

# react原理

## 函数式编程

### 函数式编程

- ◆ 一种编程范式，概念比较多
- ◆ 纯函数
- ◆ 不可变值

## vdom和diff算法

- ◆ h 函数
- ◆ vnode 数据结构
- ◆ patch 函数

LDFODD168

```
{
  tag: 'div',
  props: {
    className: 'container',
    id: 'div1'
  },
  children: [
    {
      tag: 'p',
      children: 'vdom'
    },
    {
      tag: 'ul',
      props: { style: 'font-size: 20px' },
      children: [
        {
          tag: 'li',
          children: 'a'
        },
        // ....
      ]
    }
  ]
}
```

更多 IT 教程 微信

- ◆ 只比较同一层级，不跨级比较
  - ◆ tag 不相同，则直接删掉重建，不再深度比较
  - ◆ tag 和 key，两者都相同，则认为是相同节点，不再深度比较
- 
- ◆ Vue2.x Vue3.0 React 三者实现 vdom 细节都不同
  - ◆ 核心概念和实现思路，都一样
  - ◆ 面试主要考察后者，不用全部掌握细节

## JSX本质

---

```
// https://www.babeljs.cn/

// // JSX 基本用法
// const imgElem = <div id="div1">
//   <p>some text</p>
//   <img src={imgUrl}/>
// </div>

const imgElem = React.createElement("div", {
  id: "div1"
}, React.createElement("p", null, "some text"), React.createElement("img", {
  src: imgUrl
}));

// // JSX style
// const styleData = { fontSize: '30px', color: 'blue' }
// const styleElem = <p style={styleData}>设置 style</p>

const styleElem = React.createElement("p", {
  style: styleData
}, "\u8BBE\u7F6E style");

// // JSX 加载组件
// const app = <div>
//   <Input submitTitle={onSubmitTitle}/>
```

```

//      <List list={list}/>
// </div>

const app = React.createElement("div", null, React.createElement(Input, {
  submitTitle: onSubmitTitle
}), React.createElement(List, {
  list: list
}));

// // JSX 事件
// const eventList = <p onClick={this.clickHandler}>
//     some text
// </p>

const eventList = React.createElement("p", {
  onClick: this.clickHandler
}, "some text")

// // JSX list
// const listElem = <ul>{this.state.list.map((item, index) => {
//     return <li key={item.id}>index {index}; title {item.title}</li>
// }}}</ul>

const listElem = React.createElement("ul", null, this.state.list.map((item,
index) => {
  return React.createElement("li", {
    key: item.id
  }, "index ", index, "; title ", item.title);
}));

// // 总结
// React.createElement('div', null, [child1, child2, child3])
// React.createElement('div', {...}, child1, child2, child3)
// React.createElement(List, null, child1, child2, '文本节点')
// // h 函数
// // 返回 vnode
// // patch

```

## JSX 本质

- ◆ React.createElement 即 h 函数，返回 vnode
- ◆ 第一个参数，可能是组件，也可能是 html tag
- ◆ 组件名，首字母必须大写（React 规定）

# JSX 本质

```
// 第一个参数是 List 组件
React.createElement(List, {
  list: list
})

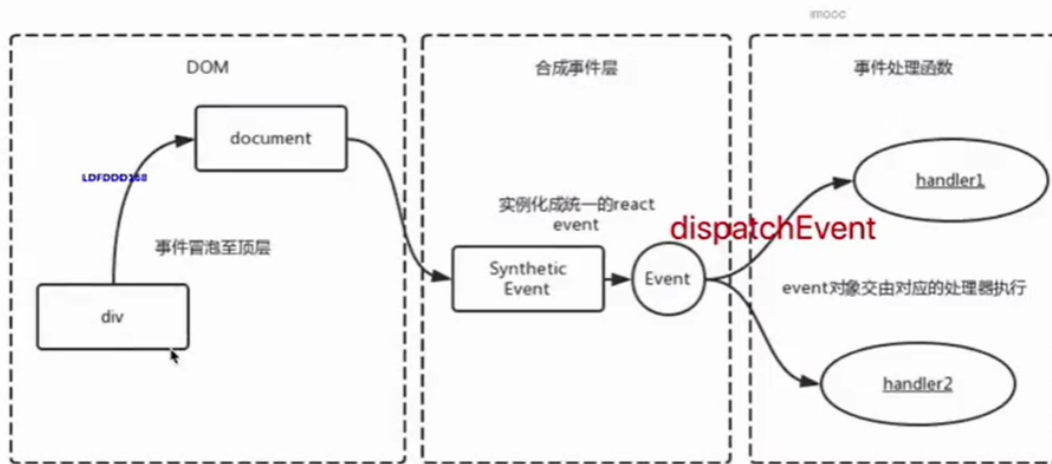
// 找到 List 组件 jsx 结构, 继续拆分
React.createElement("ul", null, list.map(
  function (item, index) {
    return React.createElement("li", {
      key: item.id
    }, "title ", item.title)
  }
))
)
```

## 合成事件

### 合成事件

- ◆ 所有事件挂载到 document 上
- ◆ event 不是原生的, 是 SyntheticEvent 合成事件对象
- ◆ 和 Vue 事件不同, 和 DOM 事件也不同

## 合成事件 - 图示



## 为何要合成事件机制？

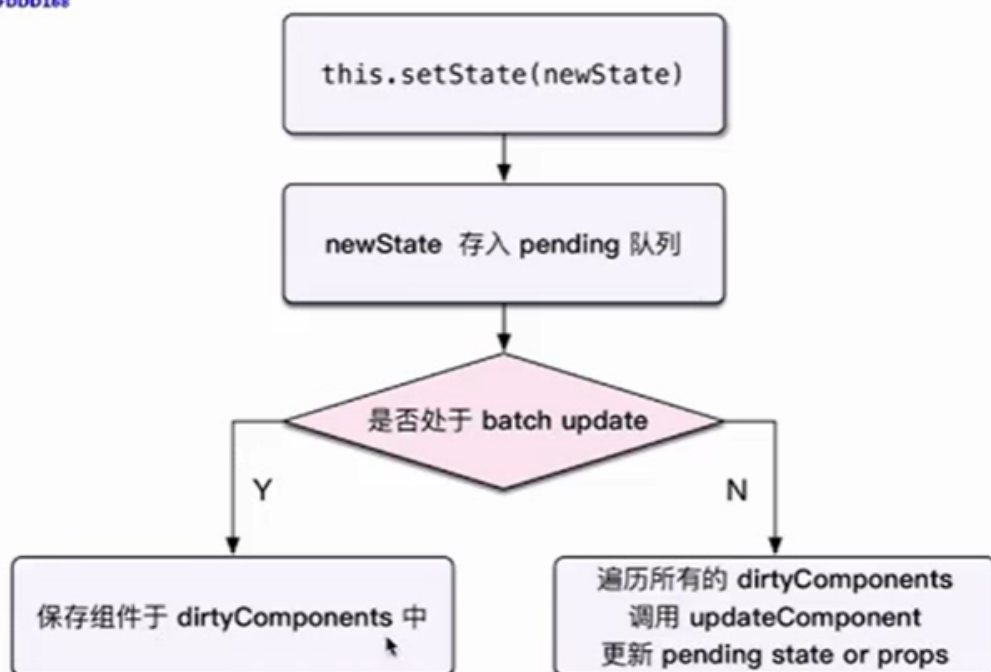
- ◆ 更好的兼容性和跨平台
- ◆ 载到 `document`，减少内存消耗，避免频繁解绑
- ◆ 方便事件的统一管理（如事务机制）

## setState和batchUpdate

- ◆ 有时异步（普通使用），有时同步（`setTimeout`、DOM 事件）
- ◆ 有时合并（对象形式），有时不合并（函数形式）
- ◆ 后者比较好理解（像 `Object.assign`），主要讲解前者

# setState 主流程

LDFDD168



## isBatchingUpdates

```
class ListDemo extends React.Component {
  constructor(props) { ...
  }
  Complexity is 5 Everything is cool!
  render() { ...
  }
  increase = () => {
    // 开始: 处于 batchUpdate
    // isBatchingUpdates = true
    this.setState({
      count: this.state.count + 1
    })
    // 结束
    // isBatchingUpdates = false
  }
}
```

```
class ListDemo extends React.Component {
  constructor(props) { ...
  }
  Complexity is 5 Everything is cool!
  render() { ...
  }
  increase = () => {
    // 开始: 处于 batchUpdate
    // isBatchingUpdates = true
    setTimeout(() => {
      // 此时 isBatchingUpdates 是 false
      this.setState({
        count: this.state.count + 1
      })
    })
    // 结束
    // isBatchingUpdates = false
  }
}
```

更多IT教程 微信 3528527

## isBatchingUpdates

```
componentDidMount() {  
  // 开始: 处于 batchUpdate  
  // isBatchingUpdates = true  
  document.body.addEventListener('click', () => {  
    // 此时 isBatchingUpdates 是 false  
    this.setState({  
      count: this.state.count + 1  
    })  
    console.log('count in body event', this.state.count)  
  })  
  // 结束  
  // isBatchingUpdates = false  
}
```

## setState 异步还是同步？

- ◆ setState 无所谓异步还是同步
- ◆ 看是否能命中 batchUpdate 机制
- ◆ 判断 isBatchingUpdates

## 哪些能命中 batchUpdate 机制


- ◆ 生命周期（和它调用的函数）
- ◆ React 中注册的事件（和它调用的函数）
- ◆ React 可以“管理”的入口



## 哪些不能命中 batchUpdate 机制

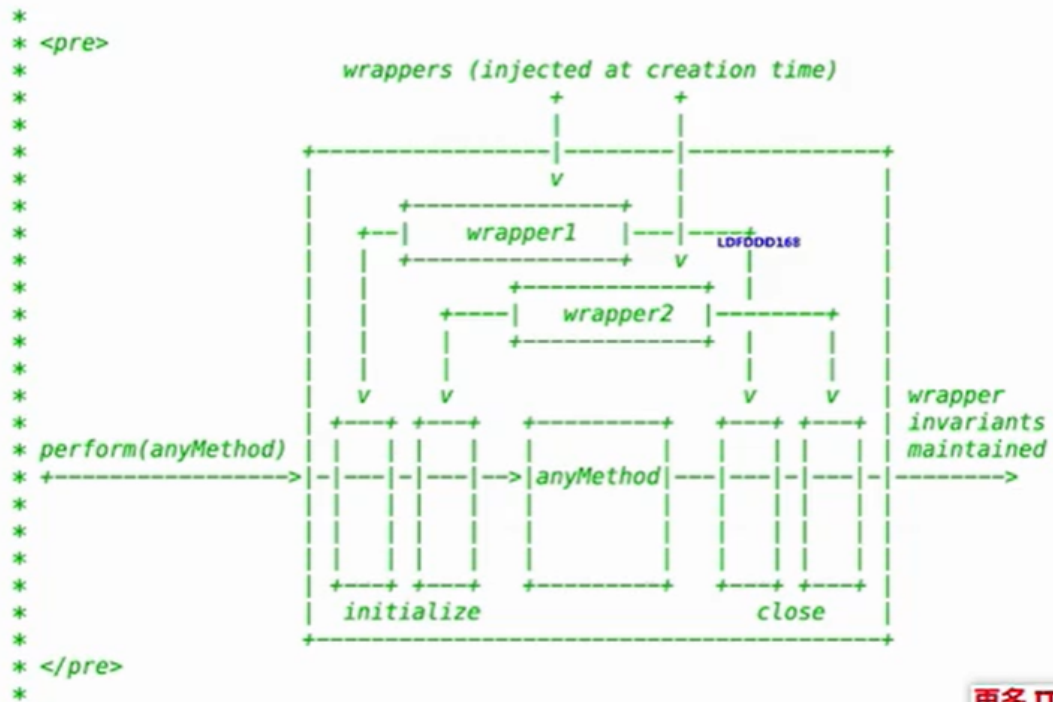
- ◆ setTimeout setInterval 等 ( 和它调用的函数 )
- ◆ 自定义的 DOM 事件 ( 和它调用的函数 )
- ◆ React “管不到” 的入口

## transaction 事务机制

```
class ListDemo extends React.Component {  
  constructor(props) { ...  
  }  
  Complexity is 5 Everything is cool!  
  render() {  ...  
  }  
  increase = () => {  
    // 开始: 处于 batchUpdate  
    // isBatchUpdates = true  
  
    // 其他任何操作  
  
    // 结束  
    // isBatchingUpdates = false  
  }  
}
```



## transaction 事务机制



## 百名 IT 教程

## transaction 事务机制

```
transaction.initialize = function () {  
    console.log('initialize')  
}  
  
transaction.close = function () {  
    console.log('close')  
}  
  
function method(){  
    console.log('abc')  
}  
  
transaction.perform(method)
```

LD/000168

```
// 输出 'initialize'
// 输出 'abc'
// 输出 'close'
```

## 组件渲染

## 组件渲染和更新过程

- ◆ JSX 如何渲染为页面
- ◆ setState 之后如何更新页面
- ◆ 面试考察全流程

## 组件渲染过程

- ◆ props state
- ◆ render() 生成 vnode
- ◆ patch(elem, vnode)

## 组件更新过程

- ◆ setState(newState) --> dirtyComponents ( 可能有子组件 )
- ◆ render() 生成 newVnode
- ◆ patch(vnode, newVnode)

## 更新的两个阶段

- ◆ 上述的 patch 被拆分为两个阶段：  
WTOOC
- ◆ reconciliation 阶段 - 执行 diff 算法，纯 JS 计算
- ◆ commit 阶段 - 将 diff 结果渲染 DOM

## 可能会有性能问题

LDFOOD168

- ◆ JS 是单线程，且和 DOM 渲染共用一个线程
- ◆ 当组件足够复杂，组件更新时计算和渲染都压力大
- ◆ 同时再有 DOM 操作需求（动画，鼠标拖拽等），将卡顿

## 关于 fiber

- ◆ React 内部运行机制，开发者体会不到  
LDFOOD168
- ◆ 了解背景和基本概念即可