配置

基本配置

plugins

- 类型： `Array`
- 默认值： `[]`

配置插件列表。

数组项为指向插件的路径，可以是 npm 依赖、相对路径或绝对路径。如果是相对路径，则会从项目根目录开始找。比如：

```js
export default {
  plugins: [
    // npm 依赖
    'umi-plugin-react',
    // 相对路径
    './plugin',
    // 绝对路径
    `${__dirname}/plugin.js`,
  ],
};
```

如果插件有参数，则通过数组的形式进行配置，第一项是路径，第二项是参数，类似 babel 插件的配置方式。比如：

```js
export default {
  plugins: [
    // 有参数
    [
      'umi-plugin-react',
      {
        dva: true,
        antd: true,
      },
    ],
```

routes

- 类型： `Array`
- 默认值： `null`

配置路由。

umi 的路由基于 react-router 实现，配置和 react-router@4 基本一致，详见路由配置章节。

```js
export default {
  routes: [
    {
      path: '/',
      component: '../layouts/index',
      routes: [
        { path: '/user', redirect: '/user/login' },
        { path: '/user/login', component: './user/login' },
      ],
    },
  ],
};
```

注：

1. component 指向的路由组件文件是从 `src/pages` 目录开始解析的
2. 如果配置了 `routes` ，则优先使用配置式路由，且约定式路由会不生效

disableRedirectHoist

- 类型： `Boolean`
- 默认值： `false`

禁用 redirect 上提。

出于一些原因的考虑，我们在处理路由时把所有 redirect 声明提到路由最前面进行匹配，但这导致了一些问题，所以添加了这个配置项，禁用 redirect 上提。

```
};
```

history

- 类型： `String | [String, Object]`
- 默认值： `browser`

指定 history 类型，可选 `browser` 、 `hash` 和 `memory` 。

比如：

```js
export default {
  history: 'hash',
};
```

outputPath

- 类型： `String`
- 默认值： `./dist`

指定输出路径。

base

- 类型： `String`
- 默认值： `/`

指定 react-router 的 base，部署到非根目录时需要配置。

publicPath

- 类型： `String`
- 默认值： `/`

指定 webpack 的 publicPath，指向静态资源文件所在的路径。

runtimePublicPath

- 类型： `Boolean`
- 默认值： `false`

值为 `true` 时使用 HTML 里指定的 `window.publicPath` 。

cssPublicPath 2.2.5+

- 类型： `String`
- 默认值：同 publicPath

为 CSS 指定额外的 publicPath 。

mountElementId

- 类型： `String`
- 默认值： `root`

指定 react app 渲染到的 HTML 元素 id。

minimizer

- 类型： `String`
- 默认值： `uglifyjs`
- 选项： `uglifyjs|terserjs`

Which minimizer to use. UglifyJS does not support es6 while terser does.

hash

- Type: `Boolean`
- Default: `false`

是否开启 hash 文件后缀。

targets 2.1.0+

配置浏览器最低版本，会自动引入 polyfill 和做语法转换，配置的 targets 会和合并到默认值，所以不需要重复配置。

比如要兼容 ie11，需配置：

```js
export default {
  targets: {
    ie: 11,
  },
};
```

context

- 类型： `Object`
- 默认值： `{}`

配置全局 context，会覆盖到每个 pages 里的 context。

exportStatic

- 类型： `Boolean | Object`
- 默认值： `false`

如果设为 `true` 或 `Object`，则导出全部路由为静态页面，否则默认只输出一个 index.html。

比如：

```
"exportStatic": {}
```

exportStatic.htmlSuffix

- 类型： `Boolean`
- 默认值： `false`

启用 `.html` 后缀。

exportStatic.dynamicRoot

- 类型： `Boolean`
- 默认值： `false`

部署到任意路径。

singular

- 类型： `Boolean`
- 默认值： `false`

如果设为 `true` ，启用单数模式的目录。

- src/layout/index.js
- src/page
- model（如果有开启 umi-plugin-dva 插件的话）

mock.exclude 2.4.5+

- 类型： `Array` of `String`
- 默认值： `[]`

排除 mock 目录下不作 mock 处理的文件。

比如要 exclude 所有 `_` 前缀的文件和文件夹，

```js
export default {
  mock: {
    exclude: ['mock/**/_*.js', 'mock/_*/**/*.js'],
  },
};
```

block 2.7.0+

- 类型: `Object`
- 默认值: `{ defaultGitUrl: "https://github.com/umijs/umi-blocks" }`

```
    defaultGitUrl: 'https://github.com/ant-design/pro-blocks',
    npmClient: 'cnpm', // 优先级低于 umi block add [block] --npm-client
  },
};
```

ssr beta 2.8.0+

- 类型: `Boolean | Object`
- 默认值: `false`

用于服务端渲染（Server-Side Render）。

开启后，生成客户端静态文件的同时，也会生成 `umi.server.js` 和 `ssr-client-mainifest.json` 文件。

```js
export default {
  ssr: {
    // https://github.com/liady/webpack-node-externals#optionswhitelist-
    externalWhitelist: [],
    // 客户端资源 manifest 文件名，默认是 ssr-client-mainifest.json
    manifestFileName: 'ssr-client-mainifest.json',
  },
};
```

其中 `ssr-client-mainifest.json` 是按路由级别的资源映射文件，例如：

```json
{
  "/": {
    "js": [
      "umi.6791e2ab.js",
      "vendors.aed9ac63.async.js",
      "layouts__index.12df59f1.async.js",
      "p__index.c2bcd95d.async.js"
    ],
    "css": [
      "umi.baa67d11.css",
      "vendors.431f0bf4.chunk.css",
      "layouts__index.0ab34177.chunk.css",
      "p__index.1353f910.chunk.css"
    ]
```

```
      "umi.6791e2ab.js",
      "vendors.aed9ac63.async.js",
      "layouts__index.12df59f1.async.js",
      "p__news__$id.204a3fac.async.js"
    ],
    "css": ["umi.baa67d11.css", "vendors.431f0bf4.chunk.css", "layouts__index.0ab341
  }
}
```

在 Node.js 中使用如下：

```js
/**
 *
 * @param {*}
 * ctx（server 执行上下文，`serverRender` 通过 `ctx.req.url` 获取当前路由）
 * @return html 片段
 */
async function UmiServerRender(ctx) {
  // mock 一个 window 对象
  global.window = {};
  // 引入模块
  const serverRender = require('./dist/umi.server');
  // 提供 react-dom/server，避免 React hooks ssr 报错
  const { ReactDOMServer } = serverRender;

  const {
    // 当前路由元素
    rootContainer,
    // 页面模板
    htmlElement,
    // 匹配成功的前端路由，比如 /user/:id
    matchPath,
    // 初始化 store 数据，若使用 dva
    g_initialData,
  } = await serverRender.default(ctx);

  // 元素渲染成 html
  const ssrHtml = ReactDOMServer.renderToString(htmlElement);
  return ssrHtml;
}
```

JS

```
// pages/news/$id.jsx
const News = props => {
  const { id, name, count } = props || {};

  return (
    <div>
      <p>
        {id}-{name}
      </p>
    </div>
  );
};

/**
 *
 * @param {*}
 * {
 *  route （当前路由信息）
 *  store（需开启 `dva: true`, `store.dispatch()` 会返回 Promise）
 *  isServer (是否为服务端执行环境)
 * }
 */
News.getInitialProps = async ({ route, store, isServer }) => {
  const { id } = route.params;
  const data = [
    {
      id: 0,
      name: 'zero',
    },
    {
      id: 1,
      name: 'hello',
    },
    {
      id: 2,
      name: 'world',
    },
  ];
  return Promise.resolve(data[id] || data[0]);
};
```

预渲染（Pre-Rendering）使用，umi-example-ssr-with-egg

webpack

chainWebpack

通过 webpack-chain 的 API 扩展或修改 webpack 配置。

比如：

```js
chainWebpack(config, { webpack }) {
  // 设置 alias
  config.resolve.alias.set('a', 'path/to/a');

  // 删除进度条插件
  config.plugins.delete('progress');
}
```

theme

配置主题，实际上是配 less 变量。支持对象和字符串两种类型，字符串需要指向一个返回配置的文件。比如：

```
"theme": {
  "@primary-color": "#1DA57A"
}
```

或者，

```
"theme": "./theme-config.js"
```

treeShaking 2.4.0+

- 类型： `Boolean`
- 默认值： `false`

e.g.

```js
export default {
  treeShaking: true,
};
```

比如 ant-design-pro 开启 tree-shaking 之后，gzip 后的尺寸能减少 10K。

define

通过 webpack 的 DefinePlugin 传递给代码，值会自动做 `JSON.stringify` 处理。比如：

```js
"define": {
  "process.env.TEST": 1,
  "USE_COMMA": 2,
}
```

externals

配置 webpack 的?externals?属性。比如：

```js
// 配置 react 和 react-dom 不打入代码
"externals": {
  "react": "window.React",
  "react-dom": "window.ReactDOM"
}
```

alias

配置 webpack 的 resolve.alias 属性。

devServer

配置 webpack 的 devServer 属性。

devtool

disableCSSModules

禁用 CSS Modules。

disableCSSSourceMap

禁用 CSS 的 SourceMap 生成。

extraBabelPresets

定义额外的 babel preset 列表，格式为数组。

extraBabelPlugins

定义额外的 babel plugin 列表，格式为数组。

extraBabelIncludes

定义额外需要做 babel 转换的文件匹配列表，格式为数组，数组项是 webpack#Condition。

extraPostCSSPlugins

定义额外的 PostCSS 插件，格式为数组。

以使用 postcss-px-to-viewport 插件为例。

先使用 `npm install postcss-px-to-viewport --save-dev` 安装，然后配置 extraPostCSSPlugins

```js
import pxToViewPort from 'postcss-px-to-viewport';

const config = {
  ...otherConfig,
  extraPostCSSPlugins: [
    pxToViewPort({
      viewportWidth: 375,
      viewportHeight: 667,
```

```
        minPixelValue: 1,
        mediaQuery: false,
      }),
    ],
  };
```

cssModulesExcludes

指定项目目录下的文件不走 css modules，格式为数组，项必须是 css 或 less 文件。

copy

定义需要单纯做复制的文件列表，格式为数组，项的格式参考 copy-webpack-plugin 的配置。

比如：

```html
"copy": [
  {
    "from": "",
    "to": ""
  }
]
```

proxy

配置 webpack-dev-server 的 proxy 属性。如果要代理请求到其他服务器，可以这样配：

```html
"proxy": {
  "/api": {
    "target": "http://jsonplaceholder.typicode.com/",
    "changeOrigin": true,
    "pathRewrite": { "^/api" : "" }
  }
}
```

sass

配置 node-sass 的选项。注意：使用 sass 时需在项目目录安装 node-sass 和 sass-loader 依赖。

manifest

配置后会生成 asset-manifest.json，option 传给 https://www.npmjs.com/package/webpack-manifest-plugin。比如：

```html
"manifest": {
  "basePath": "/app/"
}
```

ignoreMomentLocale

忽略 moment 的 locale 文件，用于减少尺寸。

lessLoaderOptions

给 less-loader 的额外配置项。

cssLoaderOptions

给 css-loader 的额外配置项。

autoprefixer 2.4.3+

配置传给 autoprefixer 的配置项。

- 类型： `Object`
- 默认： `{ browsers: DEFAULT_BROWSERS, flexbox: 'no-2019' }`

如果你想兼容旧版本 iOS Safari 的 flexbox，应该需要配置上 `flexbox: true` 。

uglifyJSOptions

配置传给 uglifyjs-webpack-plugin@2.x 的配置项。

- 类型： `Object | Function`
- 默认：af-webpack/src/getConfig/uglifyOptions.js

如果值为 `Object`，会做浅合并。

比如：

```js
export default {
  uglifyJSOptions: {
    parallel: false,
  },
};
```

如果要修改深层配置，可以用函数的形式。

比如：

```js
export default {
  uglifyJSOptions(opts) {
    opts.uglifyOptions.compress.warning = true;
    return opts;
  },
};
```

browserslist deprecated

配置 browserslist，同时作用于 babel-preset-env 和 autoprefixer。

e.g.

```js
export default {
  browserslist: ['> 1%', 'last 2 versions'],
};
```

1. 配置 browserslist 之后，targets 会失效
2. 不推荐使用 browserslist，推荐用 targets

在 GitHub 上编辑此页

Last Updated: 2019/7/18 下午3:54:03