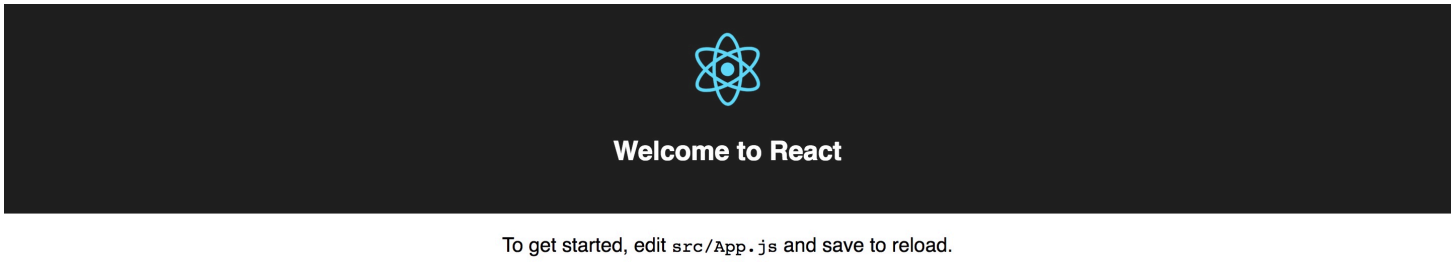
TODOMVC README.md

Quick Overview

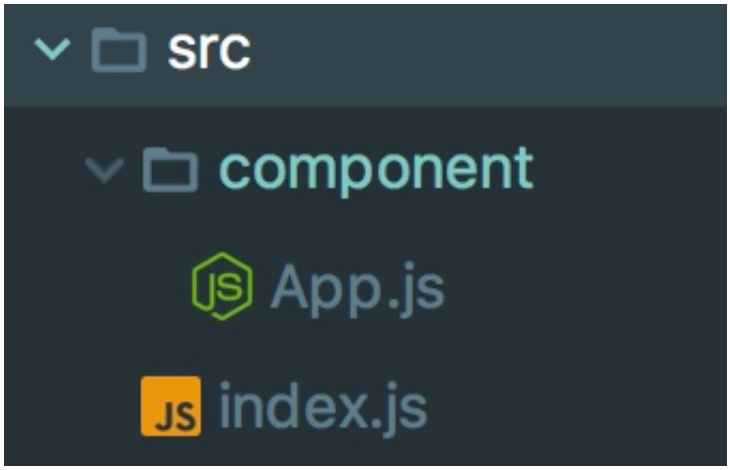
```
npx create-react-app todomvc
cd todomvc
npm start
npm install
npm start
npm i todomvc-app-css
npm i classnames --save-dev
```



- 如果能看到上面的页面，准备阶段已经完成

项目准备

- 打开todomvc
- src目录下创建新的目录`component`
- 将App.js文件移入到component目录下
- 将src目录下除了index.js文件以及component目录其他所有的文件都删除



- 如果src的目录结构如下项目构建即可完成

index.js

- 引入todomvc-app-css

```
  import 'todomvc-app-css/index.css'
```

index.js文件内容如下

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './component/App';
import 'todomvc-app-css/index.css';

ReactDOM.render(<App />, document.getElementById('root'));
```

App.js

- 删除多余代码，保证文件中内容如下

```
import React, { Component } from 'react';
class App extends Component {
  render() {
    return (
      <div>
      </div>
    );
  }
}
export default App;
```

此时发现命令行中

```
Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://10.201.132.70:3000/

Note that the development build is not optimized.
To create a production build, use yarn build.
```

浏览器中输入`http://localhost:3000/`即可看到一个空白的页面

public/index.html

- 将public目录下除了index.html以外的所有文件删除
- index.html内容如下

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>React TodoMVC Example</title>
</head>
<body>
<div class="todoapp" id="root"></div>
<!--
  This HTML file is a template.
  If you open it directly in the browser, you will see an empty page.

  You can add webfonts, meta tags, or analytics to this file.
  The build step will place the bundled scripts into the <body> tag.

  To begin the development, run `npm start` in this folder.
  To create a production bundle, use `npm run build`.
-->
</body>
</html>
```

请不要停止命令行中运行的`npm start`命令 他会帮助你监听着代码变化实时

在页面上显示最新的内容

开始编码

设计标题TODO

- `component`目录下创建一个新的文件`Header.js`
- 开始编写文件内容
- 文件中首先引入React

```
import React, {Component} from 'react'
```

- 创建一个名为`Header`的组件类，其中包含一个render方法

```
class Header extends Component {
render() {
return (
<header className="header">
  <h1>todos</h1>
</header>)
}
}
```

- 将这个组件暴露出去

```
export default Header;
```

- 在App.js文件中将其引入

```
import Header from './Header';
class App extends Component {
render() {
return (
  <div>
    <Header />
  </div>
);
}
}
```

此时打开浏览器可以看到一个todos的标题'

设计输入框

- component目录下创建一个新的文件`TodoTextInput.js`包含一个input框
- 其中引入React的依赖、创建TodoTextInput组件类其中包含一个render方法、将该组件类暴露出去

```
import React, {Component, PropTypes} from 'react'
import classnames from 'classnames'
class TodoTextInput extends Component {
    render() {
        return (
            <input
                type="text"
                autoFocus="true"
                placeholder="What needs to be done?"
                className='new-todo'
            />
        )
    }
}
export default TodoTextInput;
```

- 接着在Header.js中引入TodoTextInput

```
import React, {Component} from 'react'
import TodoTextInput from "./TodoTextInput";
class Header extends Component {
render() {
return (
<header className="header">
  <h1>todos</h1>
  <TodoTextInput/>
</header>)
}
}
export default Header;
```

- 给输入框添加功能
    - 添加onKeyDown事件

```
render() {
return (
  <input className='new-todo'
         type="text"
         placeholder="What needs to be done?"
         autoFocus="true"
         onKeyDown={this.handleSubmit.binf(this)} />
)
}
```

- 完成handleSubmit方法

```
import React, {Component, PropTypes} from 'react'

export default class TodoTextInput extends Component {

  handleSubmit = e => {
    const text = e.target.value.trim()
    if (e.which === 13) {
      this.props.onSave(text)
    }
  }

  render() {
    return (
      <input className='new-todo'
             type="text"
             placeholder="What needs to be done?"
             autoFocus="true"
             onKeyDown={this.handleSubmit} />
    )
  }
}
```

- 去父组件定义onSave方法

```
import React, {Component} from 'react'
import TodoTextInput from "./TodoTextInput";

class Header extends Component {
  render() {
    return (
    <header className="header">
      <h1>todos</h1>
      <TodoTextInput
        onSave={this.props.onSave}
      />
    </header>)
  }
}

export default Header;
```

- 去根组件(App)定义onSave方法
    - 先在跟组件定一个state数据todos

```
import React, {Component} from 'react'
import Header from './Header'
import MainSection from './MainSection'

const initialState = [
  {
    text: 'React ES6 TodoMVC',
    completed: false,
    id: 0
  }
]

class App extends Component {
  constructor(props) {
    super(props)
    this.state = {
      todos: initialState,
      gameID: null,
      player: 0
    }
  }
```

- 接着在App组件中定义onSave方法，一旦事件被触发修改state todos的数据

```
const todos = [
  {
    id: this.state.todos.reduce((maxId, todo) => Math.max(todo.id, maxId
), -1) + 1,
    completed: false,
    text: text
  },
  ...this.state.todos
]
this.setState({todos})
```

在根组件准备好数据，传递给展示列表组件MainSection

```
render() {
return (
  <div>
    <Header
      onSave={this.onSave.bind(this)}
    />
    <MainSection todos={this.state.todos}/>
  </div>
);
}
```

接着请自行动手定义MainSection 组件并完成功能