UmiJS

路由

> **提示**
>
> 下文介绍的路由使用可以在 umi-examples/routes 和 umi-examples/routes-via-config 里找到示例代码。

umi 会根据 `pages` 目录自动生成路由配置。

约定式路由

基础路由

假设 `pages` 目录结构如下：

```
+ pages/
  + users/
    - index.js
    - list.js
  - index.js
```

那么，umi 会自动生成路由配置如下：

```
[
  { path: '/', component: './pages/index.js' },
  { path: '/users/', component: './pages/users/index.js' },
  { path: '/users/list', component: './pages/users/list.js' },
]
```

动态路由

umi 里约定，带 `$` 前缀的目录或文件为动态路由。

比如以下目录结构：

```
    - index.js
    - comments.js
  + users/
    $id.js
  - index.js
```

会生成路由配置如下：

```js
[
  { path: '/', component: './pages/index.js' },
  { path: '/users/:id', component: './pages/users/$id.js' },
  { path: '/:post/', component: './pages/$post/index.js' },
  { path: '/:post/comments', component: './pages/$post/comments.js' },
]
```

可选的动态路由

umi 里约定动态路由如果带 `$` 后缀，则为可选动态路由。

比如以下结构：

```
+ pages/
  + users/
    - $id$.js
  - index.js
```

会生成路由配置如下：

```js
[
  { path: '/': component: './pages/index.js' },
  { path: '/users/:id?': component: './pages/users/$id$.js' },
]
```

嵌套路由

umi 里约定目录下有 `_layout.js` 时会生成嵌套路由，以 `_layout.js` 为该目录的 layout 。

```
+ pages/
  + users/
    - _layout.js
    - $id.js
    - index.js
```

会生成路由配置如下：

```js
[
  { path: '/users', component: './pages/users/_layout.js',
    routes: [
     { path: '/users/', component: './pages/users/index.js' },
     { path: '/users/:id', component: './pages/users/$id.js' },
   ],
  },
]
```

全局 layout

约定 `src/layouts/index.js` 为全局路由，返回一个 React 组件，通过 `props.children` 渲染子组件。

比如：

```js
export default function(props) {
  return (
    <>
      <Header />
      { props.children }
      <Footer />
    </>
  );
}
```

不同的全局 layout

你可能需要针对不同路由输出不同的全局 layout，umi 不支持这样的配置，但你仍可以在 `layouts/index.js` 对 location.path 做区分，渲染不同的 layout 。

```js
export default function(props) {
  if (props.location.pathname === '/login') {
    return <SimpleLayout>{ props.children }</SimpleLayout>
  }

  return (
    <>
      <Header />
      { props.children }
      <Footer />
    </>
  );
}
```

404 路由

约定 `pages/404.js` 为 404 页面，需返回 React 组件。

比如：

```js
export default () => {
  return (
    <div>I am a customized 404 page</div>
  );
};
```

> 注意：开发模式下，umi 会添加一个默认的 404 页面来辅助开发，但你仍然可通过精确地访问 `/404` 来验证 404 页面。

通过注释扩展路由

约定路由文件的首个注释如果包含 yaml 格式的配置，则会被用于扩展路由。

比如：

```
+ pages/
  - index.js
```

```js
/**
 * title: Index Page
 * Routes:
 *   - ./src/routes/a.js
 *   - ./src/routes/b.js
 */
```

则会生成路由配置：

```js
[
  { path: '/', component: './index.js',
    title: 'Index Page',
    Routes: [ './src/routes/a.js', './src/routes/b.js' ],
  },
]
```

配置式路由

如果你倾向于使用配置式的路由，可以配置 `.umirc.(ts|js)` 或者 `config/config.(ts|js)` 配置文件中的 `routes` 属性，**此配置项存在时则不会对 `src/pages` 目录做约定式的解析。**

比如：

```js
export default {
  routes: [
    { path: '/', component: './a' },
    { path: '/list', component: './b', Routes: ['./routes/PrivateRoute.js'] },
    { path: '/users', component: './users/_layout',
      routes: [
        { path: '/users/detail', component: './users/detail' },
        { path: '/users/:id', component: './users/id' }
      ]
    },
  ],
};
```

注意：

权限路由

umi 的权限路由是通过配置路由的 `Routes` 属性来实现。约定式的通过 yaml 注释添加，配置式的直接配上即可。

比如有以下配置：

```js
[
  { path: '/', component: './pages/index.js' },
  { path: '/list', component: './pages/list.js', Routes: ['./routes/PrivateRoute.js'
]
```

然后 umi 会用 `./routes/PrivateRoute.js` 来渲染 `/list` 。

`./routes/PrivateRoute.js` 文件示例：

```js
export default (props) => {
  return (
    <div>
      <div>PrivateRoute (routes/PrivateRoute.js)</div>
      { props.children }
    </div>
  );
}
```

路由动效

路由动效应该是有多种实现方式，这里举 react-transition-group 的例子。

先安装依赖，

```sh
$ yarn add react-transition-group
```

在 layout 组件（ `layouts/index.js` 或者 pages 子目录下的 `_layout.js` ）里在渲染子组件时用 TransitionGroup 和 CSSTransition 包裹一层，并以 `location.pathname` 为 key，

```
export default withRouter(
  ({ location }) =>
    <TransitionGroup>
      <CSSTransition key={location.pathname} classNames="fade" timeout={300}>
        { children }
      </CSSTransition>
    </TransitionGroup>
)
```

上面用到的 `fade` 样式，可以在 `src` 下的 `global.css` 里定义：

```css
.fade-enter {
  opacity: 0;
  z-index: 1;
}

.fade-enter.fade-enter-active {
  opacity: 1;
  transition: opacity 250ms ease-in;
}
```

面包屑

面包屑也是有多种实现方式，这里举 react-router-breadcrumbs-hoc 的例子。

先安装依赖，

```sh
$ yarn add react-router-breadcrumbs-hoc
```

然后实现一个 `Breakcrumbs.js` ，比如：

```js
import NavLink from 'umi/navlink';
import withBreadcrumbs from 'react-router-breadcrumbs-hoc';

// 更多配置请移步 https://github.com/icd2k3/react-router-breadcrumbs-hoc
const routes = [
  { path: '/', breadcrumb: '首页' },
```

```
export default withBreadcrumbs(routes)(({ breadcrumbs }) => (
  <div>
    {breadcrumbs.map((breadcrumb, index) => (
      <span key={breadcrumb.key}>
        <NavLink to={breadcrumb.props.match.url}>
          {breadcrumb}
        </NavLink>
        {(index < breadcrumbs.length - 1) && <i> / </i>}
      </span>
    ))}
  </div>
));
```

然后在需要的地方引入此 React 组件即可。

启用 Hash 路由

umi 默认是用的 Browser History，如果要用 Hash History，需配置：

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

Scroll to top

在 layout 组件（ `layouts/index.js` 或者 pages 子目录下的 `_layout.js` ）的 componentDidUpdate 里决定是否 scroll to top，比如：

```
import { Component } from 'react';
import withRouter from 'umi/withRouter';

class Layout extends Component {
  componentDidUpdate(prevProps) {
    if (this.props.location !== prevProps.location) {
      window.scrollTo(0, 0);
    }
  }
  render() {
```

```
}

export default withRouter(Layout);
```

参考

- https://reacttraining.com/react-router/

在 GitHub 上编辑此页

Last Updated: 2019/6/26 上午11:58:50

← 目录及约定 在页面间跳转 →