React

What ?

React 是 Facebook 提供的一个为数据提供渲染视图的 js 库

Why ?

1) 大量的DOM操作 -> 浏览器遇到性能瓶颈

VDOM (Virtual DOM)

2) 逻辑非常复杂 -> 维护成本高

单向数据流：让数据单向流动

特点：

1) 声明式设计：可以轻松描述应用程序

2) 高效：通过VDOM，最大限度的减少DOM的操作

3) 灵活：reactjs 是一个非常灵活的Library

4) 组件：更好的完成视图的封装和复用

5) JSX：JavaScriptXML 并不是新语言，只不过是允许在 js 中写标签

how ?

搭建环境

方式1：引入对应的 js文件

react.js

react-dom.js

browser.min.js

方式2：CLI(command line interface)

npx create-react-app my-app

cd my-app

npm start

基本用法

① 一个方法

ReactDOM.render(A,B);

② 编译器 babel

支持将es6、jsx转换为当前高级浏览器能够识别的语法

```
<script type='text/babel'></script>
```

③ jsx

支持在 js 中直接去编写标签

核心概念

- Component
- JSX

  jsx(javascriptxml) 并不是一种新的语言，只不过是允许在js中写标签，允许在

  ```
  标签中执行运算

  ① 遇到 `<`，首字母是小写就会用html来解析，如果是大写按照组件的方式来渲染

  ② 遇到 `{` ，就是用js来执行代码块并将结果输出

  注意事项：

  ① 不允许直接渲染多个元素

  ② 每个元素都要写开始和结束标记

  jsx 并不是强制要求，我们可以通过方法 `React.createElement` 也能实现页面效果，但是过于复杂
  ```

- props
- ref
- state

Component

组件是可被反复使用的，带有特定功能的视图

回顾：

```
// Vue:
//global
Vue.component('my-cart',{})
  <my-cart></my-cart>
```

```
//local
Vue.component('my-cart',{
  components:{
    'my-test':{
    }
  }
})
// Angular:
import {Component} from '@angular/core'
@Component({
  selector:'demo01',
  templateUrl:''
})
export class Demo01Component{}
app.module.ts
@NgModule({
  declarations:[Demo01Component]
})
<demo01></demo01>
```

React:

① 创建

var MyTest = React.createClass();
② 调用

`<MyTest></MyTest>`

注意事项：

① 组件类的名称遵循大驼峰

② 组件类的render方法在指定要渲染的元素时，不允许在第一个元素的位置直接换行

③ 不能直接渲染多个，要放在一个顶层标签中

复合组件：

允许在一个组件中来调用其它的封装好的组件

Props

父 -> 子

> 回顾：
>
> props 用来完成父与子的通信

```
// Vue：
// ① 传值
<son age="20"></son>
// ② 接收
Vue.component('son',{
// 允许接收age属性的值
// 创建一个变量age
// this.age
props:['age']
})
```

```
// Angular：
// ① 传值
<son age="20"></son>
// ② 接收
import {Input} from '@angular/core'
export class Son{
 @Input() age="";
 //this.age
}
```

React 属性传值的基本用法:

```
// ① 传值
<Son myName="zhangsan"></Son>
// ② 收值
Son = React.createClass({
  //this.props.myName
})
```

this.props.children

> this.props 对象中的属性和调用该组件类时指定的属性是一一对应的，
>
> 有一个例外:this.props.children

```
//this.props.children.map()
React.Children.map(
this.props.children,
(value,index)=>{
  // value就是嵌套的其中的一个子元素
  // index当前遍历时的小标
})
```

注意事项：

this.props.children属性对应的值是不确定的，

如果调用了一个子元素-->object

如果调用了多个子元素-->Array

如果没有调用子元素，undefined

子 -> 父

回顾：

借助props来实现子与父通信

```
// Vue:
// ① 给子组件在调用时 绑定事件以及对应的事件处理函数
Vue.component('father',{
  methods:{
    rcv(msg){}
  },
  template:`<son @myEvent="rcv"></son>`
})
// ② 子组件触发事件并传值
this.$emit('myEvent',123)
```

```
// Angular:
// ① 绑定
rcv(msg){
  // msg触发事件时传来的值
}
<son (myEvent)="rcv($event)"></son>
// ② 触发
import {Output,EventEmitter} from '@angular/core'
@Output() myEvent = new EventEmitter();
this.myEvent.emit('123')
```

共识：父组件可以传一个方法 给子组件，子组件也可以调用这个父组件的方法

①父组件定义一个有参数的方法

rcv(msg){}

②将方法传递给子组件

<Son func={this.rcv}></Son>

③子组件接收到该属性对应的方法，去调用并传值

this.props.func(123)

注意事项：this.props对象中的属性的值是不允许直接修改的

ref

回顾：

```
// Vue:（子-->父）
// ① 调用子组件时 指定ref
<son ref="mySon"></son>
// ② 在父组件中指定通过指定的属性来调用子组件中定义好的属性和方法
this.$refs.mySon.**
```

```
// React:
// ① 给准备要调用的子组件指定一个ref
<MySon ref="zhangsan"></MySon>
// ② 父组件就可以通过引用的名称找到子组件的实例对象
 this.refs.zhangsan
```

state

state 状态

React将组件中数据的变化 通过状态来反映出来，把组件比作是一个状态机

基本功能：

① 管理数据

数据的初始化

```
getIntialState(){
  return {count:1}
}
```

数据的读操作

```
this.state.count
```

数据的写操作

```
// 方法1
this.setState({count:3});
// 方法2:
this.setState({},()=>{
  //状态写操作成功之后的回调函数
})
```

②数据绑定(将数据绑定到视图)

> 小总结
>
> ① jsx:
>
> 在React中构建组件的必不可少一个语法（模板、数据和方法）
>
> ② 组件：
>
> 构建复杂页面的一部分
>
> ③ props:
>
> ①父->子
>
> ②子->父
>
> ③调用组件时嵌套的子元素
>
> ④ ref:
>
> 实现在父组件中主动的获取子组件的实例对象，获取该对象中定义的数据和方法
>
> ⑤ state
>
> 管理数据
>
> 绑定：将状态中的数据 绑定到 视图

生命周期

回顾

```
// Vue:
create/mount/update/destroy
 //两个钩子函数:
  before**
  **ed
// Angular:
  ngOnInit
  ngDoCheck
  ngOnChanges
  ngOnDestroy
```

React组件的生命周期:

1) mount

componentWillMount

componentDidMount

注意事项：

要想操作dom元素，必须等到挂载

2) update

componentWillUpdate

componentDidUpdate

两种情况：

①当前组件的状态发生变化时

②当前组件所传递来的属性的值发生变化

注意事项：

①不允许在 componentWill/DidUpdate 这两个钩子函数中，

执行状态的 **写操作**（陷入无穷循环，会耗尽给当前进程所分配的资源）

3) componentWillReceiveProps

componentDidReceiveProps

当通过属性给一个组件传值时，传的值发生了变化时，可以保存在状态中（允许）

4) unmount

componentWillUnmount

在组件销毁时，执行相关的清理工作

受控表单元素

注意事项：

在React中处理表单元素，不是必须非得按照受控表单元素来处理

1、什么是受控表单元素

表单元素指定属性:

input textarea -->value

checkbox/radio -->checked

select/option --> selected

如果指定了上述属性，表单元素就是受控的表单元素(用户直接操作，结果是错误的)

2、解决方案

方案1：作为非受控表单元素来处理

value --> defaultValue

方案2：借助于状态解决受控表单元素出现的问题

① 初始化一个状态

```
getInitialState(){
  return {myValue:"北京"}
}
```

② 将状态的值 绑定到受控表单元素

```
<input value={this.state.myValue}/>
```

③ 给受控表单元素绑定对应的事件和事件处理函数(完成状态的写操作)

```
handleChange(e){
  //获取输入框的值
  var nowValue = e.target.value
  //保存在状态
  this.setState({myValue:nowValue});
}
<input onChange={this.handleChange} value={this.state.myValue}/>
```

进阶知识

回顾：

控制流的指令：

条件判断

`v-if  *ngIf`

循环功能

`v-for *ngFor`

> 常见面试题：Angular 中在一个组件调用时，可以同时使用判断和循环吗？
>
> `<li *ngIf="" *ngFor=""></li>`
>
> 答案：不允许在一个组件中同时调用多个结构型指令，是不会报错的，可以使用 `ng-container` 来解决问题

React

1、条件判断

方案1：短路逻辑（与运算）

```
{
  expression
    &&
  <Test></Test>
}
```

方案2：定义一个方法

```
// 定义方法
showSth(){
  //运算
  if(条件为真){
    return <组件>
  }else{
    return <组件>
  }
}
// 调用方法:
{
  this.showSth()
}
```

2、循环

```
{
  集合.map((value,index)=>{
    return <组件 key={index}></组件>
  })
}
```