Webapp第4天课程

01. 复习React项目搭建过程

02. 使用class关键字创建组件

03. class定义的组件中一些注意事项

04. React中this.state的使用和React的事件绑定机制

1、在constructor中定义this.state = {}定义组件私有属性，可读可写的
2、在组件中通过{this.state.xxx}来访问

3、没有指令 用onClick来绑定，C大写
{/* 1.1 在React中，如果想要为元素绑定事件，不能使用 网页中 传统的 onclick 事件，而是需要 使用 React 提供的 onClick */}
{/* 1.2 也就是说：React中，提供的事件绑定机制，使用的 都是驼峰命名，同时，基本上，传统的 JS 事件，都被 React 重新定义了一下，改成了 驼峰命名 onMouseMove */}
{/* 2.1 在 React 提供的事件绑定机制中，事件的处理函数，必须直接给定一个 function，而不是给定一个 function 的名称 ，用{}包裹*/}
{/* 2.2 在为 React 事件绑定 处理函数的时候，需要通过 this.函数名， 来把 函数的引用交给 事件 */}

```
<input type="button" value="修改 msg" id="btnChangeMsg" onClick={this.changeMsg}
```

4、方法中的this默认为undefined，1改为等于箭头函数 2调用时bind（this）

5、更新 修改this.state = {xx}是不推荐的，用this.setState(obj)

```
// 注意： 这是固定写法，this.state 表示 当前组件实例的私有数据对象，就好比 vue 中，组件实例身上的 data(){ return {} } 函数
this.state = {}
}
```

05. this.setState的使用注意事项

1. 只会覆盖显示定义的，没有提供的值不会覆盖，按需修改
2、this.setState第一个参数支持函数，函数必须返回对象，此函数第一个参数为旧的数据，第二个参数为外部传递的参数props
3、经过测试this.setState修改数据是异步执行的
4、要想顺序执行，在this.setState中传递第二个参数，回调函数

对应需求，访问修改后的state中的数据

```
this.setState(function (prevState, props) {
  // console.log(props)
  return {
    msg: '123'
  }
}, function () {
  // 由于 this.setState 是异步执行的，所以，如果想要立即拿到最
  console.log(this.state.msg)
})
```

06. 有状态组件和无状态组件的对比

// 注意： 两种创建组件的方式，有着本质上的区别，其中，
// 使用 function 构造函数创建的组件，内部没有 state 私有数据，只有 一个 props 来接收外界传递过来的数据；
// 使用 class 关键字 创建的组件，内部，除了有 this.props 这个只读属性之外，还有一个 专门用于 存放自己私有数据的 this.state 属性，这个 state 是可读可写的！
// 基于上面的区别：我们可以为这两种创建组件的方式，下定义了： 使用 function 创建的组件，叫做【无状态组件】；使用 class 创建的组件，叫做【有状态组件】
// 有状态组件和无状态组件，最本质的区别，就是有无 state 属性；同时， class 创建的组件，有自己的生命周期函数、状态，但是，function 创建的 组件，只有props
// 问题来了：什么时候使用 有状态组件，什么时候使用无状态组件呢？？？
// 1. 如果一个组件需要存放自己的私有数据，或者需要在组件的不同阶段执行不同的业务逻辑，此时，非常适合用 class 创建出来的有状态组件；
// 2. 如果一个组件，只需要根据外界传递过来的 props，渲染固定的 页面结构就完事儿了，此时，非常适合使用 function 创建出来的 无状态组件；（使用无状态组件的小小好处：由于剔除了组件的生命周期，所以，运行速度会相对快一丢丢）

07. 渲染评论列表-第1版

0、有状态组件
1、数据挂载到this.state中
2、如何渲染，如何循环？
A、在return的外面用for循环，创建一个数组，数组中包含循环dom
B、this.state.mets.map(e=>{})
3、 不要忘了key

```
var arr = []
this.state.cmts.forEach(item => {
  arr.push(<h1>{item.user}</h1>)
})
```

```
return <div>
  {/* 我们可以直接在 JSX 语法内部，使用 数组的 map 函数，来遍历数组的每一项，并使用 map 返回操作后的最新的数组 */}
  {this.state.cmts.map(item => {
    return <h1>{item.user}</h1>
  })}
</div>
```

08. 渲染评论列表-第2版

1、将评论项单独抽离出来作为一个组件
2、key值的绑定，绑定到最外层上
3、合理使用属性扩散运算符

```
// 封装一个 评论项 组件，此组件由于不需要自己的 私有数据，所以直接定义为 无状态组件
function CommentItem(props) {
  return <div>
    <h1>评论人：{props.user}</h1>
    <h3>评论内容：{props.content}</h3>
  </div>
}
```

```
return <div>
  {/* 我们可以直接在 JSX 语法内部，使用 数组的 map 函数，来遍历数组的每一项，并使用 map 返回
  {this.state.cmts.map((item, i) => {
    return <CommentItem user={item.user} content={item.content} key={i}></CommentItem>
  })}
</div>
```

```
return <CommentItem {...item} key={i}></CommentItem>
```

09. 渲染评论列表-第3版

1、优化代码，将评论列表和评论项抽离到单独文件中
2、不需要注册对比vue

10. 渲染评论列表-第4版

1、新建样式表
2、导入主文件
3、写样式，h1添加className，通过class指定
行内样式 字符串是不对的，用js语法
注意驼峰命名法，padding-left转化为paddingLeft
Px可以省略用一个数字代替

```
<div style="border:1px solid gray;">
```

```
Uncaught Error: The `style` prop expects a mapping from style properties to values, not a string. For example, style={{marginRight: spacing + 'em'}} when using JSX.
```

```
// 在 写 style 样式的时候，外层的 { } 表示 要写JS代码了，内层的 { } 表示 用一个JS对象来表示样式
return <div style={{ border: '1px solid #ccc', margin: '10px 0', paddingLeft: '15px' }}>
```

11. 渲染评论列表-第5版

1、样式优化将行内样式抽离出来，
2、 封装到一个对象中
3、将对象导出到单独文件中
在使用import时，只能放在模块最顶端

12. React中启用css样式表文件的模块化

1、去除行间样式，改成css模块化
新建css文件引入 开始时相安无事
将父组件的类改为title，和评论项的title类重名，样式出错，混杂在一起，怎么办呢？
Vue有scop React种无scop 使用css的模块化
import '../../css/commentItem.css'这种导入方式不是模块化
import itemStyles from '../../css/commentItem.css'是模块化的写法，
在未配置loade前样式照样可以使用，但是itemStyle导出为空对象

修改loader 见图1，module，此时导出的itemStyle为一个命名类与自动计算类的映射如图2
使用时所有类名前加上itemStyle如图3

注释掉commonlist的导入，需求，想让commonlist中可以使用commomitem中的title类，该怎么做呢 如图四将title共享出去，此时itemStyle访问不到title

可以将类名替换一下 查看图1.1
Name指向样式表名 localid指向原类名
总结配置方法：1、导出方式具体导出一个对象obj
2、修改loader加一个modules参数 复数
3、所有类名前面加上对象obj
注：导出公共的类 用 ：global（.类名） 如图5

```
rules: [
  { test: /\.css$/, use: ['style-loader', 'css-loader?modules'] },
```

```
{ test: /\.css$/, use: ['style-loader', 'css-loader?modules&localIdentName=[name]_[local]-[hash:5]'] },
```

```
/* 注意：当启用 CSS 模块化之后，这里所有的类名，都是私有的，如果想要把类名设置成全局的一个类，可以把这个类名，用 :global() 给包裹起来 */
:global(.title){
  font-size: 16px;
  color: purple;
}
```

```
return <div className={itemStyles.box}>
  <h1 className={itemStyles.title}>评论人：{props.user}</h1>
  <h3 className={itemStyles.title}>评论内容：{props.content}</h3>
```

13. 组件生命周期-创建阶段的生命周期函数

1虚拟dom挂载期间的生命周期函数
1、componentWillMount 数据生成 虚拟dom未生成，适合更新数据
2、render 生成虚拟dom
3、componentDidMount组件挂载到页面

14. 组件生命周期-生命周期图详解

组件运行期间的生命周期 3中触发重新render的事件
1、修改props
2、修改state
3、卸载组件

15. 组件生命周期-分别总结各阶段每个生命周期函数的特点

- **组件创建阶段**：组件创建阶段的生命周期函数，有一个显著的特点：创建阶段的生命周期函数，在组件的一辈子中，只执行一次；

  > componentWillMount: 组件将要被挂载，此时还没有开始渲染虚拟DOM render：第一次开始渲染真正的虚拟DOM，当render执行完，内存中就有了完整的虚拟DOM了 componentDidMount: 组件完成了挂载，此时，组件已经显示到了页面上，当这个方法执行完，组件就进入都了 运行中 的状态

- **组件运行阶段**：也有一个显著的特点，根据组件的state和props的改变，有选择性的触发0次或多次；

  > componentWillReceiveProps: 组件将要接收新属性，此时，只要这个方法被触发，就证明父组件为当前子组件传递了新的属性值； shouldComponentUpdate: 组件是否需要被更新，此时，组件尚未被更新，但是，state 和 props 肯定是最新的 componentWillUpdate: 组件将要被更新，此时，尚未开始更新，内存中的虚拟DOM树还是旧的 render: 此时，又要重新根据最新的 state 和 props 重新渲染一棵内存中的 虚拟DOM树，当 render 调用完毕，内存中的旧DOM树，已经被新DOM树替换了！此时页面还是旧的 componentDidUpdate: 此时，页面又被重新渲染了，state 和 虚拟DOM 和 页面已经完全保持同步

- **组件销毁阶段**：也有一个显著的特点，一辈子只执行一次；

  > componentWillUnmount: 组件将要被卸载，此时组件还可以正常使用；