

react基础

react并不是一个渐近式的框架，它是一个重量级的框架，它不能像vue一样，首先导入一个 `vue.js` 来看看效果，然后再导入 `vue-router` 来完成单页面，如果要数据请求再导入 `axios`，这些过程都是慢慢来的，但是react不行

react以前是可以这么做的，但是到 `react 16` 这个版本以后就摒弃了这种渐近式的方式直接以脚手架的方式运行

安装react的脚手架

```
$ npm install create-react-app -g
```

上面的命令就是安装react的脚手架，也简称安装cra

安装成功了以后在控制台输入 `create-react-app -v` 来查看版本

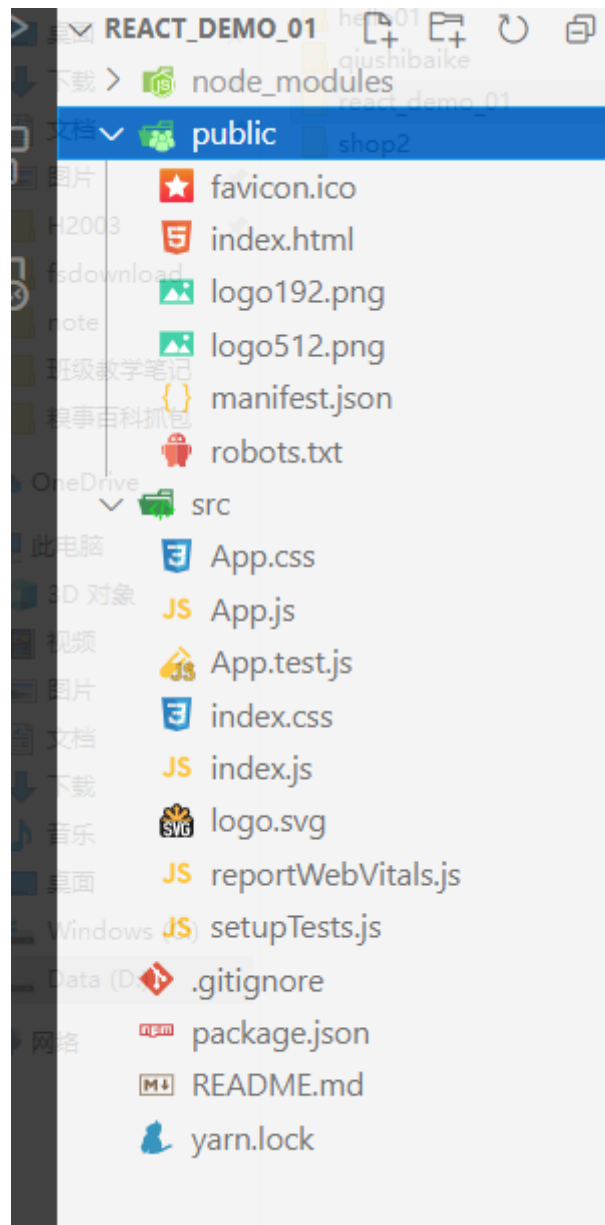
```
C:\Users\Administrator>create-react-app -V
4.0.3 安装成功了
```

通过脚手架去创建项目

通过刚刚安装的脚手架，我们可以去创建创建react的项目，命令如下

```
$ create-react-app 项目名称
$ create-react-app react_demo_01
```

react项目的结构



1. `public` 公开目录
2. `src` 源文件的目录，这个目录下面存放的都是开发文件
 - `index.js` 整个程序入口文件（也是webpack的入口文件，相当于vue里面的 `main.js` 文件）
 - `index.css` 全局样式，在这里的样式可以在全局使用
 - `App.js` 这是根组件【react里面的组件是以js文件的形式存在的，相当于vue里面的 `App.vue`】
 - `App.css` 是组件 `App.js` 的样式文件，相当于 `vue` 文件中的 `<style>`
 - `App.test.js` 是vue中做单元测试的时候使用【测试人员使用】
 - `setupTests.js` 测试人员在启动测试之前做配置的时候使用的
 - `reportWebVitals.js` 在开发过程当中如果出现错误会把提示信息展示在界面上
3. `package.json` 记录了当前项目的相关信息

react项目的启动

通过查看 `package.json` 里面的命令，我们看到有一个启动命令叫 `start`

```
$ npm run start
```

通过上面的命令我们可以启动的我们的界面

react项目的启动分析

App.js文件

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div>
      <h2>大家好啊，我是标哥</h2>
    </div>
  );
}

export default App;
```

上面的方法返回了一个标签，这是 `React` 的特殊语法格式叫 `jsx`

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App.js';
import reportWebVitals from './reportWebVitals';

//react启动了渲染函数
ReactDOM.render(
  <App></App>,
  document.getElementById('root')
);
```

在上面的代码里面的，`ReactDOM.render` 这个方法在 `id='root'` 这个的这个节点下面渲染了一个 `<APP />` 虚拟DOM，而 `<APP />` 就是我们刚刚 `App.js` 所生产的组件

所以我们可以理解为 `id='root'` 就相当于我们之前vue里面的托管区域

react基础语法讲解

react是以“一切皆组件”的原则来进行开发，它的组件又是以对象的形式存在，所以在React最初的时候它是借用于ES6里面的 `class` 来创建对象，然后将这个对象再继承 `React` 就可以了

`react` 组件内部使用的是 `jsx` 的语法，它是一种特殊的语法格式，我们现来学习

创建组件

组件的创建分两种方式

1. 函数式组件【新语法，React Hook Api】
2. class式组件【以前的语法】

我们现在以 `class` 式组件的试去创建

1. 第一步：先新建一个 `App.js` 的文件
2. 第二步：在 `App.js` 的这个文件内部导入 `react`
3. 第三步：通过 `class` 方式的方式创建一个对象

- 第四步：导出这个 `class` 的对象
- 第五步：重写内部的 `render` 方式

通过上面的步骤以后我们就可以得到下面的代码

```
import React from "react";

class App extends React.Component{
  // 一定要重写这个方法，这个方法所返回的内容就是这个组件的内容，它可以直接返回HTML标签
  // 你们可以认为这个返回的内容相当于vue中的`template`，所以它只能有一个根标签
  render(){
    return <div>
      这是我的第一个组件，也是根组件
    </div>
  }
}

export default App;
```

在上面的代码里面，我们可以看一点，在 JavaScript 的代码里面，我们又写入了 HTML 的代码，这样一种语法格式是 React 的特点，它种语法叫 `jsx` 语法

`render`方法返回的内容将会是这个组件最终所显示的内容

注意事项：

1. 组件的名称首字母必须是大写，并且必须是默认导出，所以上面的App并且是大写
2. `render`方法必须重写，并且一定要的返回的内容
3. 返回的内容不能有片段代码，如果有片段代码，只能使用 `React.Fragment` 包裹

组件的数据

当我们创建好组件以后，组件的内部是有状态的，这个状态可以理解为这个组件所需要的数据【组件的数据也称之为组件的状态】

```
import React from "react";

class App extends React.Component {
  constructor() {
    super(...arguments);
    //我们在这里构建组件内部所需要的数据
    this.state = {
      userName: "张珊"
    }
  }
  // 一定要重写这个方法，这个方法所返回的内容就是这个组件的内容，它可以直接返回HTML标签
  // 你们可以认为这个返回的内容相当于vue中的`template`，所以它只能有一个根标签
  render() {
    return <React.Fragment>
      <div>
        Hello world!
      </div>
    </React.Fragment>
  }
}

export default App;
```

在上面的 `state` 里面，我们已经完成了最初的值

数据渲染

普通数据渲染

```
<h2>大家好， 我是{this.state.userName}</h2>
<h2>这里有一个数据是{Math.abs(this.state.num)}</h2>
<h1>{this.state.num > 0 ? "正数" : "负数"}</h1>
```

直接在html的标签内部使用 `{}` 即可，在 `jsx` 语法里面 `{}` 里面的代码代表的就是js代码

条件渲染

在vue里面，我们可以通过 `v-if` 去实现条件渲染，在微信小程序里面我们可以通过 `wx:if` 去实现条件渲染，在 `react` 当中也可以实现这样的功能

```
{this.state.flag && <h2>大家好，这个东西可以显示，也可以不显示</h2>}
{this.state.isGirl ? <h2>女生</h2> : <h1>男生</h1>}
```

上面这种方式是一种简单的条件渲染方式，我们还可以实现更为复杂的条件渲染

```
renderAge(age){
  if(age<18){
    return <h2>你还没有成年</h2>
  }
  else if(age<40){
    return <h2>你已成年</h2>
  }
  else{
    return <h2>你已经是个大叔了</h2>
  }
}
render(){
  return <React.Fragment>
    {this.renderAge(this.state.age)}
  </React.Fragment>
}
```

列表渲染

在vue里面可以通过 `v-for` 来进行列表的数据渲染，小程序里面则是 `wx:for` 来进行，在 `react` 的 `jsx` 语法里面，也是可以的

```
<ul>
  {this.state.stuList.map((item, index) => {
    return <li key={index}>{item}</li>
  })}
</ul>
<ul>
  {this.state.stuList.map((item, index) => <li key={index}>{item}</li>)}
</ul>
```

上面的方法是调用了数组当中的 `map` 来完成了数据渲染的功能，**请注意这里的箭头的使用方法，有 `return` 及没有 `return` 的区别，有欧花括号与没有花括号的区别**

对于复杂的数据渲染，我们可以单独的写一个方法，如下

```
renderStuList(stuList) {  
  let results = [];  
  stuList.forEach((item, index) => {  
    if (item !== "李四") {  
      results.push(<li key={index}>{item}</li>);  
    }  
  });  
  return results;  
}
```

然后再渲染的时候再调用方法就行了，如下

```
<ul>  
  {this.renderStuList(this.state.stuList)}  
</ul>
```

样式渲染

在react里面，怎么使用样式呢？样式的使用无非就是 `class` 或 `style`，但是这两个都与之前的有区别

P02.css

```
.box{  
  width: 100px;  
  height: 100px;  
  background-color: deeppink;  
}
```

P02.js

```
import React from "react";  
import "./P02.css";  
class P02 extends React.Component{  
  render(){  
    return <React.Fragment>  
      <div className="box"></div>  
    </React.Fragment>  
  }  
}  
export default P02;
```

上面我们在js文件里面直接导入了 `css` 文件，这是可行的，这是最基本的使用方法，这种方式导入以后，它是全局的使用方式

此处情况对于我们单页面开发来说是很麻烦的，因为class样式容易混淆。在vue当中解决这个问题非常简单

```

<template>

</template>
<script>

</script>
<style scoped>
  /*在vue里面只用添加一个scoped即可*/
</style>

```

在 `React` 当中，它没有 `scoped`，但是又想实现 `css` 的模块化，怎么办呢？

模块化的样式

在 `react` 的 `cra` 当中，如果要实现模块化的 `CSS`，则可以通过下面的方式来进行

1. 新建一个样式文件，以 `module.css` 结尾，如 `P02.module.css`
2. 在里面编写样式文件

```

/* 这个时候这个CSS文件是以module为名的，react就会认为它是一个块有模块化的css */
.loginBox{
  width: 200px;
  height: 200px;
  background-color: blue;
}

```

3. 在 `js` 的组件当中去导入这个样式文件

```

import React from "react";
import "./P02.css";
// 导入模块化的CSS
import p02Style from "./P02.module.css";

class P02 extends React.Component{
  render(){
    return <React.Fragment>
      <div className="box"></div>
      <div className={p02Style.loginBox}></div>
    </React.Fragment>
  }
}
export default P02;

```

导入的时候会到一个变量，这个变量可以认为是一个 `CSS` 对象

4. 得到导入的模块对象以后，再调用和 `class` 名称即可

```

<div className={p02Style.loginBox}></div>

```

通过上面的方式，我们就可以让 `CSS` 样式实现模块化了，最终在页面上面渲染的时候如下

```

▼ <div id="root">
  <div class="box"></div>
  <div class="P02_loginBox__2iHon"></div> == $0
  <hr>
  <div class="box"></div>

```

style动态样式的设置

在vue里面可以通过 `style` 去设置动态样式，`react` 里面也是可以的

在 `react` 里面，`style` 不能像以前一样直接书写，否则报错，下面就是错误的演示

```
<div style="border:1px solid red;height:100px"></div>
```

如果非要使用 `style` 则只能通过下面的方式来进行

```

import React from "react";
import "./P02.css";
// 导入模块化的CSS
import p02Style from "./P02.module.css";

class P02 extends React.Component {
  constructor() {
    super(...arguments);
    this.state = {
      //在这里先定义一个css对象
      cssObj: {
        border: "1px solid red",
        height: "200px",
        width: "200px",
        marginTop: "50px"
      }
    }
  }
  render() {
    return <React.Fragment>
      <div className="box"></div>
      <div className={p02Style.loginBox}></div>
      { /*在这里在通过这种方式使用刚刚定义的对象*/ }
      <div style={this.state.cssObj}></div>
    </React.Fragment>
  }
}
export default P02;

```

通过上面的方式，我们确实可以使用 `style` 去完成，但是有个点不好，我们不能可能每次使用 `style` 都要单独去构建一个对象这个时候，我们可以向下面这种方式去完成

```

<div style={{border:"1px solid red",color:"blue",height:"200px"}}>
  我是通过直接写的方式来的
</div>

```

react组件及组件化开发

在 react 里面一切皆主键，所以我们可以把所有的东西都看成是组件，那么 react 是如何实现组件化开发的？组件化开发的过程当中必然有组件传值，组件插槽，组件的数据流以及react独有的HOC高阶组件

之前我们已经学习过了组件的创建过程，组件目前是基于了class存在的，然后再去继承 `React.Component` 即可

PageHeader.js组件的代码

```
import React from "react";
import pageHeaderStyle from "../PageHeader.module.css";

class PageHeader extends React.Component {
  render() {
    return <div className={pageHeaderStyle.header}>
      {this.props.showBack === true && <div className=
{pageHeaderStyle.leftBack}>
        返回
      </div>}
      {this.props.title}
      <div className={pageHeaderStyle.rightMenu}>
        {this.props.children}
      </div>
    </div>
  }
}
export default PageHeader;
```

P03.js的代码

```
import React from "react";
import PageHeader from "../../components/PageHeader/PageHeader";

class P03 extends React.Component{
  constructor(){
    super(...arguments);
    this.state = {
      title:"标哥的APP"
    }
  }
  render(){
    return <React.Fragment>
      { /* 我现在就想在这里创建一个组件，然后引入 ，然后传值，怎么办呢？ */ }
      <PageHeader title={this.state.title} showBack={true}>
        <span>收藏</span>
      </PageHeader>
    </React.Fragment>
  }
}
export default P03;
```

在上面的代码当中， `P03.js` 是引用了 `PageHeader` 这个组件，同时向 `PageHeader` 这个组件内部传了值，也使用了插槽。这个我们发现它有一些相同点，也有一些不同点

代码分析：

1. 组件的传值仍然使用的是自定义属性，如 `title` 与 `showBack`

2. 组件内部在接收值的传递过来的数据的时候也使用了 `props`，如 `this.props.title` 和 `this.props.showBack`

上面的两个都是相同点，不同点就在于下面

3. `showBack` 它是一个驼峰命名的，在 `jsx` 的语法城面它不用转义【以前在html与vue里面，是要转义成 `show-back` 的，现在在react的jsx里面则不用转义了】
4. 在 `vue` 当中如果使用插槽则必须要使用 `<slot>`，而 `react` 里面直接使用 `this.props.children`

在上面的代码当中我们可以看到 `react` 没有具名插槽的概念，是因为 `react` 根据就不需要这个概念，因为 `react` 是有 `jsx` 语法的

```
<PageHeader title={<a href="#">{this.state.title}</a>} showBack={true}>
  <button type="button">收藏</button>
</PageHeader>
```

react组件的props类型限制

在进行组件传值的时候，我们是可以通过自定义属性的，这点与 `vue` 及微信小程序是保持一致的，但是在 `vue` 与微信小程序里面是可以做 `props` 的类型限定的，`react` 同样也可以

第一步：先导入类型的包prop-types，这个包是 `react` 自带的【或者说是cra自带的，如果没有就安装】

```
//这个包就是用于做prop类型的检测的
import PropTypes from "prop-types";
```

第二步：在导出刚刚创建的组件之前进行设置prop的类型设置

```
class PageHeader extends React.Component{
  //省略部分代码
}
//在导出这个组件之前，先做一次类型限定
PageHeader.propTypes = {
  showBack: PropTypes.bool,
  title: PropTypes.object.isRequired
}

export default PageHeader;
```

react中的事件方法

在 `react` 当中也是有事件，只是它的事件名比较特殊，绑定的方式也比较特殊

```
import React from "react";

class P04 extends React.Component{
  constructor(){
    super(...arguments);
    this.state={
      userName: "张三"
    }
  }
  sayHello(){
    console.log("你好啊，这是一个方法")
  }
}
```

```

    }
    render(){
      return <div>
        <button type="button" onClick={this.sayHello}>按钮1</button>
      </div>
    }
  }
}

export default P04;

```

上面就是一种最简单的方式绑定

1. `onClick` 使用了驼峰命令，这是必须的【在以前的html里面，它是全小写】
2. 绑定事件的时候不能加括号，只能是一个方法名【这里千万不能加括号，加了括号就直接执行了】

这是一种最简单的方式，这种方式的绑定其实是有问题的，主要的点就在于 `this` 关键字的使用

如果使用上面的方式去绑定的时候，我是拿不到 `this`，而 `this` 以应该是指向当前对象的，所以如果我们把 `sayHello` 的代码改成如下代码

```

sayHello(){
  console.log("你好啊，这是一个方法");
  console.log(this.state.userName);
}

```

这个时候点击就会报错，因为在 `react` 里面 `this` 绑定事件以后是 `undefined`

所以为了解决这个问题，我们需要传递 `this` 过去

```

<button type="button" onClick={this.sayHello.bind(this)}>按钮2</button>

```

小技巧：对于一些简单的事件方法，我们可以直接写成箭头函数的匿名函数

```

<button type="button" onClick={(event) => {
  console.log("我是一个箭头函数");
  console.log(this.state.userName);
}}>按钮3</button>

```

动态样式渲染

最基本的动态渲染

以下是样式代码

```

.topNav{
  display: flex;
  height: 40px;
  border-bottom: 2px solid #ececec;
  justify-content: space-around;
  align-items: center;
  margin: 0;
  padding: 0;
  list-style-type: none;
}
.topNav>li.selected{
  color: red;
}

```

```
font-weight: bold;
}
```

以下是js代码

```
import React from "react";
import p04style from "./P04.module.css";

class P04 extends React.Component {
  constructor() {
    super(...arguments);
    this.state = {
      typeList: ["推荐", "图片", "视频", "糗事"],
      pageType: 0
    }
  }

  changePageType(index){
    // 在这里把index赋值给state
    // 自动渲染 与 主动渲染
    // react执行的是主动渲染
    // this.state.pageType = index; 这种方式只会赋值，不会渲染
    this.setState({
      pageType:index
    });
  }

  render() {
    return <div>

      <ul className={p04style.topNav}>
        {this.state.typeList.map((item, index) => <li
          key={index}
          className={index == this.state.pageType ? p04style.selected
: null}

          onClick={this.changePageType.bind(this,index)}

          >
            {item}
          </li>)}
      </ul>
    </div>
  }
}

export default P04;
```

代码分析：

1. react 执行也是主动渲染模式，如果不调用 `setState` 内部的状态不是会变化的，状态不变页面则不会更新数据，如果要主动渲染则调用 `setState` 进行赋值
2. 在 react 里面，做动态样式绑定的时候，可以使用 `className={index == this.state.pageType ? p04style.selected : null}` 条件达式来完成

存在的问题

如果模块化的样式与最基本的样式进行叠加的时候一定要使用下面的方式去完成

第一种方法

```
<ul className={p04Style.topNav}>
  {this.state.typeList.map((item, index) => <li
    key={index}
    className={index === this.state.pageType ? p04Style.selected
: null, "iconfont", "iconhome"}.join(" ")}
    onClick={this.changePageType.bind(this, index)}
  >
    {item}
  </li>)}
</ul>
```

第二种方法

```
<ul className={p04Style.topNav}>
  {this.state.typeList.map((item, index) => <li
    key={index}
    className={` ${index === this.state.pageType ?
p04Style.selected : null} iconfont iconhome`}
    onClick={this.changePageType.bind(this, index)}
  >
    {item}
  </li>)}
</ul>
```

跨级组件的数据传递context

在之前vue里面，我们都说过组件内部的数据来源于 `data/computed/props/vuex` 这4个地方

在react里面，它的数据数据主要来源于 `state/props/context/redux` 这4个地方

state是当前组件的状态，props则是父级接收的值，`context` 称之为上下文对象，`redux` 称之为全局状态管理

在react如果父子级件需要传值，则使用 `props`，如果这个组件是下文级的关系（例如爷爷与孙子）这个时候再使用 `props` 传递就比较麻烦了。在这种情况下，我们可以使用 `react` 提供的context来进行传值

My.js里面的代码

```
import React from "react";
import Son from "../Son/Son";
// const ctx = React.createContext();
// 一般情况下，我们会直接这个上下文对象进行解构，解构出 生产者 与 消费者
// Provider称之为生产者，Consumer称之为消费者
export const { Provider, Consumer } = React.createContext();

class My extends React.Component {
  constructor() {
    super(...arguments);
    this.state = {
      tag: "我是明朝的皇帝"
    }
  }
}
```

```

    }
  }
  render() {
    return <Provider value={this.state.tag}>
      <div >
        我是标哥哥
        <button type="button" onClick={event => {
          this.setState({
            tag: "我是乞丐"
          });
        }}>我要改变tag的值</button>
        <hr />
        我本来有一个值 ， 我的值是{this.state.tag}
        <Son></Son>
      </div>
    </Provider>
  }
}

export default My;

```

Son.js里面的代码

```

import React from "react";
import GrandSon from "../GrandSon/GrandSon";

class Son extends React.Component{
  render(){
    return <div >
      我是儿子
      <GrandSon></GrandSon>
    </div>
  }
}

export default Son;

```

GrandSon.js里面的代码

```

import React from "react";
// 导入My里面的提供的消费者
import { Consumer } from "../My/My";

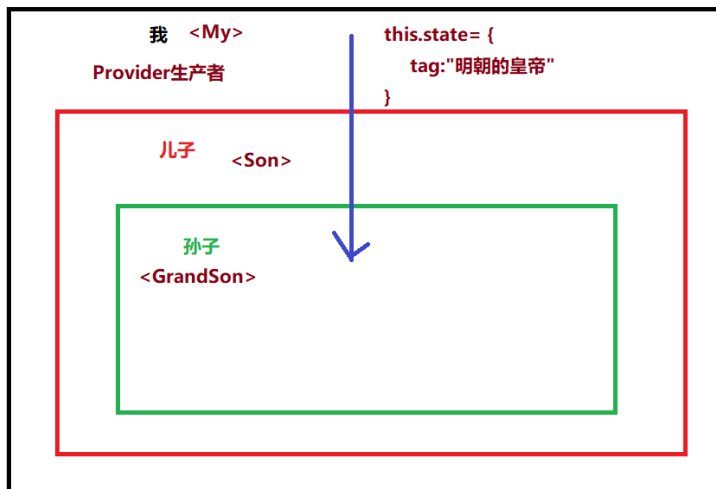
class GrandSon extends React.Component {
  render() {
    return <div >
      我是孙子
      <hr />
      <Consumer>
        {value => {
          return <h2>我从我爷爷那里拿到的值是: {value}</h2>
        }}
      </Consumer>
    </div>
  }
}

```

```
}

export default GrandSon;
```

上面我们使用了3个组件，分别是 `My` 里面包含了 `Son`，然后 `Son` 里面又包含了 `GrandSon`。结构如下图



在同一个东西里面，我们可以理解为同一个上下文context
在react里面，有这个关系，同一个上下文下面是可以共享数据的

`context` 的传递当中主要有两个点

1. `Provider` 生产者，主要是用于提供数据，用于数据的传递方
2. `Consumer` 消费者，主要用于接收数据，用于数据的接收方

通过一种你这样的方式，我们可以实现跨级组件的传递方式

react当中的refs操作DOM

在vue当中我们可以通过 `ref` 来操作某一个组件，在 `react` 里面也是一样的原理

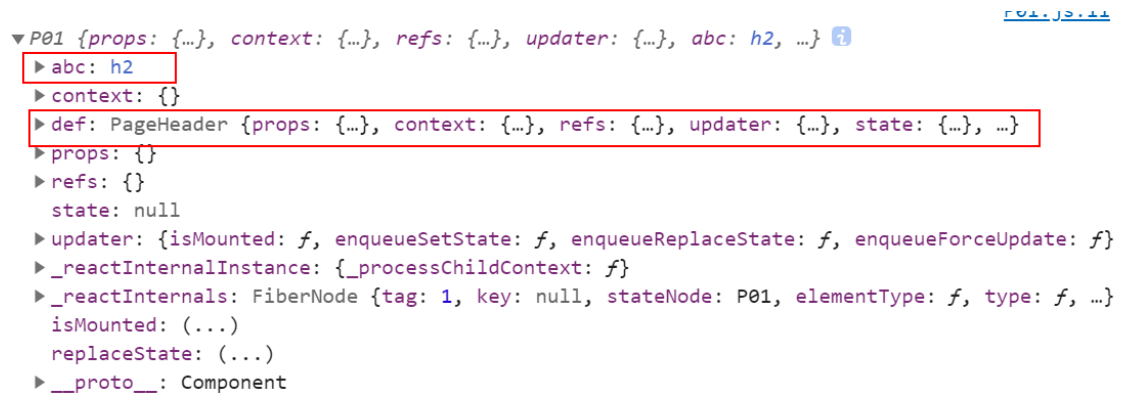
```
import React from "react";
import PageHeader from "../components/PageHeader/PageHeader";

class P01 extends React.Component {
  constructor() {
    super(...arguments);
    this.abc = null;
  }
  sayHello() {
    console.log(this);
  }

  render() {
    return <div>
      <h2 ref={ref => this.abc = ref}>这是一个2号标题</h2>
      <button type="button" onClick={this.sayHello.bind(this)}>我要获取DOM元素</button>
      <PageHeader ref={ref => this.def = ref}></PageHeader>
      <button type="button" onClick={this.sayHello.bind(this)}>获取组件virtualDOM的ref</button>
    </div>
  }
}
```

```
export default P01;
```

切记还是不能使用 `document` 来获取DOM元素，因为它有可能是虚拟DOM，所以我们可以看到在 `PageHeader` 里面我们也使用到了 `ref` 去获取到它



```
▼ P01 {props: {...}, context: {...}, refs: {...}, updater: {...}, abc: h2, ...} ⓘ  
  ▶ abc: h2  
  ▶ context: {}  
  ▶ def: PageHeader {props: {...}, context: {...}, refs: {...}, updater: {...}, state: {...}, ...}  
    ▶ props: {}  
    ▶ refs: {}  
    state: null  
    ▶ updater: {isMounted: f, enqueueSetState: f, enqueueReplaceState: f, enqueueForceUpdate: f}  
    ▶ _reactInternalInstance: {_processChildContext: f}  
    ▶ _reactInternals: FiberNode {tag: 1, key: null, stateNode: P01, elementType: f, type: f, ...}  
      isMounted: (...)  
      replaceState: (...)  
    ▶ __proto__: Component
```

通过上图我们可以看到 `abc` 是一个真实的DOM，而 `def` 则是一个虚拟DOM `PageHeader`