link: null
title: 珠峰架构师成长计划
description: bin\react-scripts2.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=83 sentences=266, words=1051

1.create-react-app工作流

```
create-react-app react-project

Creating a new React app in C:\aprepare\react-project.

Installing packages. This might take a couple of minutes.

Installing react, react-dom, and react-scripts with cra-template...

yarn add v1.22.10
info No lockfile found.

[1/4] Resolving packages...

[2/4] Fetching packages...

[3/4] Linking dependencies...

[4/4] Building fresh packages...

success Saved lockfile.

success Saved 881 new dependencies.

Done in 49.71s.

Removing template package using yarnpkg...

yarn remove v1.22.10
[1/2] Removing module cra-template...

[2/2] Regenerating lockfile and installing missing dependencies...

success Uninstalled packages.

Done in 10.19s.

Created git commit.

Success! Created react-project at C:\aprepare\react-project
Inside that directory, you can run several commands:
  yarn start
    Starts the development server.

  yarn build
    Bundles the app into static files for production.
```

2.初始化react-scripts2

```
mkdir react-scripts2
cd react-scripts2
npm init -y
yarn add react react-dom
yarn add cross-spawn fs-extra chalk webpack webpack-dev-server babel-loader babel-preset-react-app html-webpack-plugin  open
```

```
  "scripts": {
    "build": "node ./bin/react-scripts2 build",
    "start": "node ./bin/react-scripts2 start"
  },
  "bin": {
    "react-scripts2": "./bin/react-scripts2"
  },
```

```
npm run build
npm run start
```

3.实现build命令

- 1.设置 `process.env.NODE_ENV = 'production';`
- 2.获取 `webpack&#x914D;&#x7F6E;`文件
- 3.清空 `build`目录
- 4.拷贝 `public`目录下的文件到 `build`目录
- 5.创建 `compiler`并调用 `run`方法进行编译

bin\react-scripts2.js

```
const spawn = require('cross-spawn');
const args = process.argv.slice(2);
const script = args[0];
spawn.sync(
    process.execPath,
    [require.resolve('../scripts/' + script)],
    { stdio: 'inherit' }
);
```

scripts\build.js

```
process.env.NODE_ENV = 'production';
const chalk = require('chalk');
const fs = require('fs-extra');
const webpack = require('webpack');
const configFactory = require('../config/webpack.config');
const paths = require('../config/paths');
const config = configFactory('production');

fs.emptyDirSync(paths.appBuild);
copyPublicFolder();
build();

function build() {
    const compiler = webpack(config);
    compiler.run((err, stats) => {
        console.log(err);
        console.log(chalk.green('Compiled successfully.\n'));
    })
}

function copyPublicFolder() {
    fs.copySync(paths.appPublic, paths.appBuild, {
      filter: file => file !== paths.appHtml,
    });
}
```

config\paths.js

```
const path = require('path');
const appDirectory = process.cwd();
const resolveApp = relativePath => path.resolve(appDirectory, relativePath);
module.exports = {
    appHtml: resolveApp('public/index.html'),
    appIndexJs:resolveApp('src/index.js'),
    appBuild: resolveApp('build'),
    appPublic: resolveApp('public')
}
```

config\webpack.config.js

```
const paths = require('../config/paths');
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = function (webpackEnv) {
    const isEnvDevelopment = webpackEnv === 'development';
    const isEnvProduction = webpackEnv === 'production';
    return {
        mode: isEnvProduction ? 'production' : isEnvDevelopment && 'development',
        output: {
            path: paths.appBuild
        },
        module:{
            rules:[
                {
                    test: /\.(js|jsx|ts|tsx)$/,
                    include: paths.appSrc,
                    loader: require.resolve('babel-loader'),
                    options: {
                      presets: [
                        [
                          require.resolve('babel-preset-react-app')
                        ]
                      ]
                    }
                  },
            ]
        },
        plugins:[
            new HtmlWebpackPlugin({
                inject: true,
                template: paths.appHtml
            })
        ]
    }
}
```

src\index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
ReactDOM.render(<h1>helloh1>,document.getElementById('root'));
```

public\index.html

```

<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>reacttitle>
head>
<body>
    <div id="root">div>
body>
html>
```

4.实现start命令

scripts\start.js

```
process.env.NODE_ENV = 'development';
const configFactory = require('../config/webpack.config');
const createDevServerConfig = require('../config/webpackDevServer.config');
const WebpackDevServer = require('webpack-dev-server');
const DEFAULT_PORT = parseInt(process.env.PORT, 10) || 3000;
const HOST = process.env.HOST || '0.0.0.0';
const config = configFactory('development');
const webpack = require('webpack');
const chalk = require('chalk');
const compiler = createCompiler({ config, webpack });
const serverConfig = createDevServerConfig();
const devServer = new WebpackDevServer(compiler, serverConfig);
devServer.listen(DEFAULT_PORT, HOST, err => {
    if (err) {
        return console.log(err);
    }
    console.log(chalk.cyan('Starting the development server...\n'));
});

function createCompiler({ config, webpack }) {
    let compiler = webpack(config);
    return compiler;
}
```

config\webpackDevServer.config.js

```
module.exports = function () {
    return {
        hot: true
    }
};
```

5.知识

- 使用dotenv，只需要将程序的环境变量配置写在 `.env`文件中

.env

```
MONGODB_HOST=localhost
MONGODB_PORT=27017
MONGODB_DB=test
MONGODB_URI=mongodb:
```

```
const dotenvFile = '.env';
require('dotenv-expand')(
    require('dotenv').config({
        path: dotenvFile,
    })
);
console.log(process.env.MONGODB_HOST);
console.log(process.env.MONGODB_PORT);
console.log(process.env.MONGODB_DB);
console.log(process.env.MONGODB_URI);
```

- NODE_PATH 就是NODE中用来寻找模块所提供的路径注册环境变量

```
let fs = require('fs');
let path = require('path');
const appDirectory = fs.realpathSync(process.cwd());

process.env.NODE_PATH = [
    './a',
    path.resolve(appDirectory, 'b'),
    './c',
].join(path.delimiter);
process.env.NODE_PATH = (process.env.NODE_PATH || '')
  .split(path.delimiter)
  .filter(folder => folder && !path.isAbsolute(folder))
  .map(folder => path.resolve(appDirectory, folder))
  .join(path.delimiter);
console.log(process.env.NODE_PATH);
```

- 匹配semver版本的正则表达式

```
const semverRegex = require('semver-regex');
console.log(semverRegex().test('v1.0.0'));

console.log(semverRegex().test('1.2.3-alpha.10.beta.0+build.unicorn.rainbow'));

console.log(semverRegex().exec('unicorn 1.0.0 rainbow')[0]);

console.log('unicorn 1.0.0 and rainbow 2.1.3'.match(semverRegex()));
```

```
const semver = require('semver')
console.log(semver.valid('1.2.3'));
console.log(semver.valid('a.b.c'));
console.log(semver.clean('  =v1.2.3   '));
console.log(semver.satisfies('1.2.3', '1.x || >=2.5.0 || 5.0.0 - 7.2.3'));
console.log(semver.gt('1.2.3', '9.8.7'));
console.log(semver.lt('1.2.3', '9.8.7'));
console.log(semver.valid(semver.coerce('v2')));
console.log(semver.valid(semver.coerce('42.6.7.9.3-alpha')));
```

```
var globby = require('globby');
(async () => {
    const paths = await globby(['images','photos'],{
      expandDirectories: true
    });
    console.log(paths);
})();
```

- globalThis提供统一的机制来访问全局对象
    - Web 浏览器 window
    - Node.js global

- Web Worker self

```

console.log(globalThis);

console.log(globalThis);

console.log(globalThis);
```

```
const bfj = require('bfj');

bfj.read('big.json')
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.log(error)
  });
```

- 1.将依赖包的版本区间解析为某个具体的版本号
- 2.下载对应版本依赖的 tar 包到本地离线镜像
- 3.将依赖从离线镜像解压到本地缓存
- 4.将依赖从缓存拷贝到当前目录的 node_modules 目录
- PnP工作原理是作为把依赖从缓存拷贝到 node_modules 的替代方案

```
#create-react-app 已经集成了对 PnP 的支持。只需在创建项目时添加 --use-pnp 参数
npx create-react-app react-project --use-pnp
# 在已有项目中开启 PnP
yarn --pnp
yarn add uuid
```

- 只要 installConfig.pnp 的值是一个真值且当前版本的 Yarn 支持，PnP 特性就会被启用

```
{
  "installConfig": {
    "pnp": true
  }
}
```

uuid.js

```
let uuid = require('uuid');
console.log(uuid.v4());
```

- 由于在开启了 PnP 的项目中不再有 node_modules 目录，所有的依赖引用都必须由 .pnp.js 中的 resolver 处理
- 因此不论是执行 script 还是用 node 直接执行一个 JS 文件，都必须经由 Yarn 处理

```
yarn run build
yarn node uuid.js
```