# React & Angular

```
React & Angular
   Reading
   React简介和基础知识
      React是什么
      React 特点
      Virtual Dom
   JSX
   React组件基础
      组件分类
         函数组件
         类组件
      使用 React 组件
      State和Props
         Props
         State
         State和Props的对比
      React组件通信
   生命周期
      生命周期-挂载阶段
      生命周期 - 更新阶段
   Hooks
         函数组件和类组件的主要区别
   Redux
   Routing
   Redbook-ui
      创建项目
      项目结构解释
   React vs. Angular
      架构
      数据绑定
```

## Reading

- 主要看这个: https://www.yuque.com/fechaichai/qeamqf/xbai87#8d7af2df
- https://www.yuque.com/fairy-era/xurq2q/kffe18

## React简介和基础知识

## React是什么

React 是由 Facebook 开发并开源的一种用于构建用户界面的 JavaScript 库。它可以被单独用于开发<mark>单页面应用(SPA)</mark>,也可以和其他库或框架一起用于开发复杂的单页面应用。

#### 为什么要选择 React:

- 组件化: React 是基于组件的,这使得<mark>代码可以复用</mark>,提高了开发效率。
- 高效: React 通过 Virtual DOM(虚拟 DOM)实现了高效的 DOM 操作。
- 社区强大: React 有一个庞大且活跃的社区,有许多高质量的开源项目。
- 开放源码: React 的源代码完全开放, 使得我们可以深入到底层去理解其原理。

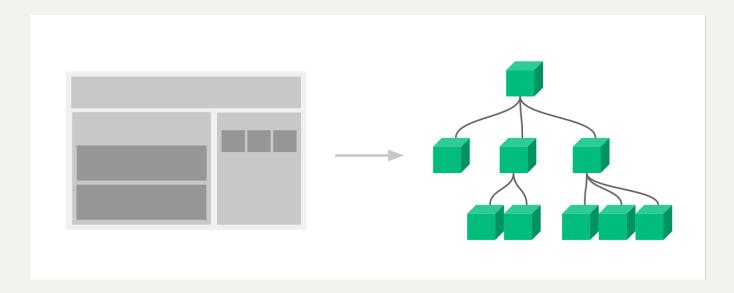
### React 特点

- 声明式UI (JSX)
  - <mark>写UI就和写普通的HTML一样</mark>,抛弃命令式的繁琐实现。
  - Question: 对比下用Javascript去操作dom来动态的输出redbook帖子的内容那个作业。
- 组件化

- <mark>组件是react中最重要的内容</mark>,组件可以通过搭积木的方式拼成一个完整的页面,通过组件的抽象可以增加复用能力和提高可维护性
- 跨平台
  - eact既可以开发web应用也可以使用同样的语法开发原生应用(reactnative),比如安卓和ios应用,甚至可以使用react开发VR应用,想象力空间十足,react更像是一个元框架 为各种领域赋能

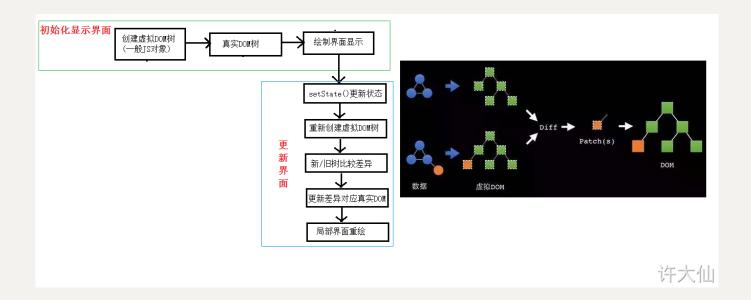






### Virtual Dom

https://www.yuque.com/fairy-era/xurq2q/ofzxib



Virtual DOM(虚拟DOM)是React中的一个核心概念。本质上,它是对真实DOM的抽象,是一种编程概念,其中的元素和组件以JavaScript对象的形式存在。当React组件的状态改变时,React会创建一个新的Virtual DOM,并与旧的Virtual DOM进行比较,来决定哪些部分需要在真实DOM中进行更改。这个过程被称为diffing。

React使用Virtual DOM的原因主要是因为在浏览器中操作DOM通常比较慢,而JavaScript运行速度快。React通过在JavaScript中进行大量操作,然后高效地、一次性地将变更应用到DOM上,以此来减少真实DOM的操作,达到优化性能的目的。

#### 以下是Virtual DOM的工作流程:

- 1. **Step 1** (**JSX**到**Virtual DOM**): 在React中,你编写的JSX代码会被转化为Virtual DOM。
- 2. **Step 2** (**Render**阶段): 当组件的状态改变,或者通过props接收到新的数据时, React会创建一个新的Virtual DOM树。这个新的树会与之前的树进行比较。
- 3. **Step 3** (**Diffing**阶段): React使用一种名为diffing的高效算法来比较新旧两棵 Virtual DOM树的差异。
- 4. **Step 4** (**Reconciliation**阶段): 在这一阶段, React会使用diffing算法的结果, 计算出最小的步骤来更新真实DOM。
- 5. **Step 5** (**Commit**阶段):最后,React会将计算出的结果应用到真实DOM。这个过程通常很快,因为它是一次性完成的,不需要多次访问DOM。

### **JSX**

https://www.yuque.com/fechaichai/qeamqf/xbai87#M1UW5

比较简单,看文章会更快。

### 优势:

- 1. <mark>采用类似于HTML的语法</mark>,降低学习成本,会HTML就会JSX
- 2. 充分利用JS自身的可编程能力创建HTML结构

注意: JSX 并不是标准的 JS 语法,是 JS 的语法扩展,浏览器默认是不识别的,脚手架中内置的 @babel/plugin-transform-react-jsx 包,用来解析该语法

### https://babeljs.io/

```
1 // JSX 语法
2 const element = <h1>Hello, world!</h1>;
3
4 // 编译后
5 const element = React.createElement(
6 'h1',
7 null,
8 'Hello, world!'
9 );
10 // javascript => 操作dom
11 // React in js, 该方法可以代替原始的js方式去操作dom. 类似于用jQuery操作dom.
```

```
BABEL
                                Docs Setup Try it out Videos Blog
                                                                     Q Search
                                                                                       Donate Team GitHub
  // 使用 React 创建 h1 标签
                                                         // 使用 React 创建 h1 标签
  const h1 = <h1 className="a">hello React</h1>
                                                          const h1 = /*#__PURE__*/ React.createElement(
                                                            "h1",
                                                                                             命令式
                                        声明式
  const list = (
                                                             className: "a"
                                                        5
      <div>
          "hello React"
           1
                                                        8);
           title="">1
                                                        9
                                                          const list = /*#__PURE__*/ React.createElement(
10
           1
                                                       10
                                                            "div",
11
           1
                                                       11
12
          12
                                                            /*#__PURE__*/ React.createElement(
                                                       13
13
      </div>
                                                             "ul",
14
                                                       14
15
                                                       15
                                                               className: "a"
                                                       16
                                                       17
                                                              /*#__PURE__*/ React.createElement("li", null, "1"),
                                                              /*#__PURE__*/ React.createElement(
                                                       18
                                                               "li",
                                                       19
                                                       2.0
                                                                 title: ""
                                                       21
                                                       22
                                                               },
                                                       23
                                                               "1"
                                                       24
                                                       25
                                                              /*#__PURE__*/ React.createElement("li", null, "1"),
                                                              /*#__PURE__*/ React.createElement("li", null, "1")
                                                       26
                                                       27
                                                       28);
                                                       29
```

## React组件基础

https://www.yuque.com/fechaichai/qeamqf/xbai87#2e8a558f

## 组件分类

### 函数组件

使用JS的函数(或箭头函数)创建的组件,就叫做函数组件

```
1 // 定义函数组件
2 function HelloFn () {
3 return <div>这是我的第一个函数组件!</div>
4 }
5
```

```
6 // 定义类组件
7 function App () {
8
    return (
9
       <div className="App">
       {/* 渲染函数组件 */}
10
11
       <HelloFn />
12
        <HelloFn></HelloFn>
     </div>
13
14
   )
15 }
16 export default App
```

#### 约定说明

- 1. 组件的名称必须首字母大写,react内部会根据这个来判断是组件还是普通的HTML标签
- 2. 函数组件必须有返回值,表示该组件的 UI 结构;如果不需要渲染任何内容,则返回 null
- 3. 组件就像 HTML 标签一样可以被渲染到页面中。组件表示的是一段结构内容,对于函数组件来说,渲染的内容是函数的返回值就是对应的内容
- 4. 使用函数名称作为组件标签名称,可以成对出现也可以自闭合

### 类组件

```
class Welcome extends React.Component {
constructor(props) {
    super(props);
    this.state = { name: 'Sara' };
}

render() {
    return <h1>Hello, {this.state.name}</h1>;
}

}
```

#### 约定说明

- 1. 类名称也必须以大写字母开头
- 2. 类组件应该继承 React.Component 父类,从而使用父类中提供的方法或属性
- 3. 类组件必须提供 render 方法render 方法必须有返回值,表示该组件的 UI 结构

## 使用 React 组件

```
const element = <Welcome name="Sara" />;

ReactDOM.render(
   element,
   document.getElementById('root')

);
```

通过用ReactDOM.render方法,将组件放进去,则就可以使用该组件。

### 还是直接过网站比较好,课堂上把重点黏贴过来

## State和Props

父往子传递信息,通过props。是readonly,不能改它里面的值。

state是存组件内部变量的

### **Props**

Props(属性)是 React 组件的输入。它们被父组件传递到子组件,并且在子组件中是不可变的。通过 Props,React 组件可以复用,并且可以自上而下地在应用中传递数据。

```
1 // 创建组件
2 function Welcome(props) {
3    // 在组件内使用 props
4    return <h1>Hello, {props.name}</h1>;
5 }
6
7 // 使用组件, 并传入 props
8 ReactDOM.render(<Welcome name="Sara" />,
    document.getElementById('root'));
9
```

### 可以把 与 进行对比。

在html标签中,src是img标签的属性。我们可以把图片网址传给s r c变量,img标签则会去获得该照片并显示出来。

Welcom组件中的name也可以认为是属性(properties)。该属性会被传递到Welcome的 props变量里。那么我们获得该信息,则可以用该信息去做一些业务逻辑。

比如welcome时候需要有一张照片。

```
function Welcome(props) {
2
    // 在组件内使用 props
3
    return <div><h1>Hello, {props.name}</h1> <img src=props.image>
   </div>;
5
  }
6
   // 使用组件,并传入 props
8 ReactDOM.render(<Welcome name="Sara" image="picture-url"/>,
   document.getElementById('root'));
10
       props = {
11
         name: "sara",
         image: "picture-url"
12
13
       }
```

#### **State**

State(状态)是 React 组件的内部数据,它可以改变,并且每次改变都会触发组件的重新 渲染。通过 State,我们可以创建具有交互性的 React 组件。

```
class Clock extends React.Component {
     constructor(props) {
2
3
       super(props);
       // 初始化 state
4
       this.state = { date: new Date() };
6
     }
7
8
     // 使用生命周期方法设置定时器
     componentDidMount() {
10
       this.timerID = setInterval(
11
12
          () => this.tick(),
         1000
13
14
       );
```

```
15
      }
16
17
      componentWillUnmount() {
18
        clearInterval(this.timerID);
      }
19
20
21
      // 更新 state
22
     tick() {
23
        this.setState({
24
          date: new Date()
25
       });
26
      }
27
28
      render() {
29
        return (
30
          <div>
31
            <h1>Hello, world!</h1>
32
            <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
          </div>
33
34
       );
35
    }
36 }
37
38 // 使用组件
39 ReactDOM.render(
40
    <Clock />,
41
     document.getElementById('root')
42
  );
43
```

### State和Props的对比

- 两者都可以决定组件的行为和渲染输出。
- Props 是组件的配置参数,通常用于传递父组件的数据和函数,是不可变的。

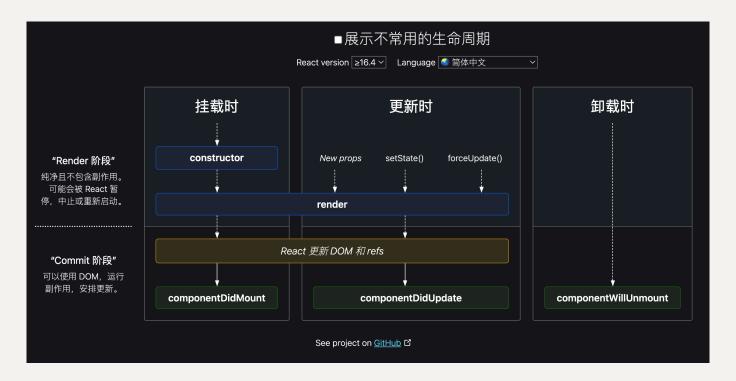
• State 是组件的内部数据,只能在组件内部使用和修改,可以随时间改变。

## React组件通信

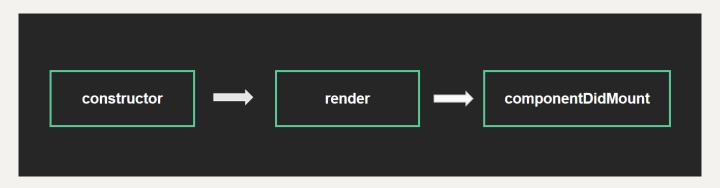
https://www.yuque.com/fechaichai/qeamqf/xbai87#1e20b61a

## 生命周期

- https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/
- https://www.yuque.com/fechaichai/qeamqf/xbai87#625cc340



## 生命周期-挂载阶段



No. 12 / 21

钩子 函数	触发时机	作用
constructor	创建组件时,最先执行,初始化 的时候只执行一次	1. 初始化state 2. 创建 Ref 3. 使 用 bind 解决 this 指向问题等
render	每次组件渲染都会触发	渲染UI(注意:不能在里面调用setState())
componentDidMount	组件挂载(完成DOM渲染)后 执行,初始化的时候执行一次	1. 发送网络请求 2.DOM操作

## 生命周期 - 更新阶段



钩子函数	触发时机	作用
componentWillUnmount	组件卸载(从页面中消	执行清理工作(比如:清理定时器
	失)	等)

## **Hooks**

Hooks的本质:一套能够使函数组件更强大,更灵活的"钩子"

## 函数组件和类组件的主要区别

前面说了组件有两大类,一类是函数组件,一类是class类组件。<mark>函数组件没有state和 props,所以功能有限。</mark>

hooks的加入,是为了让函数组件达到类组件的功能。使得组件既功能强大,又更加函数式编程。

函数组件是使用普通 JavaScript 函数定义的,接受一个 props 参数,并返回一个 React 元素。函数组件是状态(state)的"无状态"组件,这意味着他们不能访问 this.state 或使用生命周期方法。然而,随着 React Hooks 的引入,你现在可以在函数组件中使用状态和其他 React 特性。

#### 例如:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

类组件是使用 ES6 类定义的,这些类继承自 React.Component,并且至少定义一个 render 方法。类组件可以有本地状态(即可以使用 this.state),并且可以使用生命周期方法(如 componentDidMount 和 componentDidUpdate 等)。类组件还可以使用其他诸如事件处理等 React 特性。

#### 例如:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
}
```

### 主要区别

- 类组件提供更多的功能,如生命周期方法、默认 props 和显示名称等,这些功能在函数组件中是无法使用的。然而,使用 React Hooks,函数组件现在可以使用类似的状态管理和生命周期功能。
- 函数组件通常更简单和易于理解,没有 this 关键字,也没有生命周期方法。这使得源码更易于阅读和测试。
- 在类组件中,状态(state)和生命周期功能都是内置的,而在函数组件中,需要通过 React Hooks 来实现。

• 由于类组件需要继承 React.Component, 因此可能会有额外的性能开销, 但这通常在实际应用中可以忽略不计。

#### 剩下的看文章即可

- https://www.yuque.com/fechaichai/qeamqf/xbai87#1fce8dbf
- https://www.yuque.com/fechaichai/qeamqf/xbai87#9ef38bf4

```
import { useState } from 'react'
2
3 function App() {
     // 参数: 状态初始值比如,传入 0 表示该状态的初始值为 0
    // 返回值:数组,包含两个值:1 状态值(state) 2 修改该状态的函数
    (setState)
    const [count, setCount] = useState(0)
7
     constructor(props) {
       super(props);
       // 初始化 state
       this.state = { count: 0 };
10
11
     }
12
     const [date, setDate] = useState(new Date)
     const [address, setAddress] = useState({
13
      city: "guangzhou",
14
       zipcdoe: "2323"
15
16
     })
17
     return (
       <button onClick={() => { setCount(count + 1) }}>{count}
18
   </button>
19 )
20 }
21 export default App
```

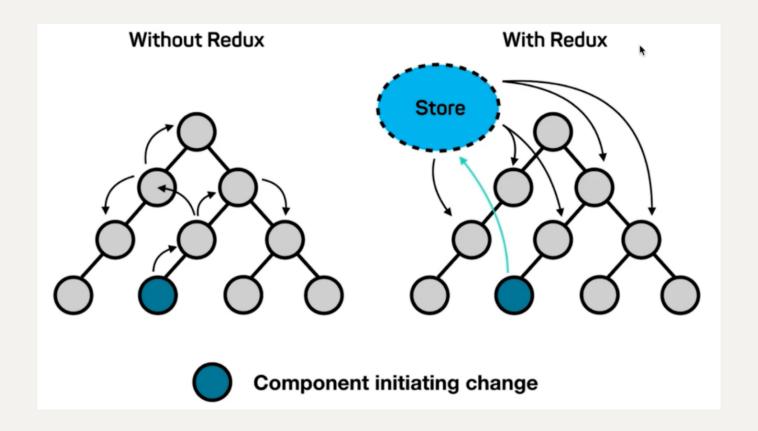
```
🤗 PostList.jsx 🗴
                                                                                ° PostList.jsx ×
REACT > project > redbook-ui-bootstrap > src > components > ∰ PostList.jsx > ❤ PostList
                                                                                                REACT > project > redbook-ui-bootstrap > src > components > hooks > ∰ PostList.jsx > [∅] F
      import React from 'react';
                                                                                                       import React, { useState, useEffect } from 'react';
       import Post from './Post'
                                                                                                        import Post from './Post';
       import { getPosts } from '../services/api';
                                                                                                        import Pagination from 'react-bootstrap/Pagination';
                                                                                                       import { getRandomData } from '../../services/fakeDataService';
                                                                                                       import { Container, Row, Col } from 'react-bootstrap';
  5
       class PostList extends React.Component {

    state = - {

  6
               posts: [],
                                                                                                        const PostList = () => {
  8
               currentPage: 0,
                                                                                                          const [posts, setPosts] = useState([]);
                                                                                                          const [currentPage, setCurrentPage] = useState(1);
const [totalPages, setTotalPages] = useState(1);
  9
               totalPages: 1,
 10
                                                                                                  10
 11
                                                                                                  11
 12
           componentDidMount() {
                                                                                                  12
                                                                                                          useEffect(() => {
 13
               this.fetchPosts(this.state.currentPage);
                                                                                                  13
                                                                                                            const fetchPostsData = async () => {
 14
                                                                                                  14
                                                                                                              try {
 15
                                                                                                  15
                                                                                                               const posts = await Array.from({ length: 100 }, () => getRandor
 16
           fetchPosts(page) {
                                                                                                  16
                                                                                                                setPosts(posts);
 17
               getPosts(page)
                                                                                                  17
                                                                                                                setTotalPages(10);
 18
                    .then(response \Longrightarrow {
                                                                                                  18
 19
                       this.setState({
                                                                                                  19
 20
                           posts: response.data,
                                                                                                  20
                                                                                                                // const response = await getRandomData(currentPage);
 21
                            totalPages: response.data.totalPages });
                                                                                                  21
                                                                                                                // setPosts(response.posts);
 22
                   3)
                                                                                                  22
                                                                                                                // setTotalPages(response.totalPages);
 23
                    .catch(error => {
                                                                                                  23
                                                                                                             } catch (error) {
                       console.error('Error fetching data', error);
                                                                                                               console.error(error);
 24
                                                                                                  24
 25
                                                                                                  25
 26
                                                                                                  26
 27
                                                                                                  27
 28
           componentDidUpdate() {
                                                                                                  28
                                                                                                            fetchPostsData():
 29
               alert("updated");
                                                                                                          }, [currentPage]);
 30
 31
                                                                                                  31
                                                                                                          const handlePageChange = (pageNumber) => {
           render() {
 32
                                                                                                  32
                                                                                                           setCurrentPage(pageNumber);
                                                                                                  33
 33
               return (
 34
                    <div>
 35
                        <div className="row">
                                                                                                  35
                                                                                                          const postGroups = [];
 36
                            {this.state.posts.map((post) => (
                                                                                                  36
                                                                                                          for (let i = 0; i < posts.length; i += 2) {
                                <div className="col-sm-6" key={post.id}</pre>
                                                                                                  37
 37
                                                                                                           postGroups.push(posts.slice(i, i + 2));
                                                                                                  38
                                    <Post post={post} />
 38
 40
                            ))}
                                                                                                  40
 41
                                                                                                  41
                                                                                                            <Container>
                        </div>
                                                                                                            {postGroups.map((group, index) => (
                                                                                                  42
```

## Redux

https://www.yuque.com/fechaichai/qeamqf/omg1xi



## Routing

https://www.yuque.com/fechaichai/qeamqf/smoknz

```
const routes = [
      {
2
        path:'/home',
        component: Home
5
      },
        path:'/about',
7
        component: About
8
9
      },
10
       {
        path:'/article',
11
12
        component: Article
13
      }
14
    ]
```

### Redbook-ui

 https://github.com/TAIsRich/chuwa-ui/tree/main/REACT/project/redbook-ui-bo otstrap

## 创建项目

你可以使用 create-react-app 命令来初始化一个基本的 React 项目结构。 create-react-app 是一个官方支持的创建单页面React应用的脚手架。以下是在终端或命令行界面运行的命令:

npx create-react-app redbook-ui

上述命令会创建一个名为 my-app 的新目录,并在该目录下生成一个新的 React 应用的初始项目结构。这会创建一个你在上面看到的基本项目结构。其中 my-app 是你应用的名字,你可以根据需要来替换它。

然后, 你可以通过以下命令进入这个新创建的目录:

1 cd my-app

在这个新创建的 React 应用中,你需要手动创建一些文件和文件夹,比如 components/和 services/文件夹,以及在这些文件夹下的 Comment.js、AddComment.js、Post.js 和 api.js 等文件。

在安装额外的库,比如 @material-ui/core 和 @material-ui/icons ,你需要在项目目录下运行以下命令:

1 npm install @material-ui/core @material-ui/icons

如果你想使用其他的命令行工具,比如 Yarn,你可以使用以下的命令来创建新的 React 应用:

1 yarn create react-app my-app

并使用以下的命令来安装额外的库:

```
yarn add @material-ui/core @material-ui/icons
```

## 后来我改成bootstrap了

## 项目结构解释

在一个标准的React项目中,常见的文件和目录结构如下:

```
my-app/
    — node modules/
    — public/
        — index.html
        - favicon.ico
        └─ manifest.json
6
      - src/
         — components/
9
            — Comment.js
            - AddComment.js
10
11
            - Post.js
            └─ App.js
12
          - services/
            L api.js
14
15
          - App.css
          - index.css
16
17
          - App.js
          - index.js
18
        reportWebVitals.js
19
20
       .gitignore
      - package.json
21
22
      - package-lock.json
```

- 1. node\_modules/文件夹包含了项目所有的依赖。- 类似于Java项目的third party jars.
- 2. public/文件夹包含了静态资源如 HTML 文件、图片等。
- 3. src/ 文件夹包含了项目的源代码。源代码一般都放在这个文件夹中。
  - components/文件夹包含了所有React组件,例如
    Comment.js、AddComment.js 和 Post.js。这些组件被用于构建用户界面。
  - services/文件夹包含了跟后端服务交互的代码,例如 api.js。
  - App.js 是应用的主入口。
  - index.js 是整个React应用的根渲染文件。
  - App.css 和 index.css 包含了应用的样式。
  - reportWebVitals.js 是一个用于报告页面性能的文件,由 create-react-app 自动生成。
- 4. gitignore 是一个 Git 文件,用于列出不需要加入版本控制的文件或文件夹。
- 5. package.json 是一个包含了项目依赖以及其他一些项目信息(如项目名称,版本等)的文件。类似于pom.xml
- 6. package-lock.json 是锁定安装时的包的版本号,并且需要上传到git,以确保其他人在npm install时大家的依赖能保证一致。
- 7. README.md 通常包含了关于项目的一些基本信息,比如如何运行和构建项目等。
- 8. yarn.lock Yarn的锁定文件,与package-lock.json类似,如果你使用的是yarn作为包管理器,那么就会有这个文件。

这只是一个常见的项目结构,并不是唯一的方式。根据项目的大小和复杂性,结构可能会有所不同。重要的是保持结构的清晰和一致,使得其他开发者可以快速理解和使用。

## React vs. Angular

## 架构

- React 是一个用于构建用户界面的 JavaScript 库。它关注于视图,你需要自己选择 其他库(如 Redux、MobX)进行状态管理、路由等。
- Angular 是一个完整的前端框架,包含了从模板到完全的组件生命周期管理和状态管理等。它使用了 TypeScript 作为开发语言。

## 数据绑定

- React 采用单向数据流。组件的状态是不可变的,当状态改变时,组件就会重新渲染。这使得状态管理更直观,但有时需要更多的代码来实现这种行为。
- Angular 则采用双向数据绑定。模型的改变可以立即反映在视图上,视图的改变也能立即反映在模型上。这可以减少代码量,但可能会导致数据流不太清晰。

## 学习曲线

- React 相对简单,API 相对较少,学习曲线相对平缓。React 社区提供了大量的开源库,但是选择哪一个开源库来配合 React 使用,这可能需要开发者投入时间去研究。
- Angular 作为一个完全的框架,提供了丰富的功能,因此有更陡峭的学习曲线。由于 Angular 提供了许多内置特性,因此在一些情况下可能需要深入理解 Angular 的工作方式。

### 性能

- React 提供了 Virtual DOM,并且可以通过 shouldComponentUpdate 这个生命周期方法进一步优化性能。
- Angular 有 change detection 机制,也可以进行一定程度的性能优化,但是需要更多的理解和配置。