${name}.css`))

        .pipe(cssnano())

        .pipe(concat(`${name}.min.css`))

        .pipe(size())

        .pipe(gulp.dest(DIR.dist))

});

gulp.task('default', gulp.series('copyLess', 'copyCss', 'dist', function (done) {

    done()

}))

### 10. storybook预览组件

①. storybook目录下编辑webpack.config.js（使sk能调用jsx文件，并顺利引入less）

const path = require("path")

const webpack = require('webpack')

module.exports = {

    mode: "production",

    module: {

        rules: [

            {

                test: /\.less$/,

                use: [

                    "style-loader",

                    "css-loader",

                    "postcss-loader",

                    "less-loader"

                ],

            }

        ],

    },

    resolve: {

        extensions: [".js", ".jsx", ".js", ".json"]

    },

    plugins: [

        new webpack.ContextReplacementPlugin(/moment[/\\]locale$/, /zh-cn|en-gb/),

    ]

}

②. stories目录下编辑index.stories.js

import React from 'react';

import { storiesOf } from '@storybook/react';

// import { action } from '@storybook/addon-actions';

// import { linkTo } from '@storybook/addon-links';

// import { Welcome } from '@storybook/react/demo';

import Button from "../components/button";

import Button1 from "../components/button1";

storiesOf('Button', module).add('演示按钮', () => <><Button text={1234}></Button></>);

storiesOf('Button1', module).add('演示按钮', () => <Button1 text={1234}></Button1>);

到这里，运行yarn storybook就能看见2个按钮的展示了

③. 修改storybook预览页面的标题，超链接等：

yarn add @storybook/addon-options -D

.storybook目录下的config.js修改代码

import { configure, addDecorator } from '@storybook/react';

import { withOptions } from '@storybook/addon-options';

import { name, repository, version } from "../package.json"

// automatically import all files ending in \*.stories.js

const req = require.context('../stories', true, /\.stories\.js$/);

function loadStories() {

    req.keys().forEach(filename => req(filename));

}

addDecorator(withOptions({

    name: `${name} v${version}`,

    url: repository,

    sidebarAnimations: true,

}))

configure(loadStories, module);

.storybook目录下新建文件 addons.js

import '@storybook/addon-options/register';

这样之后，左上角的名字就改了

### 配置package.json文件

"description": "react+webpack+gulp+less+storybook 从零搭建实现了组件库的按需加载",

"keywords": [

        "ty-ui",

        "react",

        "gulp",

        "webpack",

        "less",

        "storybook",

        "react组件库"

],

"homepage": "http://www.huyongdi.com",

"bugs": {

    "url": "https://github.com/huyongdi/hyd-ui",

    "email": "354224080@qq.com"

},

"main": "lib/index.js",

"module": "es/index.js",

"files": [

    "lib",

    "es",

    "dist"

],

"sideEffects": [

    "dist/\*",

    "es/\*\*/style/\*",

    "lib/\*\*/style/\*",

    "\*.less"

],

"scripts": {

    "storybook": "start-storybook -p 6006",

    "build:storybook": "build-storybook -o doc",

    "build": "yarn run clean && yarn build:lib && yarn build:es&& yarn build:umd && yarn build:css",

    "clean": "rimraf dist lib es",

    "build:lib": "cross-env OUTPUT\_MODULE=commonjs babel components -d lib",

    "build:umd": "webpack ",

    "build:css": "gulp",

    "build:es": "babel components -d es"

},

### 12. MD文件

<p align="center">

<a href="javascript:viod(0)">

<img width="200" src="http://huyongdi.com/favicon.ico">

</a>

</p>

<h1 align="center">hyd-ui</h1>

<h4 align="center">react按需加载组件库脚手架</h4>

**## 1: 安装**

> 使用 npm

```

npm i hyd-ui --save

```

> 使用 yarn

```

yarn add hyd-ui

```

**## 2: 使用(按需导入)**

> 1. 手动引入

```js

import Button from 'hyd-ui/lib/button'; // 加载 JS

import 'hyd-ui/lib/button/style/css'; // 加载 CSS

```

> 2. 利用插件 [babel-plugin-import](https://github.com/ant-design/babel-plugin-import

```js

import { Button , Button1 } from "hyd-ui"

// .babelrc.js 或加入到webpack.config.js的babel-loader中

module.exports = {

plugins: [

["import", {

"libraryName": "hyd-ui",

"libraryDirectory": "es",

"style": true

}]

]

}

// 项目引入了多个库

module.exports = {

plugins: [

["import", {

"libraryName": "antd",

"libraryDirectory": "es",

"style": true

}],

["import", {

"libraryName": "hyd-ui",

"libraryDirectory": "es",

"style": true //需要项目已经支持less

},'hyd-ui'],

]

}

```

**## 3: 使用文档**

```js

doc目录下运行index.html (通过storybook生成，待完善)

目前有2个测试按钮(字体颜色不同)：

class Test extends React.Component {

render(){

return (

<Button text="毛衣"></Button>

<Button1 text="花欠"></Button1>

)

}

}

```

**## 4: 参考文档**

- [ant-design](https://github.com/ant-design/ant-design)

- [gulp](https://www.gulpjs.com.cn/docs/)

- [webpack](https://www.webpackjs.com/)

- [storybook](https://github.com/storybookjs/storybook)

### 13. 注意事项

1. autoprefixer 新版本(9.6.1)警告问题

  Replace Autoprefixer browsers option to Browserslist config.

  Use browserslist key in package.json or .browserslistrc file.

  Using browsers option cause some error. Browserslist config

  can be used for Babel, Autoprefixer, postcss-normalize and other tools.

  If you really need to use option, rename it to overrideBrowserslist.

  Learn more at:

  https://github.com/browserslist/browserslist#readme

  https://twitter.com/browserslist

解决办法：下面代码中browsers换成overrideBrowserslist

autoprefixer({

    browsers: [

        "Android 4.1",

        "iOS 7.1",

        "Chrome > 31",

        "ff > 31",

        "ie >= 8"

    ],

})

### 14. 引入TS

参考 <https://www.tslang.cn/docs/handbook/react-&-webpack.html>

npm install--save react react - dom @types/react @types/react - dom

npm install--save - dev typescript

npm install awesome - typescript - loader

思路：通过ts自带的tsc方法把tsx文件转化为jsx，然后走之前的逻辑

①. 根目录下新建tsconfig.json文件：

{

    "compilerOptions": {

        "strictNullChecks": true,

        "moduleResolution": "node",

        "esModuleInterop": true,

        "experimentalDecorators": true,

        "jsx": "preserve",

        "noUnusedParameters": true,

        "noUnusedLocals": true,

        "noImplicitAny": true,

        "target": "es6",

        "lib": ["dom", "es2017"]

    },

    "exclude": ["node\_modules", "lib", "es"]

}

②. package.json：

"tsc": "tsc"

"clean": "rimraf dist lib es src/\*\*/\*.jsx",

"build": "npm run clean &&npm run tsc&& npm run build:lib && npm run build:es&& npm run build:umd && npm run build:css",

③. webpack.config.js :

entry: {

    [name]: ["./src/components/index.jsx"]

},

resolve: {

    enforceExtension: false,  //导入语句去掉文件后缀

        extensions: [".js", ".jsx", ".json", ".less", ".css", '.ts', '.tsx']

},

rules里面加tsx配置

{ test: /\.tsx?$/, loader: "awesome-typescript-loader" },

④．components：

清空之前的文件

新建index.tsx：export {default as Hello} from “./hello”

新建hello文件夹

文件夹下新建index.tsx

文件夹下新建style文件夹

index.tsx:

import React from "react";

export interface HelloProps { compiler: string; framework: string; }

// 'HelloProps' describes the shape of props.

// State is never set so we use the '{}' type.

export default class Hello extends React.Component<HelloProps, {}> {

    render() {

        return <h1 className="ty-h1">Hello from {this.props.compiler} and {this.props.framework}!</h1>;

    }

}

style文件夹下新建index.tsx ( import “./index.less” )

style文件夹下新建index.less: ( .ty-h1{color:red} )

写到这里，项目已经可以通过less配置按需加载了，与之前相同

<Hello compiler="TypeScript" framework="React" />

⑤. 如果需要适配css style文件夹下面新建css.js

import './index.less'

⑥．为了使TS项目也能使用组件，通过antd源码和代码提示，需要生成d.ts文件

* ts.config文件新增

"declaration": true,

* gulpfile.js里面新加一个任务

const DIR = {  //生成绝对路径

    less: path.resolve(\_\_dirname, "src/components/\*\*/\*.less"),

    buildSrc: [

        path.resolve(\_\_dirname, "src/components/\*\*/index.less")

    ],

    lib: path.resolve(\_\_dirname, "lib"),

    es: path.resolve(\_\_dirname, "es"),

    dist: path.resolve(\_\_dirname, "dist"),

    dTs: path.resolve(\_\_dirname, "src/components/\*\*/\*.d.ts")

};

gulp.task("copDts", () => {

    return gulp

        .src(DIR.dTs)

        .pipe(gulp.dest(DIR.lib))

        .pipe(gulp.dest(DIR.es));

});

### 15. 加人utils通用函数支持

①．src目录下新建utils文件夹 utils目录下新建index.ts

export {isMac, isImsiImei} from "./reg";

②．utils目录下新建reg文件夹，reg文件夹下面新建index.ts

let isMac = function (str: string) {

    let rep = /^([A-Fa-f0-9]{2}){6}$/

    return rep.test(str)

}

let isImsiImei = function (str: string) {

    let rep = /^\d{15}$/

    return rep.test(str)

}

export { isMac, isImsiImei }

③．package.json里面新增script项 build:utils

"scripts": {

    "storybook": "start-storybook -p 6006",

    "build:storybook": "build-storybook -o doc",

    "build": "npm run clean &&npm run tsc&& npm run build:lib && npm run build:es&& npm run build:umd && npm run build:css && npm run build:utils &&npm run clean:js",

    "clean": "rimraf dist lib es utils && npm run clean:js",

    "clean:js": "rimraf src/\*\*/\*.jsx src/\*\*/\*.js",

    "build:lib": "cross-env OUTPUT\_MODULE=commonjs babel src/components -d lib",

    "build:utils": "cross-env OUTPUT\_MODULE=commonjs babel src/utils -d utils",

    "build:umd": "webpack ",

    "build:css": "gulp",

    "build:es": "babel src/components -d es",

    "tsc": "tsc"

},

"files": [

    "lib",

    "es",

    "dist",

    "utils"

],

这样就可以导入了

import {isMac,isImsiImei} from 'tyfront/utils'

### 16. 组件库打包的一些总结

antd目录：

1. dist:：umd模块，浏览器使用

1. lib：babel转译，不打包
2. es：babel转译，保留es module

关于es目录：

不管有没有配置babel-plugin-import

import {Button} from 'antd'走的都是es目录

原因是package.json中的module字段，该字段由rollup最先提出，webpack2开始支持该字段

关于按需加载：

1. 开发环境下不管是lib还是es，按需加载都需要配置babel-plugin-import
2. 对于webpack2+，rollup这类打包工具，生产环境会自动根据package.json中配置的module字段和sideEffects进行tree-shaking，配置babel-plugin-import只提供了自动加载样式的功能

babel-plugin-import的作用：

1. lib目录开发环境和生产环境的按需加载
2. es目录开发环境的按需加载
3. 自动加载样式

module字段和tree-shaking解释 https://segmentfault.com/a/1190000014286439