React-2

上节课重点内容回顾

- JSX 注意事项
  - 必须有,且只有一个顶层的包含元素 - React.Fragment
  - JSX 不是html，很多属性在编写时不一样
    - className
    - style
  - 列表渲染时，必须有 key 值
  - 在 jsx 所有标签必须闭合
  - 组件的首字母一定大写，标签一定要小写
- 类组件
  - 组件类必须继承 **React.Component**
  - 组件类必须有 **render** 方法
- 事件
  - 大小写问题
  - this 问题
- props 和 state
  - props 父组件传递过来的参数
  - state 组件自身状态

课程目标

- 掌握 setState 的各种使用情况
- 掌握 React 组件间通信
- 掌握 React 组件的生命周期
- 掌握受控组件的使用

课程内容

state 和 setState

- setState(updater, [callback])
  - updater: 更新数据 FUNCTION/OBJECT
  - callback: 更新成功后的回调 FUNCTION
  - 异步:react通常会集齐一批需要更新的组件，然后一次性更新来保证渲染的性能
  - 浅合并 Objecr.assign()

组件间通信

在 React.js 中，数据是从上自下流动（传递）的，也就是一个父组件可以把它的 state / props 通过 props 传递给它的子组件，但是子组件不能修改 props - React.js 是单向数据流，如果子组件需要修改父组件状态（数据），是通过回调函数方式来完成的。

- 父级向子级通信 把数据添加子组件的属性中，然后子组件中从props属性中，获取父级传递过来的数据

- 子级向父级通信 在父级中定义相关的数据操作方法(或其他回调), 把该方法传递给子级，在子级中调用该方法父级传递消息
- 案例：完善好友列表

跨组件通信 **context** - 扩展

- React.createContext(defaultValue); { Consumer, Provider } = createContext(defaultValue)
- Context.Provider 在父组件调用 Provider 传递数据
  - value 要传递的数据
- 接收数据
  - class.contextType = Context;
  - static contextType = Context;
    - this.context;
  - Context.Consumer `<Consumer> {(props)=>{ console.log(props); return <div></div> }} </Consumer>` 注意在使用不熟练时，最好不要再项目中使用 **context**，**context**一般给第三方库使用

组件的生命周期

所谓的生命周期就是指某个事物从开始到结束的各个阶段，当然在 React.js 中指的是组件从创建到销毁的过程，React.js 在这个过程中的不同阶段调用的函数，通过这些函数，我们可以更加精确的对组件进行控制，前面我们一直在使用的 render 函数其实就是组件生命周期渲染阶段执行的函数

生命周期演变

之前（**React 16.3** 之前）

- 挂载阶段
  - constructor
  - componentWillMount
  - render
  - componentDidMount
- 更新阶段
  - 父组件更新引起组件更新
    - componentWillReceiveProps(nextProps)
    - shouldComponentUpdate(nextProps, nextState)
    - componentWillUpdate(nextProps, nextState)
    - render
    - componentDidUpdate(prevProps, prevState)
  - 组件自身更新
    - shouldComponentUpdate
    - componentWillUpdate
    - render
    - componentDidUpdate
- 卸载阶段
  - componentWillUnmount

现在

- 挂载阶段
    - constructor
    - static getDerivedStateFromProps(props, state)
        - 注意 this 问题
    - render
    - componentDidMount
- 更新阶段
    - 父组件更新引起组件更新
        - static getDerivedStateFromProps(props, state)
        - shouldComponentUpdate()
        - componentWillUpdate()
        - render()
        - getSnapshotBeforeUpdate()
        - componentDidUpdate()
    - 组件自身更新
        - shouldComponentUpdate()
        - componentWillUpdate()
        - render()
        - getSnapshotBeforeUpdate()
        - componentDidUpdate()
- 卸载阶段
    - componentWillUnmount
- 错误处理
    - static getDerivedStateFromError()
    - componentDidCatch(error, info) 参考：http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

生命周期函数详解

constructor(props)

类的构造函数，也是组件初始化函数，一般情况下，我们会在这个阶段做一些初始化的工作

- 初始化 state
- 处理事件绑定函数的 this

render()

render 方法是 Class 组件必须实现的方法

static getDerivedStateFromProps(props, state)

该方法会在 render 方法之前调用，无论是挂载阶段还是更新阶段，它的存在只有一个目的：让组件在 props 变化时更新 state

componentDidMount()

在组件挂载后（render 的内容插入 DOM 树中）调用。通常在这个阶段，我们可以：

- 操作 DOM 节点
- 发送请求

shouldComponentUpdate(nextProps, nextState)

发生在更新阶段，getDerivedStateFromProps 之后，render 之前，该函数会返回一个布尔值，决定了后续是否执行 render，首次渲染不会调用该函数

getSnapshotBeforeUpdate(prevProps, prevState)

该方法在 render() 之后，但是在输出到 DOM 之前执行，用来获取渲染之前的快照。当我们想在当前一次更新前获取上次的 DOM 状态，可以在这里进行处理，该函数的返回值将作为参数传递给下个生命周期函数 componentDidUpdate

> 该函数并不常用。

componentDidUpdate()

该函数会在 DOM 更新后立即调用，首次渲染不会调用该方法。我们可以在这个函数中对渲染后的 DOM 进行操作

componentWillUnmount()

该方法会在组件卸载及销毁前调用，我们可以在这里做一些清理工作，如：组件内的定时器、未完成的请求等

错误处理

当渲染过程，生命周期，或子组件的构造函数中抛出错误时，会调用如下方法

- static getDerivedStateFromError()
- componentDidCatch()

受控组件

非受控组件: 我们不需要同步 value 值(defaultValue，defaultChecked)

todoList 初实现

下节课内容

- 掌握React其他 API 使用：PureComponent、ref、children、dangerouslySetInnerHTML、key
- 掌握函数式组件及常见 hooks 的使用
- 可以独立开发基于Hooks 或 Component 的 todoList 应用