

React入门



React入门

课堂目标

资源

起步

上手

文件结构

React和ReactDOM

JSX

State和setState

Props属性传递

JSX实质

条件渲染和循环

class VS 函数组件

事件监听

组件通信

大名鼎鼎的虚拟DOM

生命周期

后续展望

回顾

课堂目标

1. 学习react基础语法
2. 熟悉官方create-react-app脚手架
3. 掌握JSX语法
4. 掌握setState
5. 掌握react生命周期
6. 掌握props传递参数
7. 掌握React组件通信

##

资源

1. [react](#)
2. [create-react-app](#)

起步

上手

1. `npm install -g create-react-app` 安装官方脚手架
2. `create-react-app react01` 初始化
3. react的api比较少，基本学一次，就再也不用看文档了，核心就是js的功力
4. [demo体验](#)

文件结构

├─ README.md	文档
├─ package-lock.json	
├─ package.json	npm 依赖
├─ public	静态资源
│ ├─ favicon.ico	
│ ├─ index.html	
│ └─ manifest.json	
└─ src	源码
├─ App.css	
├─ App.js	根组件
├─ App.test.js	测试
├─ index.css	全局样式
├─ index.js	入口
├─ logo.svg	
└─ serviceworker.js	pwa支持

React和ReactDOM

删除src下面所有代码，新建index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App />, document.querySelector('#root'));
```

新建App.js

```
import React from 'react'

class KaikebaCart extends React.Component{
  render(){
    return <div>
      <button>雷猴啊</button>
    </div>
  }
}

export default KaikebaCart
```

React设计之初，就是使用JSX来描述UI，所以解耦和dom操作，react只做逻辑层，reactDom去渲染实际的dom，如果换到移动端，就用别的渲染库

JSX

上面的代码会有一些困惑的地方，首先就是JSX的语法

```
ReactDOM.render(<App />, document.querySelector('#root'));
```

看起来是js和html的混合体，被称之为JSX，实际核心的逻辑完全是js实现的

State和setState

在APP组件里，我们可以通过{}在jsx中渲染变量

```
import React from 'react'

class KaikebaCart extends React.Component{
  render(){
    const name = '开课吧'
    return <div>
      <button>{name}</button>
    </div>
  }
}

export default KaikebaCart
```

如果数据需要修改，并且同时页面响应变化，我们需要放在state中，并且使用setState来修改数据

```
import React from 'react'

class KaikebaCart extends React.Component{
  constructor(props){
    super(props)
    this.state={
      name: '开课吧'
    }
  }
}
```

```
    setTimeout(()=>{
      this.setState({
        name: 'react真不错'
      })
    }, 2000)
  }
  render(){
    return <div>
      <button>{this.state.name}</button>
    </div>
  }
}

export default KaikebaCart
```

我们不能使用 `this.state.name='react'` 而是要用`this.setState`,设置一个新的state, 对数据进行覆盖

Props属性传递

```
ReactDOM.render(<App title="开课吧真不错" />, document.querySelector('#root'));

...

<h2>{this.props.title}</h2>
```

JSX实质

jsx实质就是`React.createElement`的调用

```

class App extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}, I am {2 + 2} years old
      </div>
    )
  }
}

ReactDOM.render(
  <App name="React" />,
  mountNode
)

```

REACT 编辑器

显示 JSX

```

class App extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name,
      ", I am ",
      2 + 2,
      " years old"
    )
  }
}

ReactDOM.render(React.createElement(App, { name: "React" }),
  mountNode)

```

条件渲染和循环

React的api不多，条件渲染和循环，都是普通的js语法

```
import React from 'react'
```

```

class KaikebaCart extends React.Component{
  constructor(props){
    super(props)
    this.state={
      name: '开课吧',
      showTitle:true,
      goods: [
        { text: '百万年薪架构师', price: 100, id:1 },
        { text: 'web全栈架构师', price: 80 ,id:2},
        { text: 'Python爬虫', price: 60 ,id:3}
      ]
    }
    setTimeout(()=>{
      this.setState({
        showTitle:false
      })
    },2000)
  }
  render(){
    return <div>
      {this.state.showTitle && <h2>{this.props.title}</h2>}
      <ul>

        {this.state.goods.map(good=>{
          return <li key={good.id}>
            <span>{good.text}</span>
            <span>{good.price}</span>元

          </li>
        })}

        </ul>
        <button>{this.state.name}</button>
      </div>
    }
  }
}

export default KaikebaCart

```

class VS 函数组件

如果一个组件只根据props渲染页面，没有内部的state，我们完全可以用函数组件的形式来实现(hooks的到来 会改变这个现状)

```

function Title({title}){
  return <h2>{title}</h2>
}
<Title title={this.props.title}></Title>

```

###

事件监听

React中使用onClick类似的写法来监听事件，注意this绑定问题 react里严格遵循单项数据流，没有数据双向绑定，所以输入框要设置value和onChange

```
handleChange(e){
  this.setState({
    name:e.target.value
  })
}

// 写法1 箭头函数自动修正this
<input
  type="text"
  value={this.state.name}
  onChange={(e)=>this.handleChange(e)}
/>
// 写法2 需要在构造函数里手动绑定this，否则会报错
<input
  type="text"
  value={this.state.name}
  onChange={this.handleChange}
/>
this.handleChange = this.handleChange.bind(this)

// 写法3
handleChange=(e)=>{
  this.setState({
    name:e.target.value
  })
}
```

组件通信

做个小购物车

```
import React from 'react'
import Cart from './Cart.js'
function Title({title}){
  return <h2>{title}</h2>
}
class KaikebaCart extends React.Component{
  constructor(props){
```

```

super(props)
this.state={
  name: '开课吧',
  showTitle:true,
  goods: [
    { text: '百万年薪架构师', price: 100, id:1 },
    { text: 'web全栈架构师', price: 80 ,id:2},
    { text: 'Python爬虫', price: 60 ,id:3}
  ],
  cart:[]
}
setTimeout(()=>{
  this.setState({
    showTitle:false
  })
},2000)
}
handleClick(i){
  const good = this.state.goods[i]
  let cartIndex
  const cartGood = this.state.cart.find((v,index)=>{
    if(v.text==good.text){
      cartIndex = index
      return true
    }
  })
  console.log(cartGood)
  if(cartGood){
    const newCart = [...this.state.cart]
    console.log(123,i,newCart)
    newCart[cartIndex].count += 1
    this.setState({
      cart :newCart
    })
  }else{
    this.setState({
      cart:[...this.state.cart, {
        text:good.text,
        price: good.price,
        active: true,
        count:1
      }]
    })
  }
}
handleChange=(e)=>{
  this.setState({
    name:e.target.value
  })
}
render(){

```



```

return <div>
  {this.state.showTitle && <Title title={this.props.title}></Title>}
  <div>
    <input
      type="text"
      value={this.state.name}
      onChange={this.handleChange}
    />
  </div>
</div>

  {this.state.goods.map((good,i)=>{
    return <li key={good.id}>
      <span>{good.text}</span>
      <span>{good.price}</span>元
      <button onClick={()=>this.handleClick(i)}>添加购物车</button>
    </li>
  })}

  </div>
  <button>{this.state.name}</button>
  <hr/>
  <Cart data={this.state.cart}> </Cart>
</div>
}
}

export default KaikebaCart

```

```

import React from 'react'
import Cart from './Cart.js'
function Title({title}){
  return <h2>{title}</h2>
}
class KaikebaCart extends React.Component{
  constructor(props){
    super(props)
    this.state={
      name: '开课吧',
      showTitle:true,
      goods: [
        { text: '百万年薪架构师', price: 100, id:1 },
        { text: 'web全栈架构师', price: 80 ,id:2},
        { text: 'Python爬虫', price: 60 ,id:3}
      ],
      cart:[]
    }
    setTimeout(()=>{
      this.setState({
        showTitle:false
      })
    },2000)
  }
}

```

```

handleClick(i){
  const good = this.state.goods[i]
  let cartIndex
  const cartGood = this.state.cart.find((v, index)=>{
    if(v.text==good.text){
      cartIndex = index
      return true
    }
  })
  console.log(cartGood)
  if(cartGood){

    const newCart = [...this.state.cart]
    console.log(123,i,newCart)
    newCart[cartIndex].count += 1
    this.setState({
      cart :newCart
    })
  }else{
    this.setState({
      cart:[...this.state.cart, {
        text:good.text,
        price: good.price,
        active: true,
        count:1
      }]
    })
  }
}

handleChange=(e)=>{
  this.setState({
    name:e.target.value
  })
}

render(){
  return <div>
    {this.state.showTitle && <Title title={this.props.title}></Title>}
    <div>
      <input
        type="text"
        value={this.state.name}
        onChange={this.handleChange}
      />
    </div>
    <ul>

    {this.state.goods.map((good,i)=>{
      return <li key={good.id}>
        <span>{good.text}</span>
        <span>{good.price}</span>元
        <button onClick={()=>this.handleClick(i)}>添加购物车</button>
      </li>
    })}

  </div>
}

```

```

    }
  }

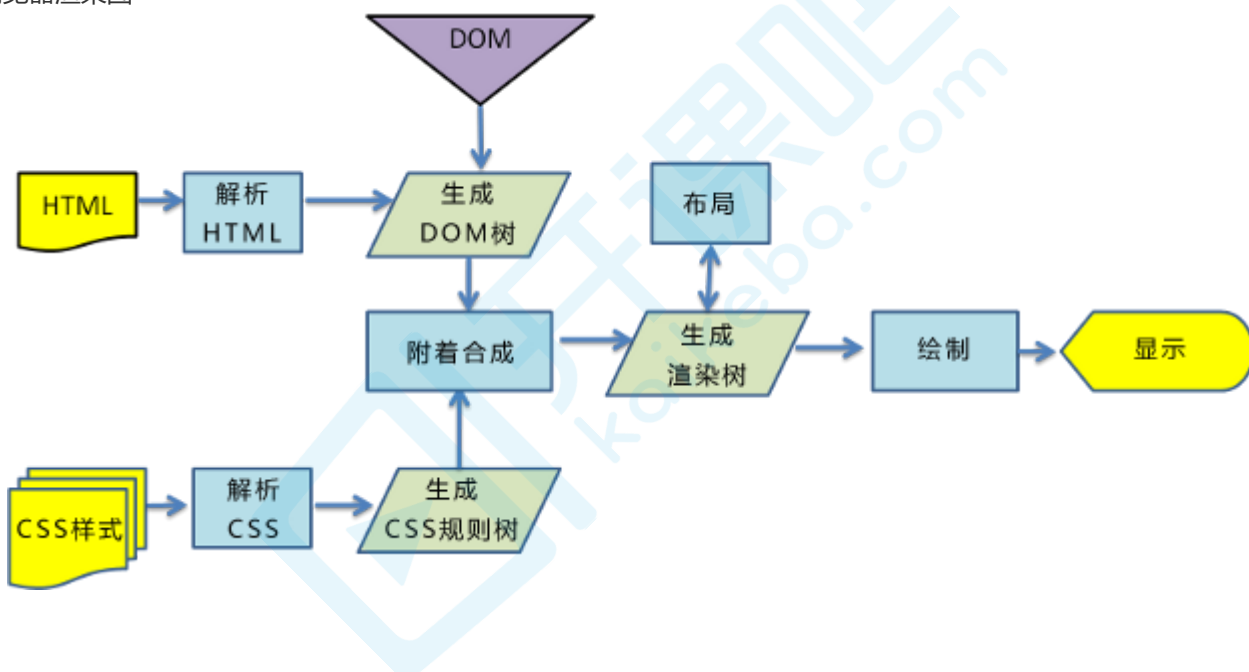
  </ul>
  <button>{this.state.name}</button>
  <hr/>
  <Cart data={this.state.cart}> </Cart>
</div>
}
}

export default KaikebaCart

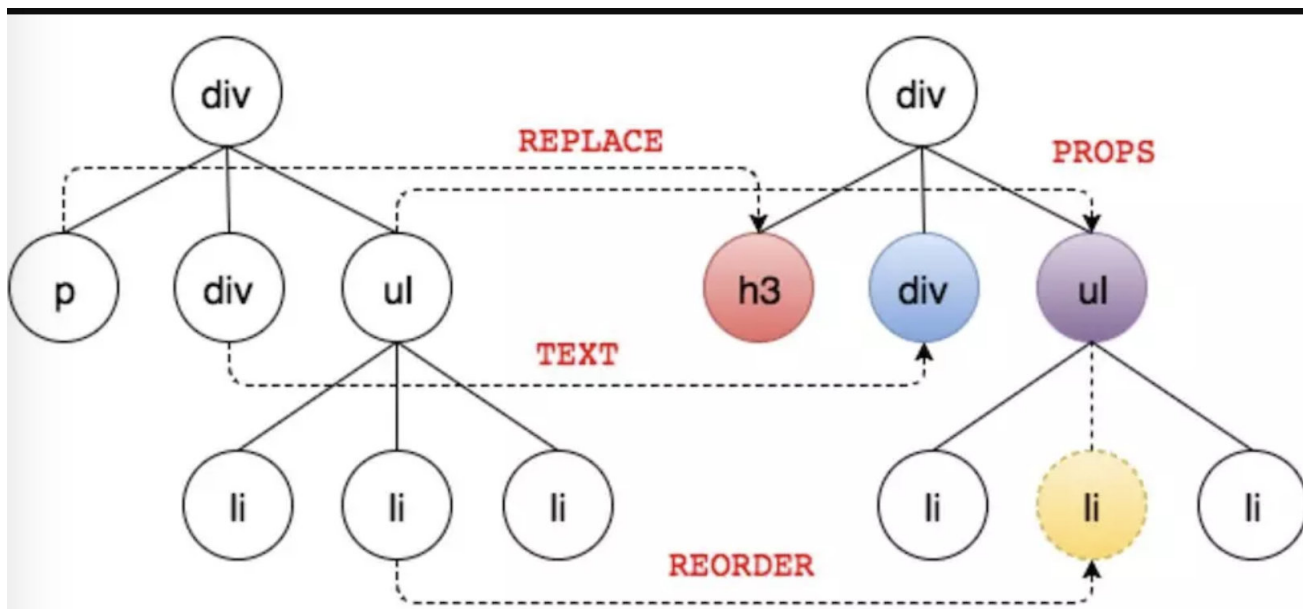
```

大名鼎鼎的虚拟DOM

浏览器渲染图



dom操作成本实在是太高，所以才有了在js里模拟和对比，JSX里使用react。createElement构建虚拟dom，每次有修改，先对比js里的虚拟dom树



生命周期

最近刚更新的生命周期 会后面配合异步渲染进行讲解



###

后续展望

回顾

