1.初始化项目

```
mkdir zhufeng-react-typesript
cd zhufeng-react-typesript
cnpm init -y
touch .gitignore  //mac或linux
type nul > .gitignore  //window
```

2.安装依赖

- types开头的包都是typeScript的声明文件，可以进入node_modules/@types/XX/index.d.ts进行查看
- DefinitelyTyped

```
cnpm i react react-dom @types/react @types/react-dom react-router-dom @ty
pes/react-router-dom react-transition-group @types/react-transition-group
react-swipe @types/react-swipe  -S
cnpm i webpack webpack-cli webpack-dev-server html-webpack-plugin -D
cnpm i typescript ts-loader source-map-loader -D
cnpm i redux react-redux @types/react-redux redux-thunk  redux-logger @ty
pes/redux-logger -S
cnpm i connected-react-router -S
```

包名	作用
react @types/react	react核心库
react-dom @types/react-dom	react操作DOM库
react-router-dom @types/react-router-dom	react路由库
react-transition-group @types/react-transition-group	react动画库
react-swipe @types/react-swipe	react轮播图组件库
webpack	webpack核心库

webpack-cli	webpack命令行文件
webpack-dev-server	webpack开发服务器
html-webpack-plugin	webpack用于生成html的插件
redux	全局状态管理库
react-redux @types/react-redux	连接react和redux的库
redux-thunk	可以让store派发一个函数的中间件
redux-logger @types/redux-logger	可以在状态改变前后打印状态的中间件
typescript	JavaScript语言扩展
ts-loader	可以让Webpack使用TypeScript的标准配置文件 `tsconfig.json` 编译TypeScript代码
source-map-loader	使用任意来自Typescript的sourcemap输出，以此通知webpack何时生成自己的sourcemaps,这让你在调试最终生成的文件时就好像在调试TypeScript源码一样

3.支持typescript

- 我们需要生成一个 `tsconfig.json` 文件来告诉 `ts-loader` 如何编译代码TypeScript代码

```
cnpm i typescript -g
tsc --init
```

Successfully created a tsconfig.json file

```
{
  "compilerOptions": {
    "outDir": "./dist",
    "sourceMap": true,
    "noImplicitAny": true,
    "module": "commonjs",
    "target": "es5",
    "jsx": "react"
  },
  "include": [
    "./src/**/*"
  ]
}
```

参数	含义
outDir	指定输出目录
sourceMap	把 ts 文件编译成 js 文件的时候，同时生成对应的sourceMap文件
noImplicitAny	如果为true的话，TypeScript 编译器无法推断出类型时，它仍然会生成 JavaScript 文件，但是它也会报告一个错误
module	模块化的代码规范
target	转换成es5
jsx	react模式会生成 `React.createElement` ，在使用前不需要再进行转换操作了，输出文件的扩展名为.js
include	需要编译的目录

4.编写webpack配置文件

webpack.config.js

```
const webpack=require('webpack');
const HtmlWebpackPlugin=require('html-webpack-plugin');
const path=require('path');
module.exports={
    mode: 'development',
    entry: "./src/index.tsx",
    output: {
        filename: "bundle.js",
        path: path.join(__dirname,'dist')
    },
    devtool: "source-map",
    devServer: {
        hot: true,
        contentBase: path.join(__dirname,'dist'),
        historyApiFallback: {
            index:'./index.html'
        }
    },
    resolve: {
        extensions: [".ts", ".tsx", ".js", ".json"]
    },
    module: {
        rules: [{
                test: /\.tsx?$/,
                loader: "ts-loader"
            },
```

```
                {
                    enforce: "pre",
                    test: /\.tsx$/,
                    loader: "source-map-loader"
                }
            ]
        },

        plugins: [
            new HtmlWebpackPlugin({
                template:'./src/index.html'
            }),
            new webpack.HotModuleReplacementPlugin()
        ],
    };
```

5. 使用React

5.1 index.js

src\index.tsx

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import Counter from './components/Counter';
ReactDOM.render((
    <Counter name="计数器"/>
),document.getElementById('root'));
```

5.2 Counter.tsx

src\components\Counter.tsx

```
import * as React from 'react';
export interface IProps{
    name: string
}
export interface IState{
    number: number,
    amount:number
}
export default class Counter extends React.Component<IProps,IState>{
    state = {number:0,amount:0}
    handleChange = (event:React.ChangeEvent<HTMLInputElement>)=>{
```

```
            this.setState({amount:parseInt(event.target.value)});
        }
        render() {
            const {name}=this.props;
            const {number,amount}=this.state;
            const container:React.CSSProperties = {
                color:'green',
                border:'1px solid red'
            }
            return (
                <div style={container}>
                    <p>{name}:{number}</p>
                    <input value={amount} onChange={event=>this.handleChange
(event)}/>
                    <button onClick={()=>this.setState({number:number+amoun
t})}>+</button>
                </div>
            )
        }
}
```

6. 代码检查

- ESLint 是一款插件化的 JavaScript 静态代码检查工具，ESLint 通过规则来描述具体的检查行为
- 官网

6.1 模块安装

```
cnpm i eslint typescript @typescript-eslint/parser @typescript-eslint/esl
int-plugin --save-dev
```

6.2 eslintrc配置文件

- 英文rules
- 中文rules
- .eslintrc.js

```
module.exports = {
    "parser":"@typescript-eslint/parser",
    "plugins":["@typescript-eslint"],
    "rules":{
        "no-var":"error",
```

```
            "no-extra-semi":"error",
            "@typescript-eslint/indent":["error",2]
        },
        "parserOptions": {
            "ecmaVersion": 6,
            "sourceType": "module",
            "ecmaFeatures": {
              "modules": true
            }
        }
    }
```

6.3 npm命令

```
    "scripts": {
        "start": "webpack",
        "build": "tsc",
    +    "eslint": "eslint src --ext .ts",
    +    "eslint:fix": "eslint src --ext .ts --fix"
    }
```

7. Git Hooks检查

- Git 基本已经成为项目开发中默认的版本管理软件，在使用 Git 的项目中，我们可以为项目设置 Git Hooks 来帮我们在提交代码的各个阶段做一些代码检查等工作
- 钩子（Hooks） 都被存储在 Git 目录下的 hooks 子目录中。 也就是绝大部分项目中的 .git/hook 目录
- 钩子分为两大类，客户端的和服务器端的
  - 客户端钩子主要被提交和合并这样的操作所调用
  - 而服务器端钩子作用于接收被推送的提交这样的联网操作，这里我们主要介绍客户端钩子

7.1 pre-commit

- `pre-commit` 就是在代码提交之前做些东西，比如代码打包，代码检测，称之为钩子（hook）
- 在commit之前执行一个函数（callback）。这个函数成功执行完之后，再继续commit，但是失败之后就阻止commit
- 在.git->hooks->下面有个pre-commit.sample*，这个里面就是默认的函数(脚本)样本

7.2 安装pre-commit

```
cnpm install pre-commit --save-dev
```

7.3 配置脚本

```
"scripts": {
    "build": "tsc",
    "eslint": "eslint src --ext .ts",
    "eslint:fix": "eslint src --ext .ts --fix"
  },
  "pre-commit": [
    "eslint"
  ]
```

> 如果没有在 `.git->hooks` 目录下生成 `pre-commit` 文件的话，则要手工创建 `node ./node_modules/pre-commit/install.js` ,如果找不到文件就重新删除并安装 node_modules

8. 编写单元测试

- 目前比较流行的React单测组合是Jest+Enzyme
- Jest是Facebook开发的一个测试框架，它集成了测试执行器、断言库、spy、mock、snapshot和测试覆盖率报告等功能。React项目本身也是使用Jest进行单测的，因此它们俩的契合度相当高。
- enzyme是由airbnb开发的React单测工具。它扩展了React的TestUtils并通过支持类似jQuery的find语法可以很方便的对render出来的结果做各种断言

8.1 安装单元测试模块

```
cnpm i jest @types/jest ts-jest enzyme @types/enzyme enzyme-adapter-react-16 @types/enzyme-adapter-react-16 -D
```

8.2 配置package.json

```
"scripts": {
    "dev": "webpack-dev-server",
    "eslint": "eslint src --ext .tsx",
    "eslint:fix": "eslint src --ext .tsx --fix",
```

```
    "test": "jest"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "ts",
      "tsx"
    ],
    "transform": {
      "^.+\\.tsx?$": "ts-jest"
    },
    "testMatch": [
      "<rootDir>/test/**/*.(spec|test).tsx"
    ]
  }
```

8.3 index.test.tsx

test\index.test.tsx

```
test('测试运行', () => {
   expect(1+1).toBe(2);
})
```

8.4 counter.spec.tsx

test\counter.spec.tsx

```
import * as React from 'react';
import { shallow, configure } from 'enzyme'
import * as  Adapter from 'enzyme-adapter-react-16'
import Counter from '../src/components/Counter';
configure({ adapter: new Adapter() })
test('Jest-React-TypeScript运行', () => {
    const renderer = shallow(<Counter name="计数器"/>);
  expect(renderer.text()).toContain('计数器');
})
```

9. 持续集成

- Travis CI 提供的是持续集成服务（Continuous Integration，简称 CI）。它绑定 Github 上面的项目，只要有新的代码，就会自动抓取。然后，提供一个运行环境，执行测试，完成构建，还

能部署到服务器

- 持续集成指的是只要代码有变更，就自动运行构建和测试，反馈运行结果。确保符合预期以后，再将新代码集成到主干
- 持续集成的好处在于，每次代码的小幅变更，就能看到运行结果，从而不断累积小的变更，而不是在开发周期结束时，一下子合并一大块代码

7.1 添加配置文件

.travis.yml

```
language: node_js
node_js:
    - '11'
install:
  - npm install
script:
  - npm test
branches:
  only:
    - master
```

7.2 创建并推送项目

```
git init
git add -A
git commit -m"init"
git push origin master
```

7.3 激活项目构建

- travis-ci