React是FaceBook（脸书）公司研发的一款JS框架（MVC）

**1. React的脚手架**

React是一款框架：具备自己开发的独立思想(MVC：Model View Controller)

-> 划分组件开发

-> 基于路由的SPA单页面开发

-> 基于ES6来编写代码(最后部署上线的时候，我们需要把ES6编译成ES5 =>基于Babel来完成编译)

-> 可能用到Less/Sass等，我们也需要使用对应的插件把他们进行预编译

-> 最后为了优化性能（减少HTTP请求次数），我们需要把JS或者CSS进行合并压缩

-> ...

-> webpack来完成以上页面组件合并、JS/CSS编译加合并等工作

前端工程化开发：

=>基于框架的组件化/模块化开发

=>基于WEBPACK的自动部署

但是配置WEBPACK是一个相对复杂的工作，我们需要自己安装很多的包，还需要自己写相对复杂的配置文件...如果我们有一个插件，基于它可以快速构建一套完整的自动化工程项目结构，那么有助于提高开发的效率 =>“脚手架”

VUE：VUE-CLI

REACT：CREATE-REACT-APP

**【create-react-app 的使用】**

>$ npm install create-react-app -g

把模块安装在全局环境下（目的：可以使用命令），MAC电脑安装的时候，前面需要加SUDO，否则没有权限

>$ create-react-app [项目名称]

基于脚手架命令，创建出一个基于React的自动化/工程化项目目录，和npm发包时候的命名规范一样，项目名称中不能出现：大写字母、中文汉字、特殊符号（-或者\_是可以的）等

**【脚手架生成目录中的一些内容】**

node\_modules 当前项目中依赖的包都安装在这里

.bin 本地项目中可执行命令，在package.json的scripts中配置对应的脚本即可（其中有一个就是：react-scripts命令）

public 存放的是当前项目的HTML页面（单页面应用放一个index.html即可，多页面根据自己需求放置需要的页面）

<!--

在REACT框架中，所有的逻辑都是在JS中完成的（包括页面结构的创建），如果想给当前的页面导入一些CSS样式或者IMG图片等内容，我们有两种方式：

1.在JS中基于ES6 Module模块规范，使用import导入，这样webpack在编译合并JS的时候，会把导入的资源文件等插入到页面的结构中（绝对不能在JS管控的结构中通过相对目录./或者../，导入资源，因为在webpack编译的时候，地址就不在是之前的相对地址了）

2.如果不想在JS中导入（JS中导入的资源最后都会基于WEBPACK编译），我们也可以把资源手动的在HTML中导入，但是HTML最后也要基于WEBPACK编译，导入的地址也不建议写相对地址，而是使用 %PUBLIC\_URL% 写成绝对地址

<link rel="manifest" href="%PUBLIC\_URL%/manifest.json">

<link rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico">

-->

src 项目结构中最主要的目录，因为后期所有的JS、路由、组件等都是放到这里面（包括需要编写的CSS或者图片等）

index.js 是当前项目的入口文件

.gitignore Git提交时候的忽略提交文件配置项

package.json 当前项目的配置清单

"dependencies": {

"react": "^16.4.1",

"react-dom": "^16.4.1",

"react-scripts": "1.1.4"

}

基于脚手架生成工程目录，自动帮我们安装了三个模块：react/react-dom/react-scripts

react-scripts集成了webpack需要的内容

->Babel一套

->CSS处理的一套

->eslint一套

->webpack一套

->其它的

没有less/sass的处理内容（项目中使用less，我们需要自己额外的安装）

----

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test --env=jsdom",

"eject": "react-scripts eject"

}

可执行的脚本“$ npm run start / $ yarn start”

start：开发环境下，基于webpack编译处理，最后可以预览当前开发的项目成果（在webpack中安装了webpack-dev-server插件，基于这个插件会自动创建一个WEB服务[端口号默认是3000]，webpack会帮我们自动打开浏览器，并且展示我们的页面，并且能够监听我们代码的改变，如果代码改变了，webpack会自动重新编译，并且刷新浏览器来完成重新渲染）

build：项目需要部署到服务器上，我们先执行 yarn build，把项目整体编译打包（完成后会在项目中生成一个build文件夹，这个文件夹中包含了所有编译后的内容，我们把它上传到服务器即可）;而且在服务上进行部署的时候，不需要安装任何模块了（因为webpack已经把需要的内容都打包到一个JS中了）

---

**2. React脚手架的深入剖析**

create-react-app脚手架为了让结构目录清晰，把安装的webpack及配置文件都集成在了react-scripts模块中，放到了node\_modules中

但是真实项目中，我们需要在脚手架默认安装的基础上，额外安装一些我们需要的模块，例如：react-router-dom/axios... 再比如：less/less-loader...

情况一：如果我们安装其它的组件，但是安装成功后不需要修改webpack的配置项，此时我们直接的安装，并且调取使用即可

情况二：我们安装的插件是基于webpack处理的，也就是需要把安装的模块配置到webpack中（重新修改webpack配置项了）

=>首先需要把隐藏到node\_modules中的配置项暴露到项目中

> $ yarn eject

首先会提示确认是否执行eject操作，这个操作是不可逆转的，一但暴露出来配置项，就无法在隐藏回去了

如果当前的项目基于GIT管理，在执行eject的时候，如果还有没有提交到历史的区的内容，需要先提交到历史区，然后在eject才可以，否则报错：This git repository has untracked files or uncommitted changes...

=>再去修改对应的配置项即可

一但暴露后，项目目录中多了两个文件夹：

config 存放的是webpack的配置文件

webpack.config.dev.js 开发环境下的配置项（yarn start）

webpack.config.prod.js 生产环境下的配置项（yarn build）

scripts 存放的是可执行脚本的JS文件

start.js yarn start执行的就是这个JS

build.js yarn build执行的就是这个JS

package.json中的内容也改了

【举个栗子：需要配置LESS】

$ yarn add less less-loader

less是开发和成产环境下都需要配置的

```

{

test: /\.(css|less)$/,

use: [

require.resolve('style-loader'),

...

{

loader: require.resolve('less-loader')

}

],

},

```

我们预览项目的时候，也是先基于webpack编译，把编译后的内容放到浏览器中运行，所以如果项目中使用了less，我们需要修改webpack配置项，在配置项中加入less的编译工作，这样后期预览项目，首先基于webpack把less编译为css，然后在呈现在页面中.

$ set HTTPS=true&&npm start 开启HTTPS协议模式（设置环境变量HTTPS的值）

$ set PORT=63341 修改端口号

====================================

**二. react & react-dom**

【渐进式框架】

一种最流行的框架设计思想，一般框架中都包含很多内容，这样导致框架的体积过于臃肿，拖慢加载的速度。真实项目中，我们使用一个框架，不一定用到所有的功能，此时我们应该把框架的功能进行拆分，用户想用什么，让其自己自由组合即可。

全家桶：渐进式框架N多部分的组合

VUE全家桶：vue-cli/vue/vue-router/vuex/axios(fetch)/vue element(vant)

REACT全家桶：create-react-app/react/react-dom/react-router/redux/react-redux/axios/ant/dva/saga/mobx...

1. react：REACT框架的核心部分，提供了Component类可以供我们进行组件开发，提供了钩子函数（生命周期函数：所有的生命周期函数都是基于回调函数完成的）

2. react-dom：把JSX语法（REACT独有的语法）渲染为真实DOM（能够放到页面中展示的结构都叫做真实的DOM）的组件

===========================

**三、把JSX（虚拟DOM）变为真实的DOM**

**参考JSX-源代码详细**