02

OPEN ORIENTED

凹 凸 实 验 室

一起来造个轮子（二）

基于Virtual Dom的组件框架

luckyadam

1 React回顾

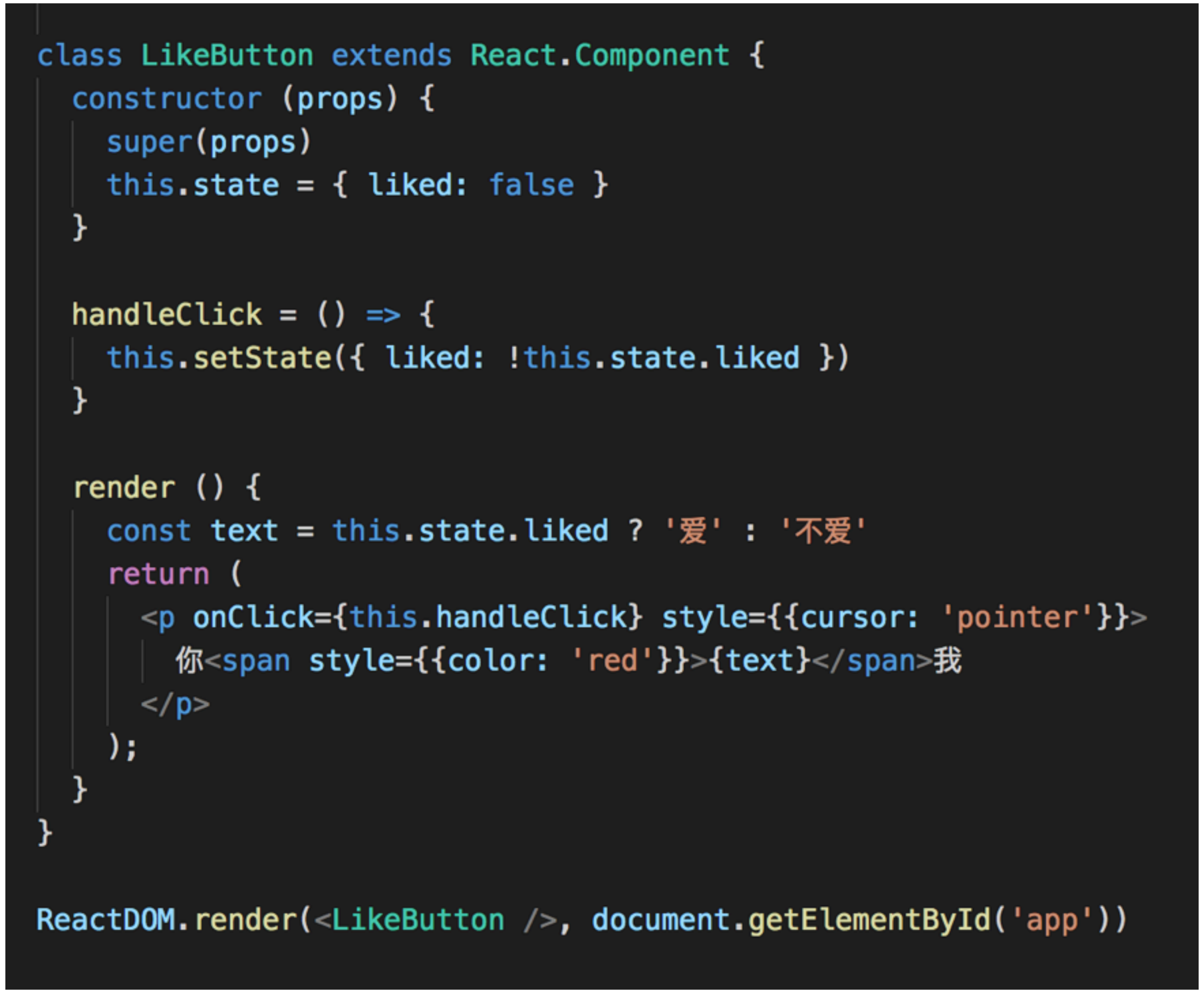
2 组件框架设计

3 属性处理

1

React回顾

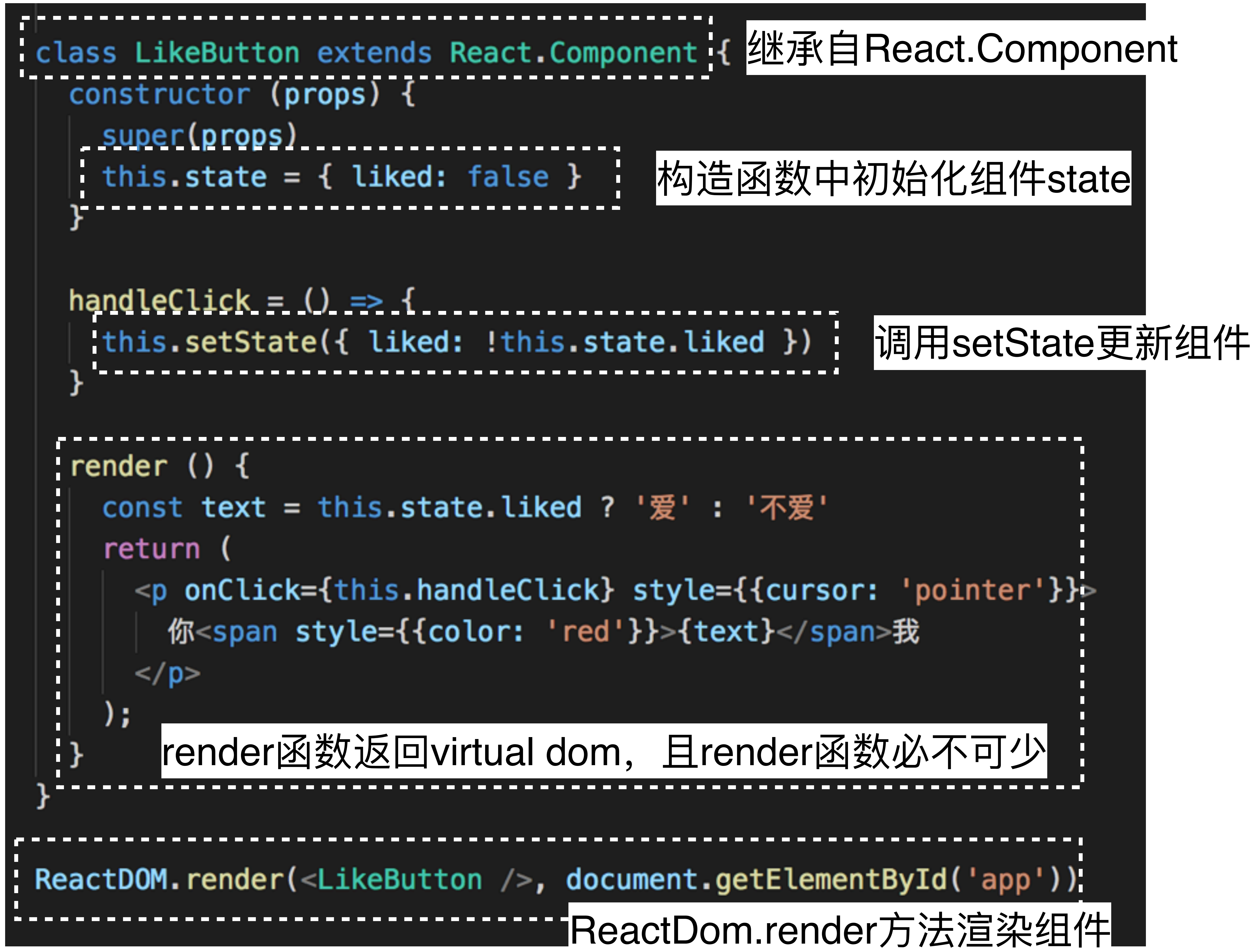
React回顾



```
class LikeButton extends React.Component {
  constructor (props) {
    super(props)
    this.state = { liked: false }
  }

  handleClick = () => {
    this.setState({ liked: !this.state.liked })
  }

  render () {
    const text = this.state.liked ? '爱' : '不爱'
    return (
      <p onClick={this.handleClick} style={{cursor: 'pointer'}}>
        你<span style={{color: 'red'}}>{text}</span>我
      </p>
    );
  }
}

ReactDOM.render(<LikeButton />, document.getElementById('app'))
```

React组件

React回顾

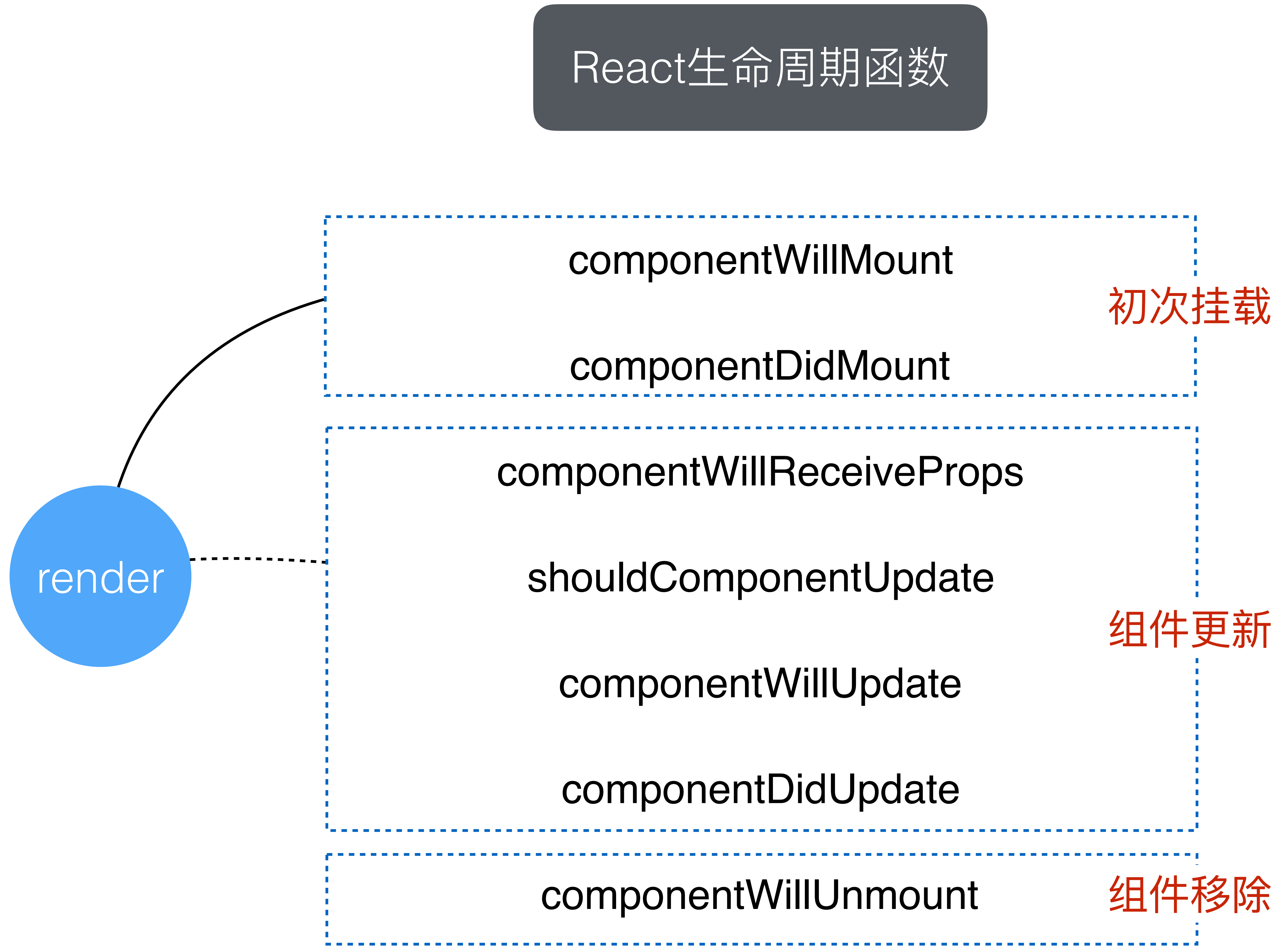


```
class LikeButton extends React.Component {
  constructor (props) {
    super(props)
    this.state = { liked: false }
  }

  handleClick = () => {
    this.setState({ liked: !this.state.liked })
  }

  render () {
    const text = this.state.liked ? '爱' : '不爱'
    return (
      <p onClick={this.handleClick} style={{cursor: 'pointer'}}>
        你<span style={{color: 'red'}}>{text}</span>我
      </p>
    );
  }
}

ReactDOM.render(<LikeButton />, document.getElementById('app'));
```

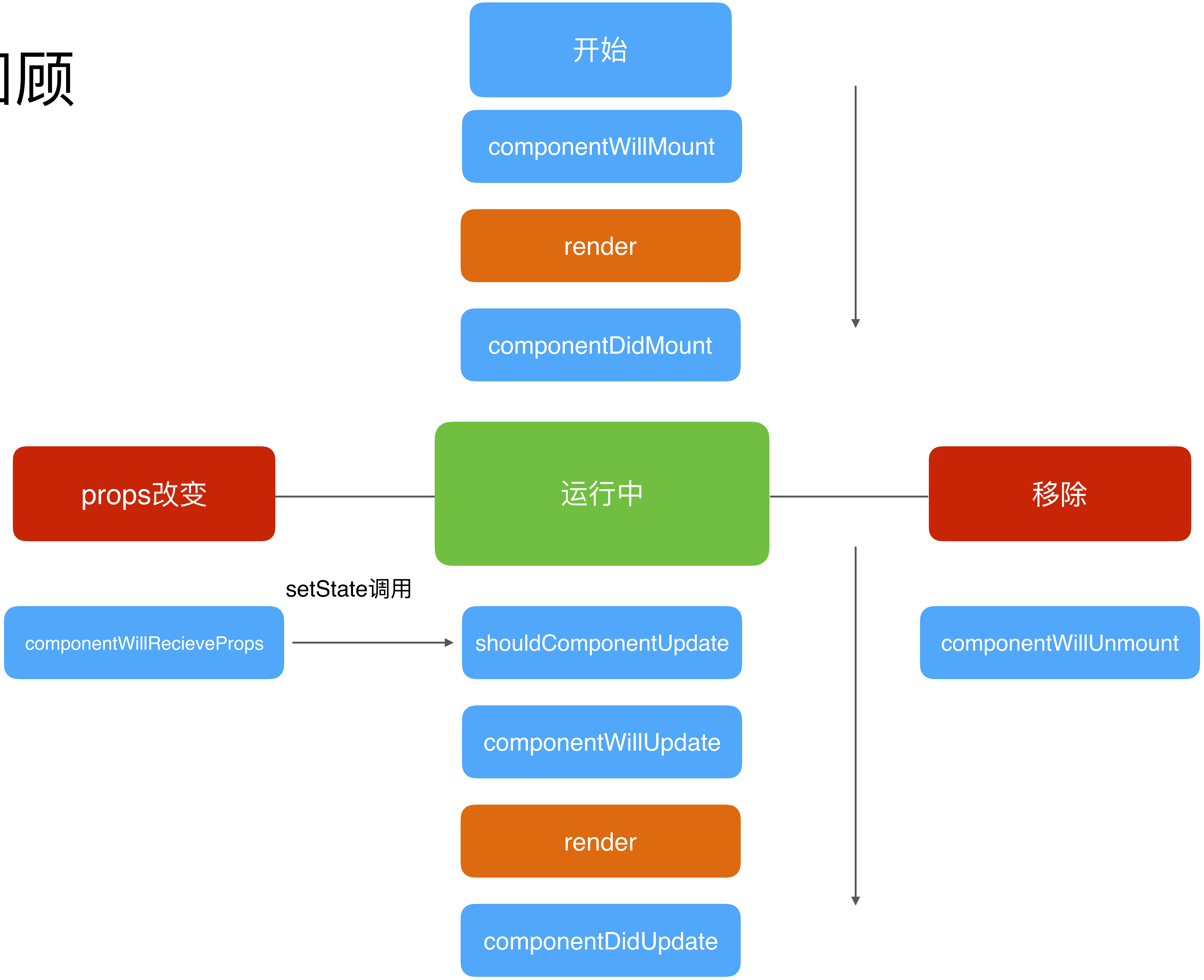
继承自React.Component
构造函数中初始化组件state
调用setState更新组件
render函数返回virtual dom，且render函数必不可少
ReactDom.render方法渲染组件

ES6写法React组件

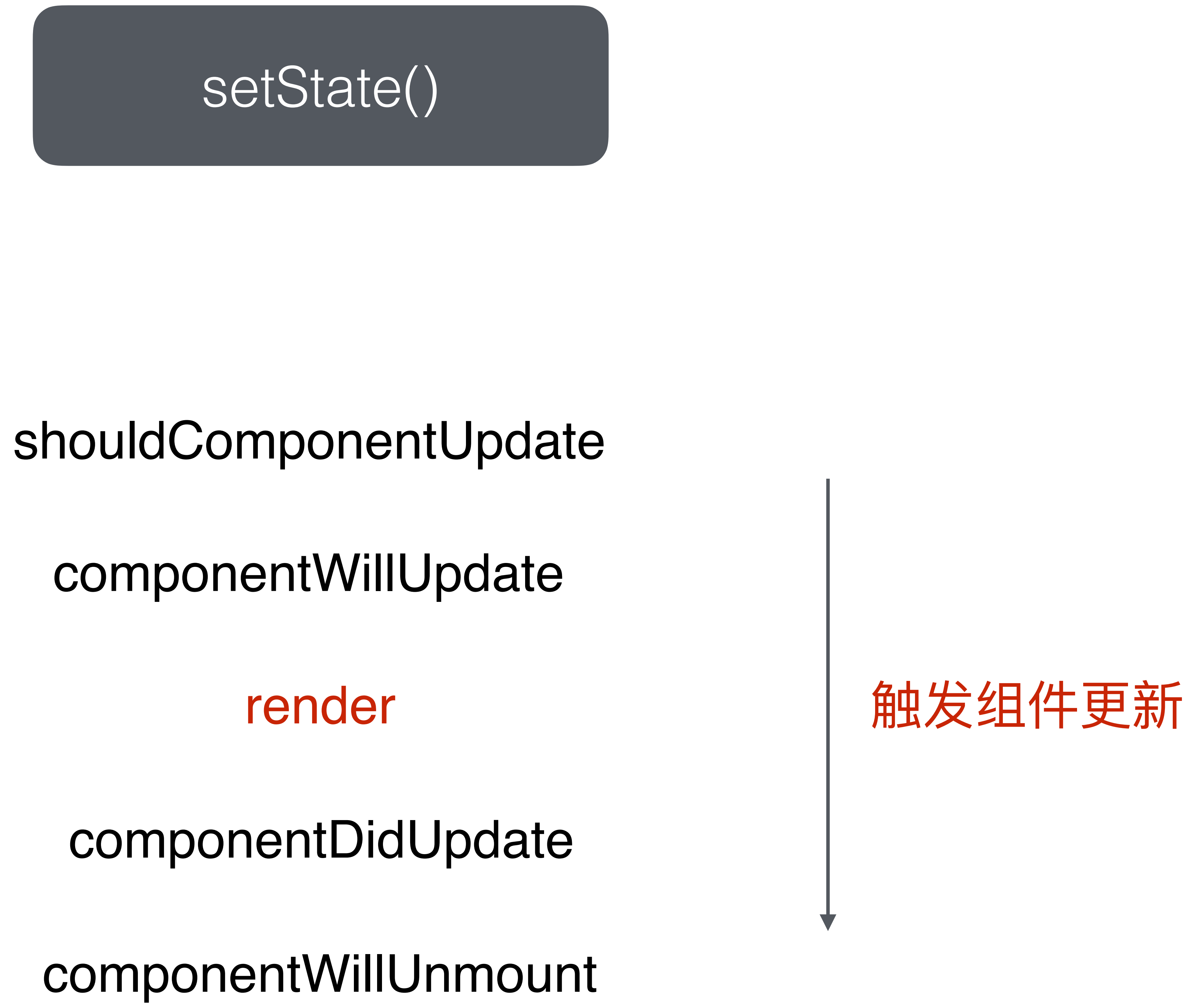
React回顾


React生命周期函数
render
componentWillMount
componentDidMount
初次挂载
componentWillReceiveProps
shouldComponentUpdate
componentWillUpdate
componentDidUpdate
组件更新
componentWillUnmount
组件移除

React回顾


开始
componentWillMount
render
componentDidMount
props改变
运行中
移除
setState调用
componentWillRecieveProps
shouldComponentUpdate
componentWillUpdate
render
componentDidUpdate
componentWillUnmount

React回顾


setState()
shouldComponentUpdate
componentWillUpdate
render
componentDidUpdate
componentWillUnmount
触发组件更新

2

组件框架设计

组件框架设计

一切皆有始

render(vnode, dom, callback)

将组件（虚拟dom）挂载到真实dom上

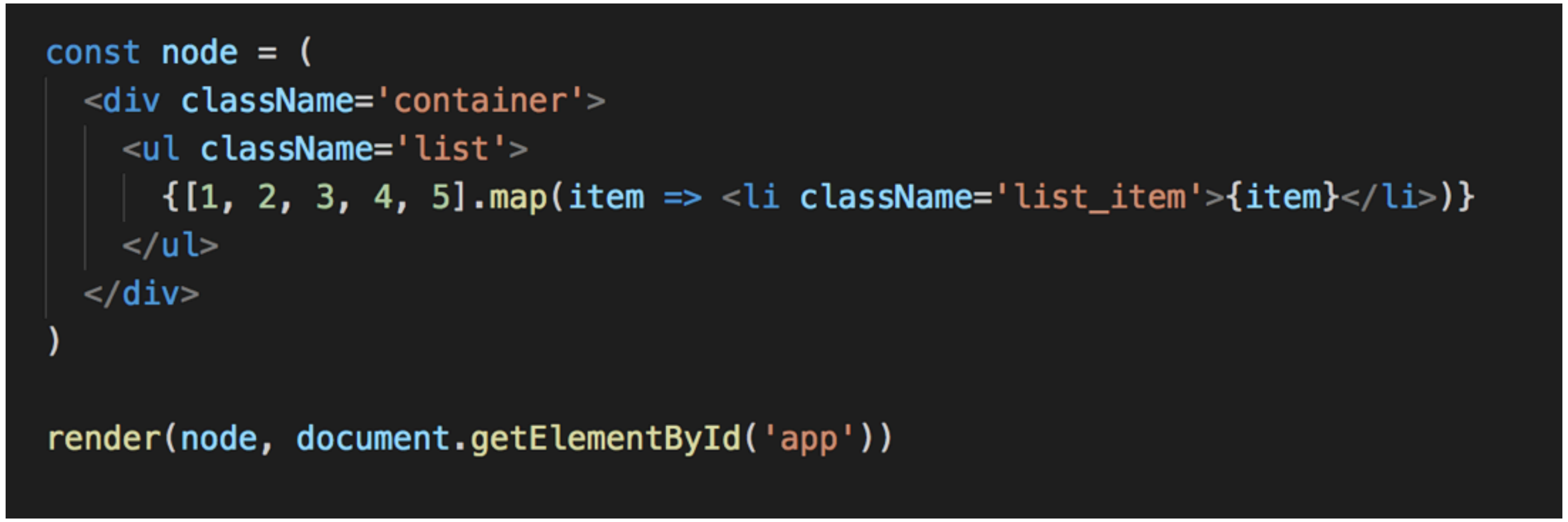
组件框架设计

一切皆有始

```
render(<div />, document.getElementById(‘app’))
        ↓ jsx编译
render(createElement(‘div’, null), document.getElementById(‘app’))
```

组件框架设计

一切皆有始

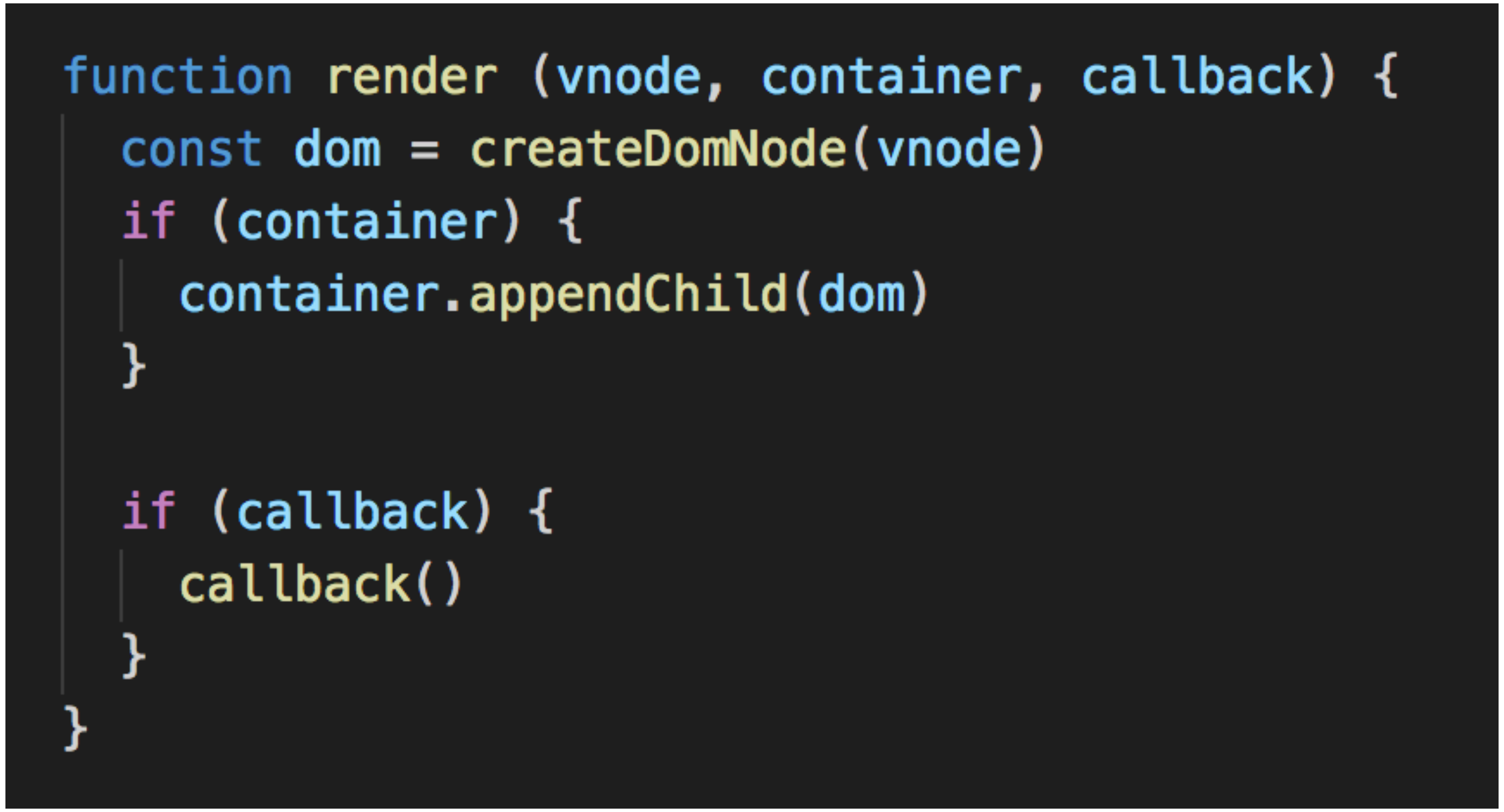


```
const node = (
  <div className='container'>
    <ul className='list'>
      {[1, 2, 3, 4, 5].map(item => <li className='list_item'>{item}</li>)}
    </ul>
  </div>
)

render(node, document.getElementById('app'))
```

渲染一个虚拟dom树到页面上

组件框架设计

一切皆有始

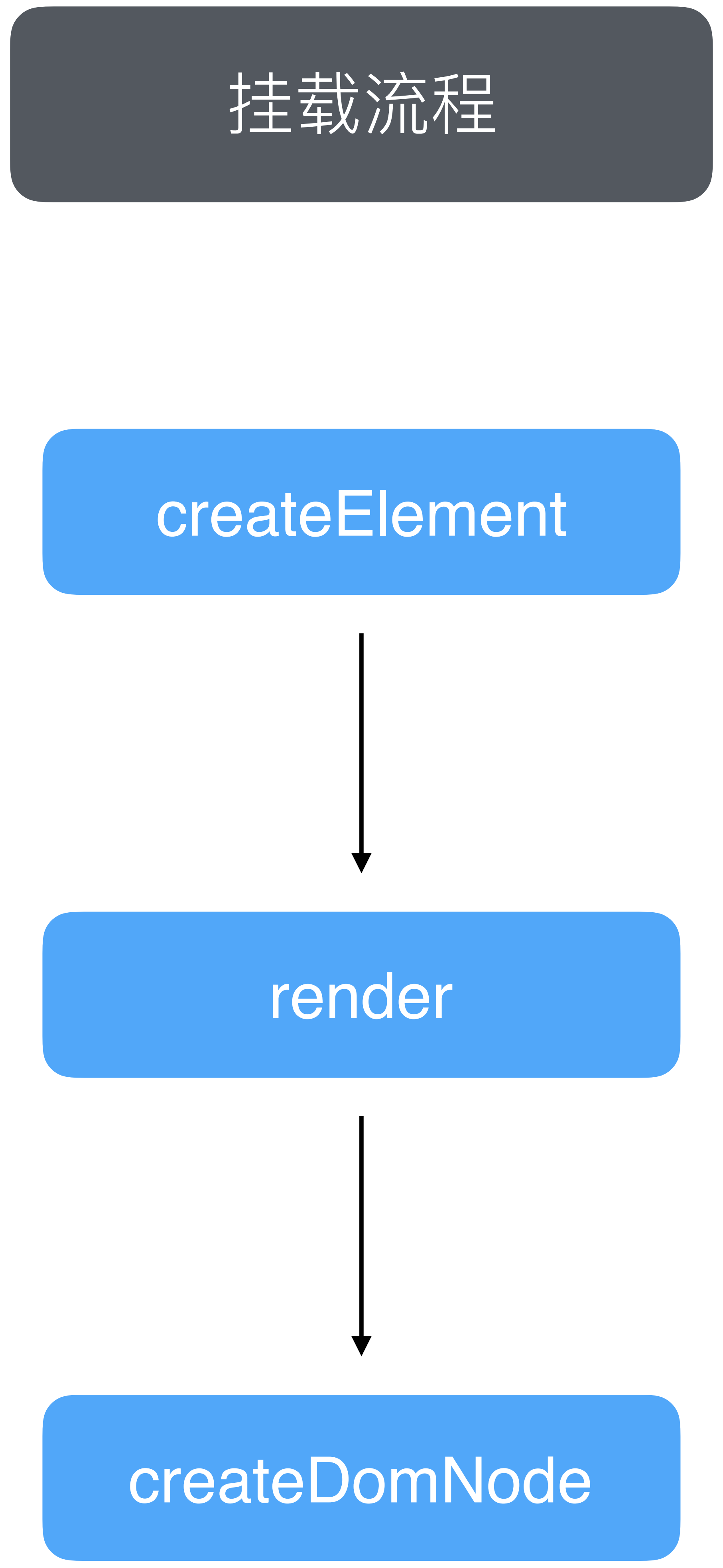


```
function render (vnode, container, callback) {
  const dom = createDomNode(vnode)
  if (container) {
    container.appendChild(dom)
  }

  if (callback) {
    callback()
  }
}
```

根据虚拟dom创建真实dom

将dom添加入容器内

组件框架设计



组件框架设计

组件?

组件框架设计

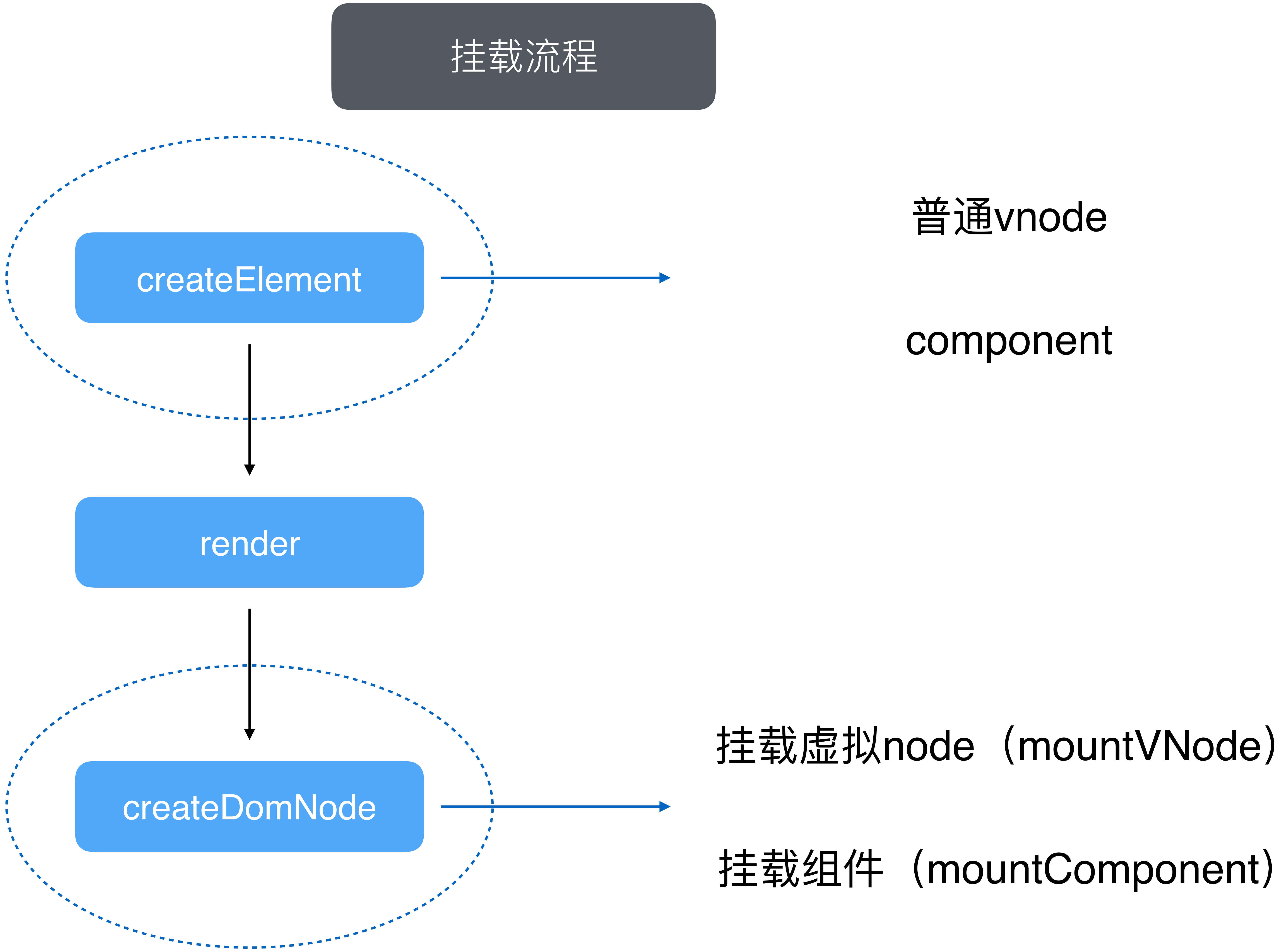
组件挂载

```
render(<List />, document.getElementById(‘app’))
```

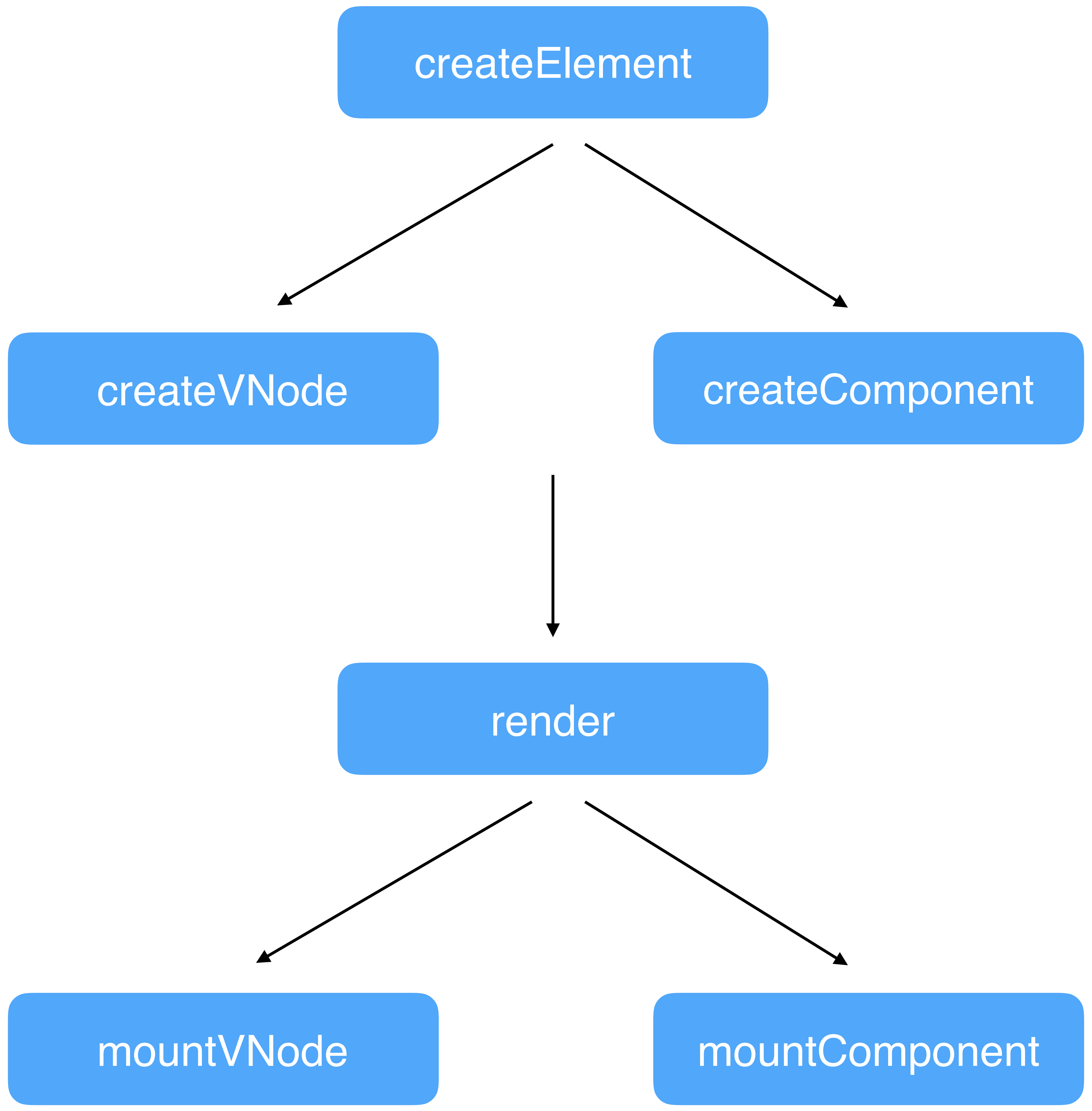
jsx编译 ↓

```
render(createElement(List, null), document.getElementById(‘app’))
```

组件框架设计

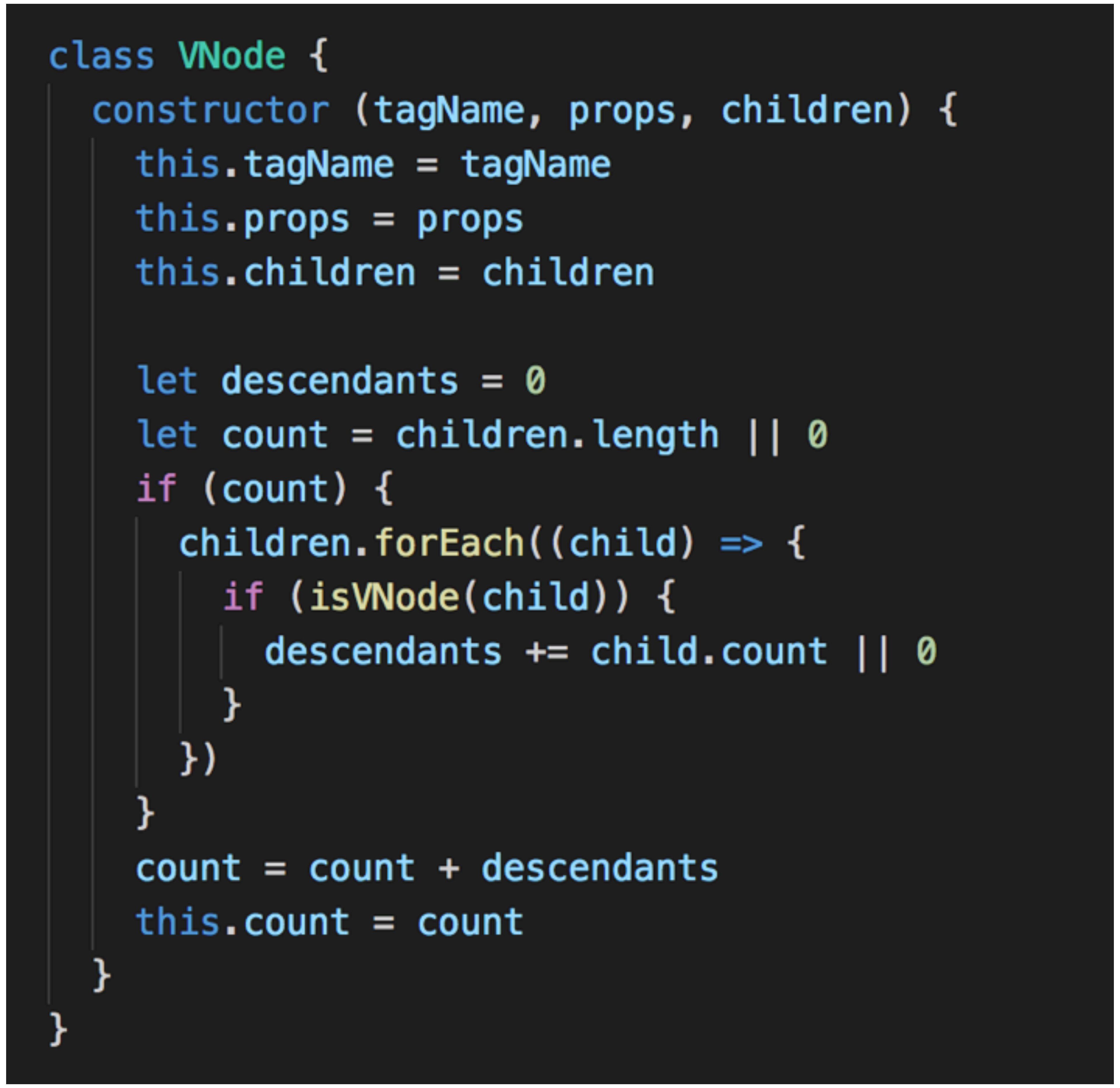

挂载流程
createElement
render
createDomNode
普通vnode
component
挂载虚拟node（mountVNode）
挂载组件（mountComponent）

组件框架设计


createElement
createVNode
createComponent
render
mountVNode
mountComponent

组件框架设计

VNODE



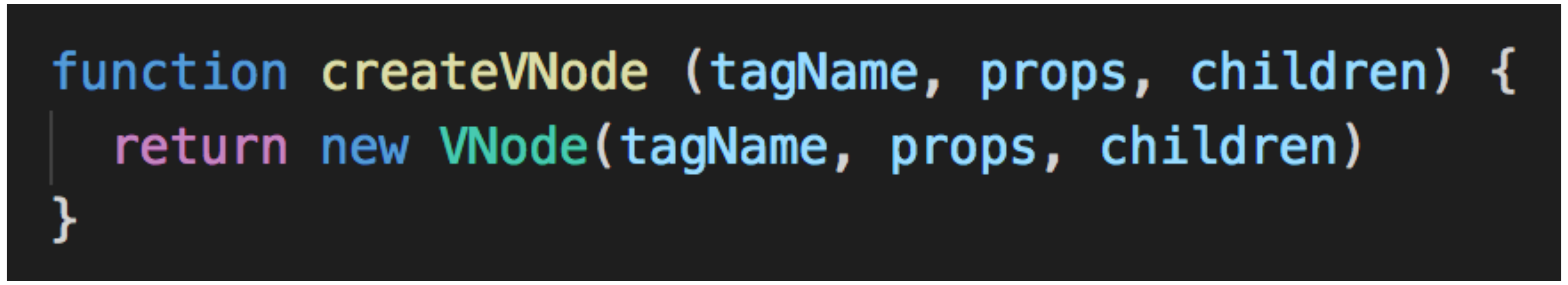
```
class VNode {
  constructor (tagName, props, children) {
    this.tagName = tagName
    this.props = props
    this.children = children

    let descendants = 0
    let count = children.length || 0
    if (count) {
      children.forEach((child) => {
        if (isVNode(child)) {
          descendants += child.count || 0
        }
      })
    }
    count = count + descendants
    this.count = count
  }
}
```

VNode类

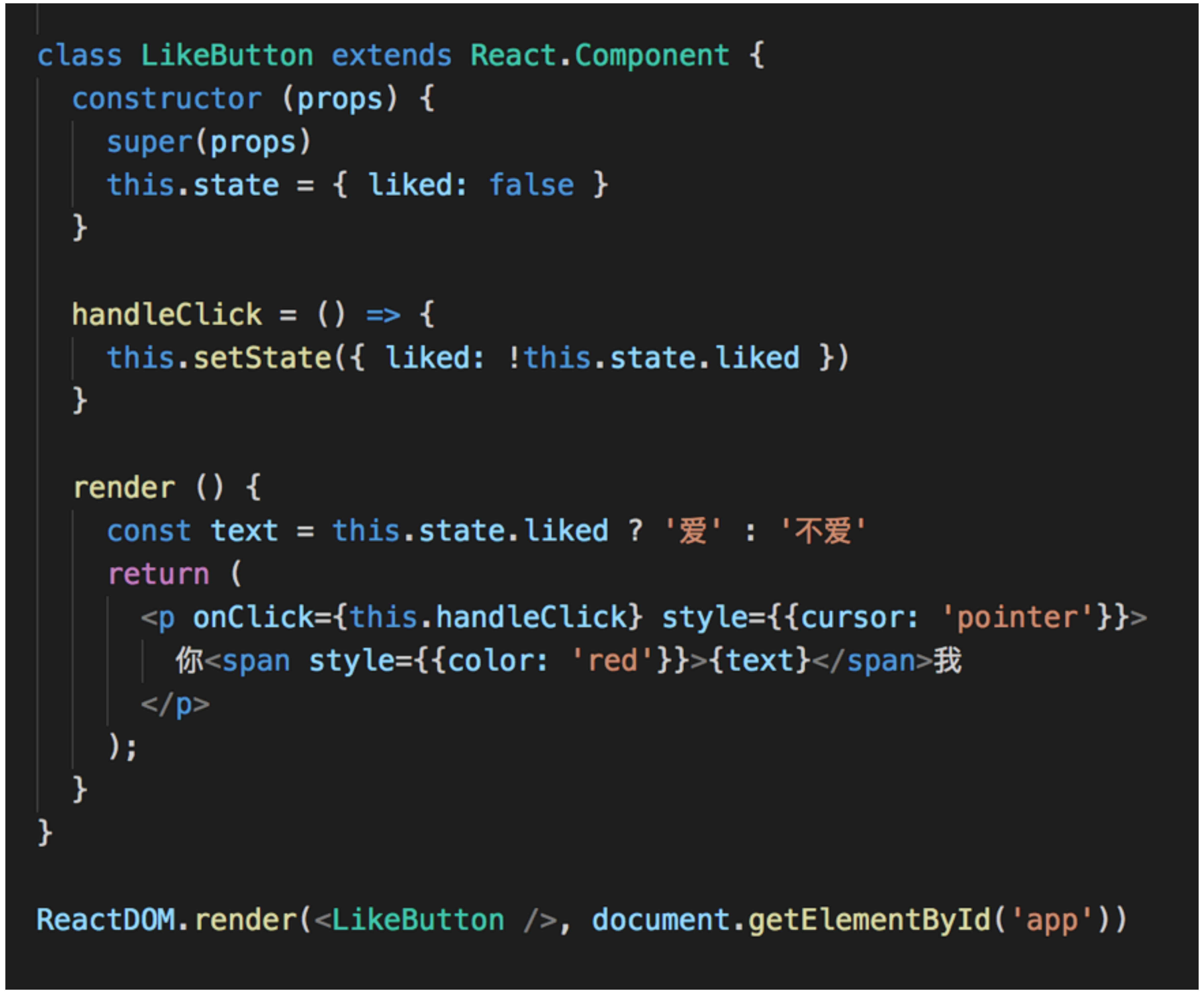
方便处理、扩展

组件框架设计

createVNode


```
function createVNode (tagName, props, children) {
  return new VNode(tagName, props, children)
}
```

组件框架设计

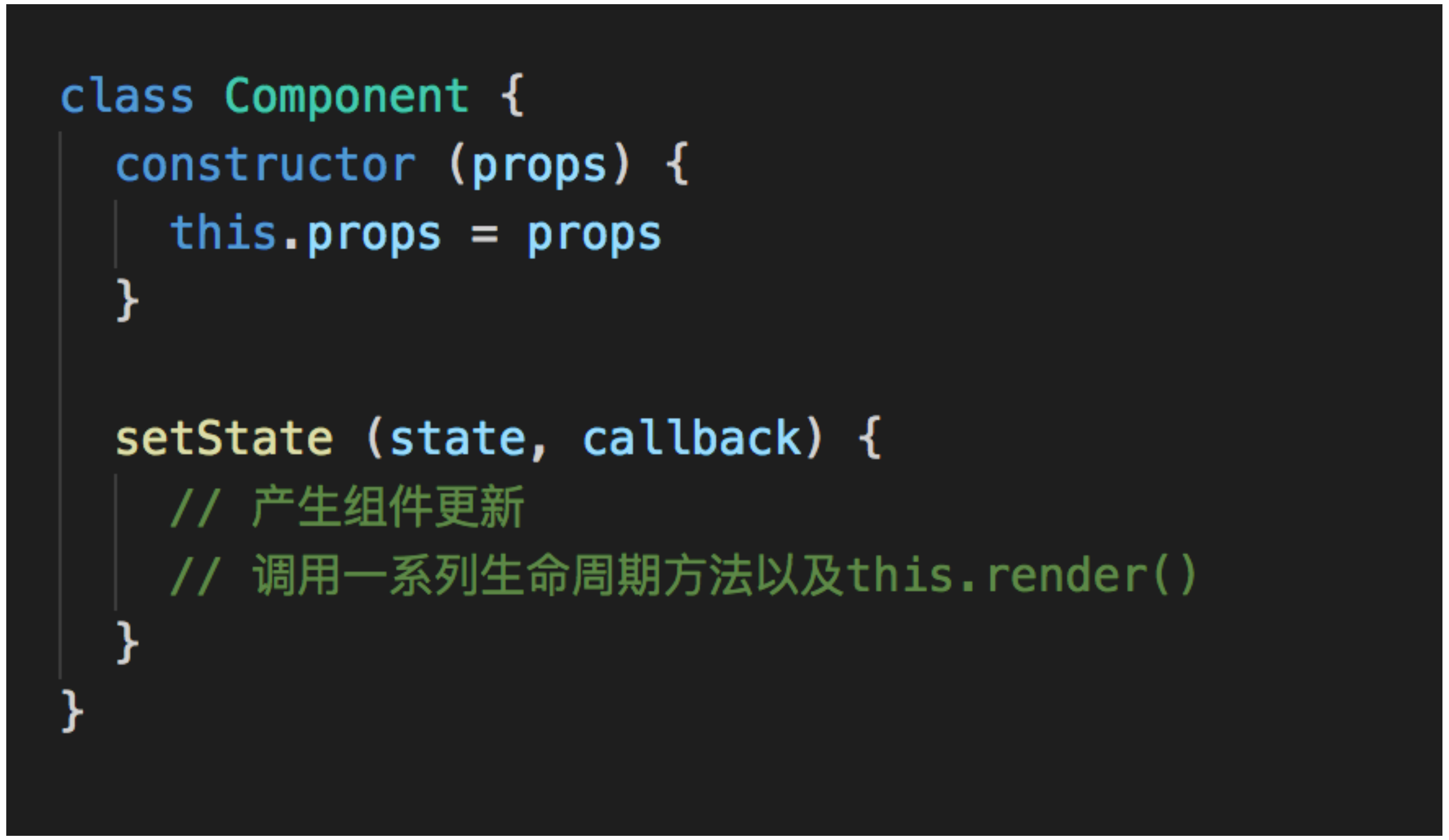


```
class LikeButton extends React.Component {
  constructor (props) {
    super(props)
    this.state = { liked: false }
  }

  handleClick = () => {
    this.setState({ liked: !this.state.liked })
  }

  render () {
    const text = this.state.liked ? '爱' : '不爱'
    return (
      <p onClick={this.handleClick} style={{cursor: 'pointer'}}>
        你<span style={{color: 'red'}}>{text}</span>我
      </p>
    );
  }
}

ReactDOM.render(<LikeButton />, document.getElementById('app'))
```

ES6写法React组件

组件框架设计

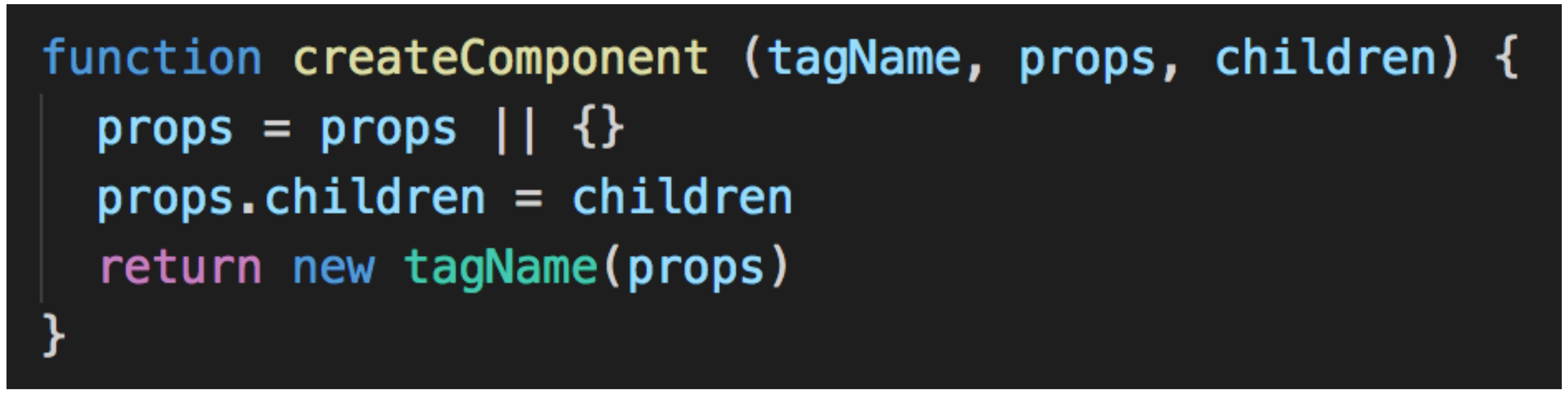
组件基类



```
class Component {
  constructor (props) {
    this.props = props
  }

  setState (state, callback) {
    // 产生组件更新
    // 调用一系列生命周期方法以及this.render()
  }
}
```

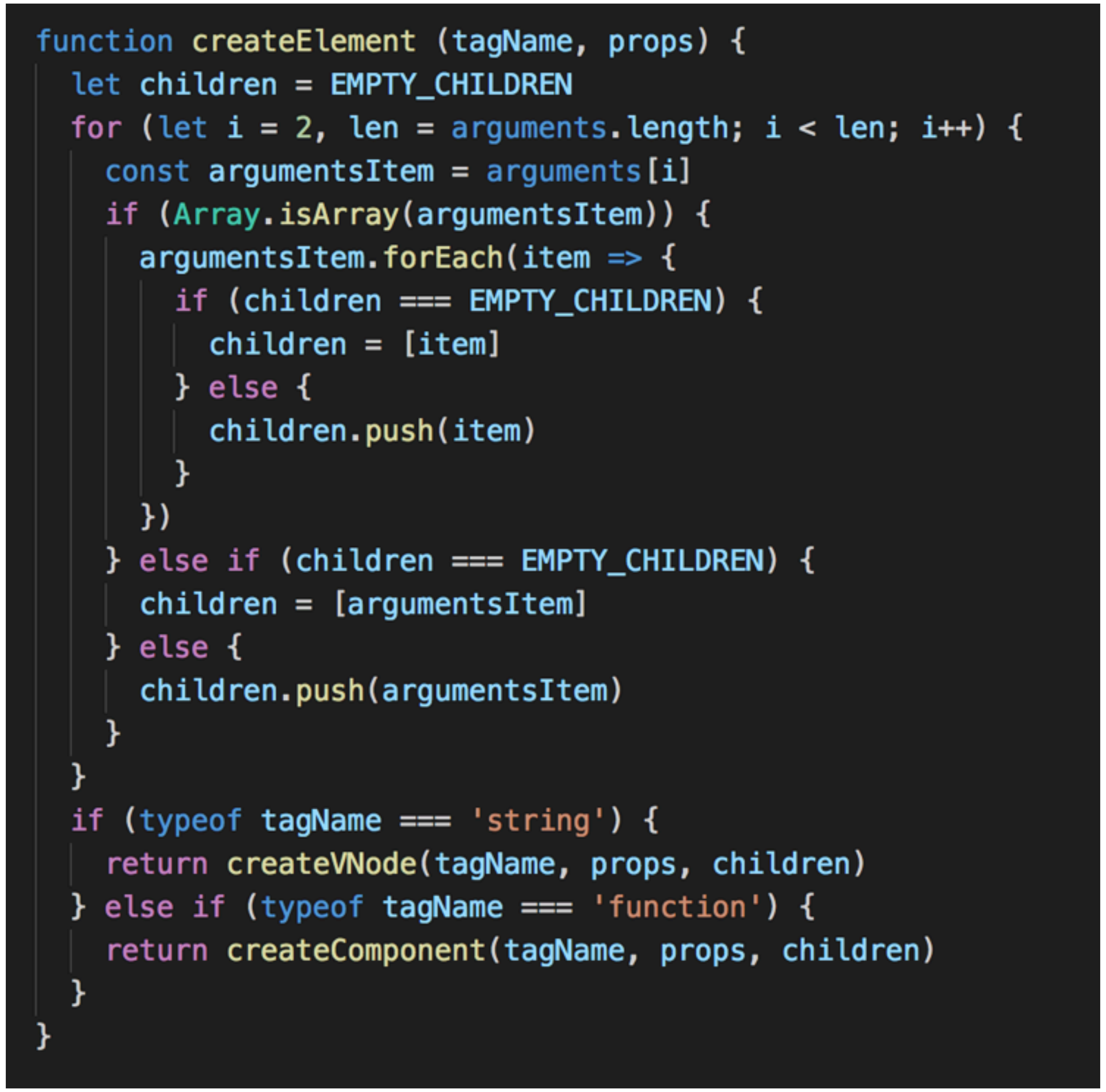
组件框架设计

createComponent


```
function createComponent (tagName, props, children) {
  props = props || {}
  props.children = children
  return new tagName(props)
}
```

组件框架设计

createElement



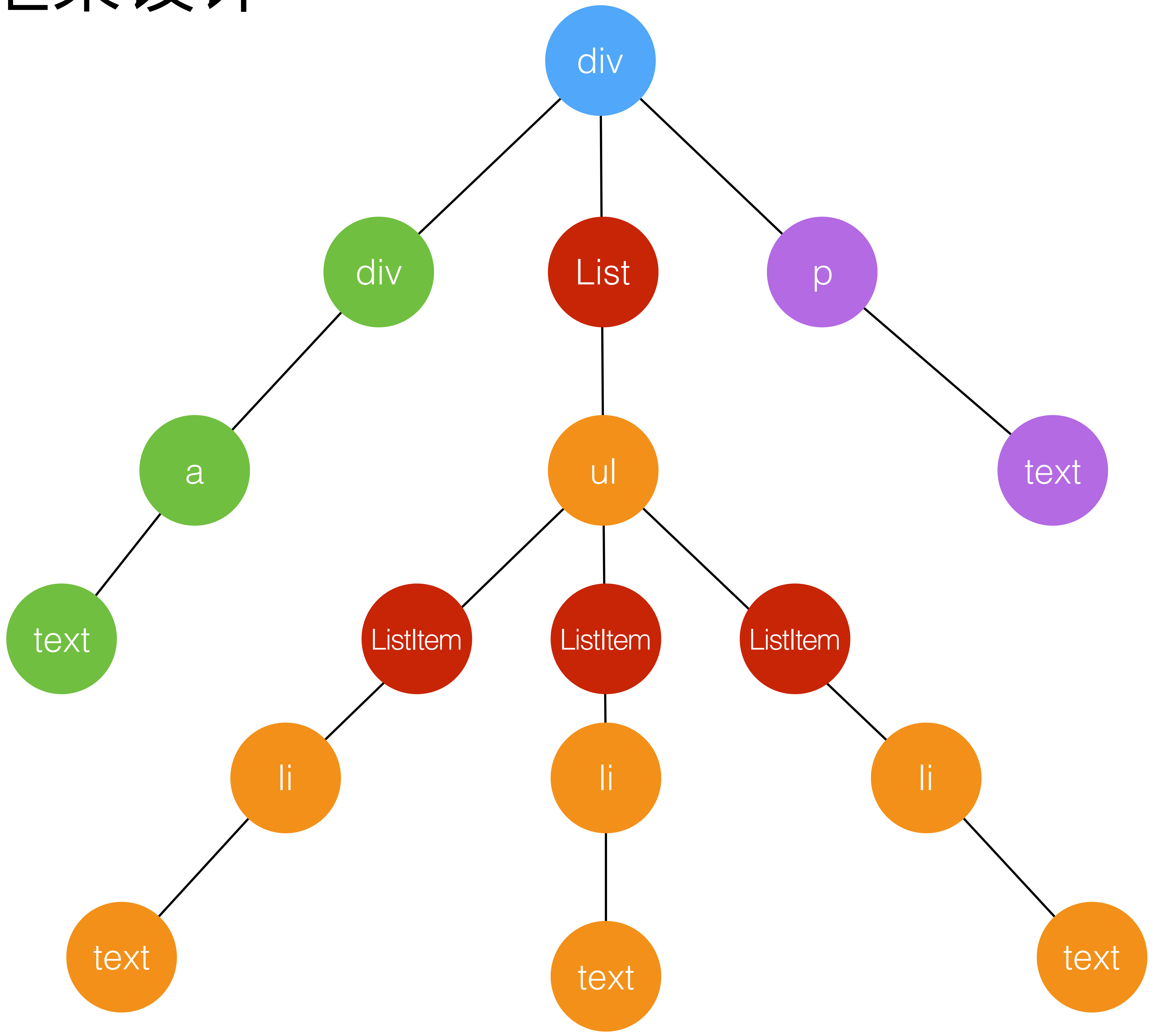
```
function createElement (tagName, props) {
  let children = EMPTY_CHILDREN
  for (let i = 2, len = arguments.length; i < len; i++) {
    const argumentsItem = arguments[i]
    if (Array.isArray(argumentsItem)) {
      argumentsItem.forEach(item => {
        if (children === EMPTY_CHILDREN) {
          children = [item]
        } else {
          children.push(item)
        }
      })
    } else if (children === EMPTY_CHILDREN) {
      children = [argumentsItem]
    } else {
      children.push(argumentsItem)
    }
  }
  if (typeof tagName === 'string') {
    return createVNode(tagName, props, children)
  } else if (typeof tagName === 'function') {
    return createComponent(tagName, props, children)
  }
}
```

createElement方法

tagName:String -> 创建虚拟node

tagName:Function -> 创建组件实例

组件框架设计



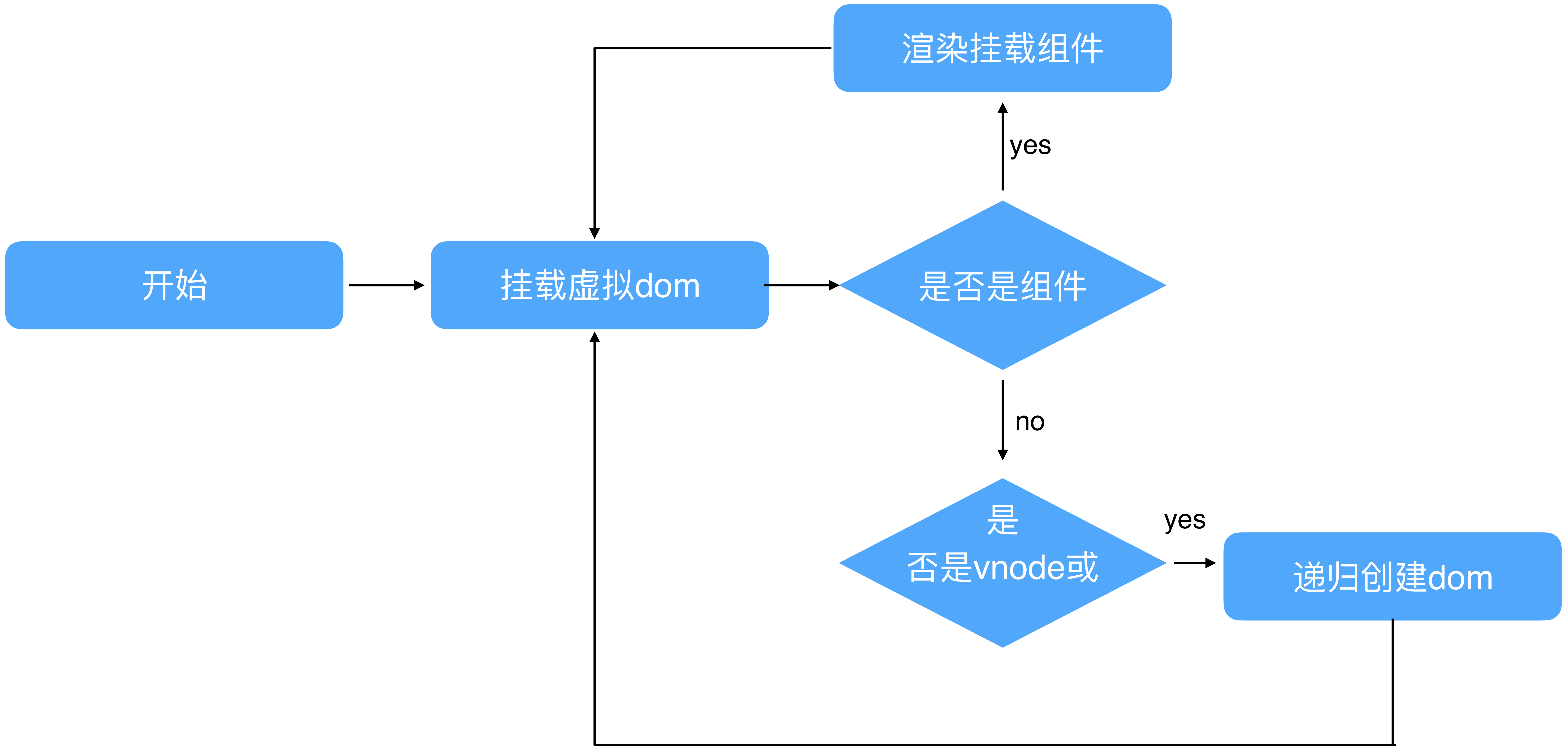
vnode节点包含组件

组件包含vnode节点

递归创建

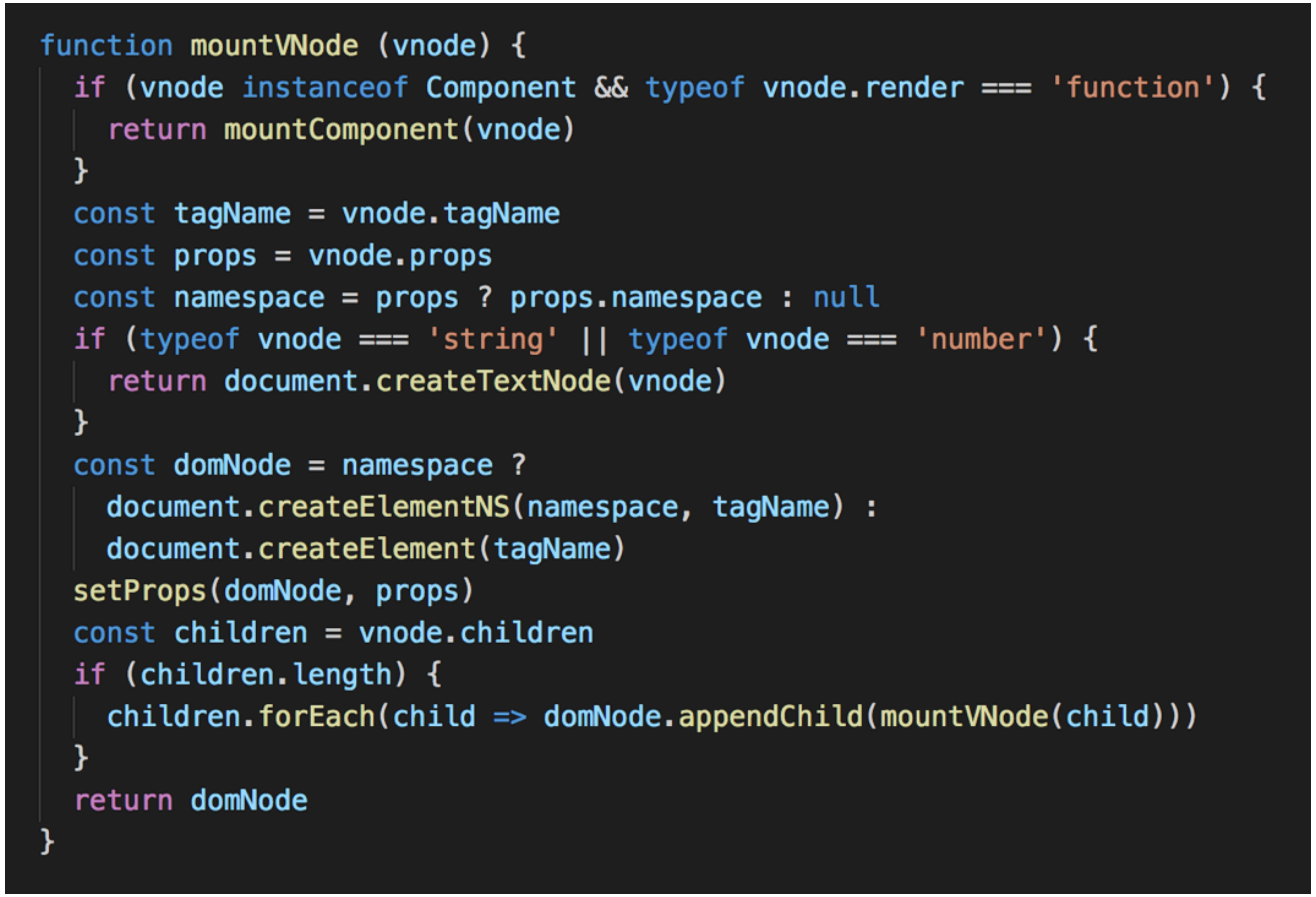
组件框架设计

挂载虚拟dom



组件框架设计

mountVNode



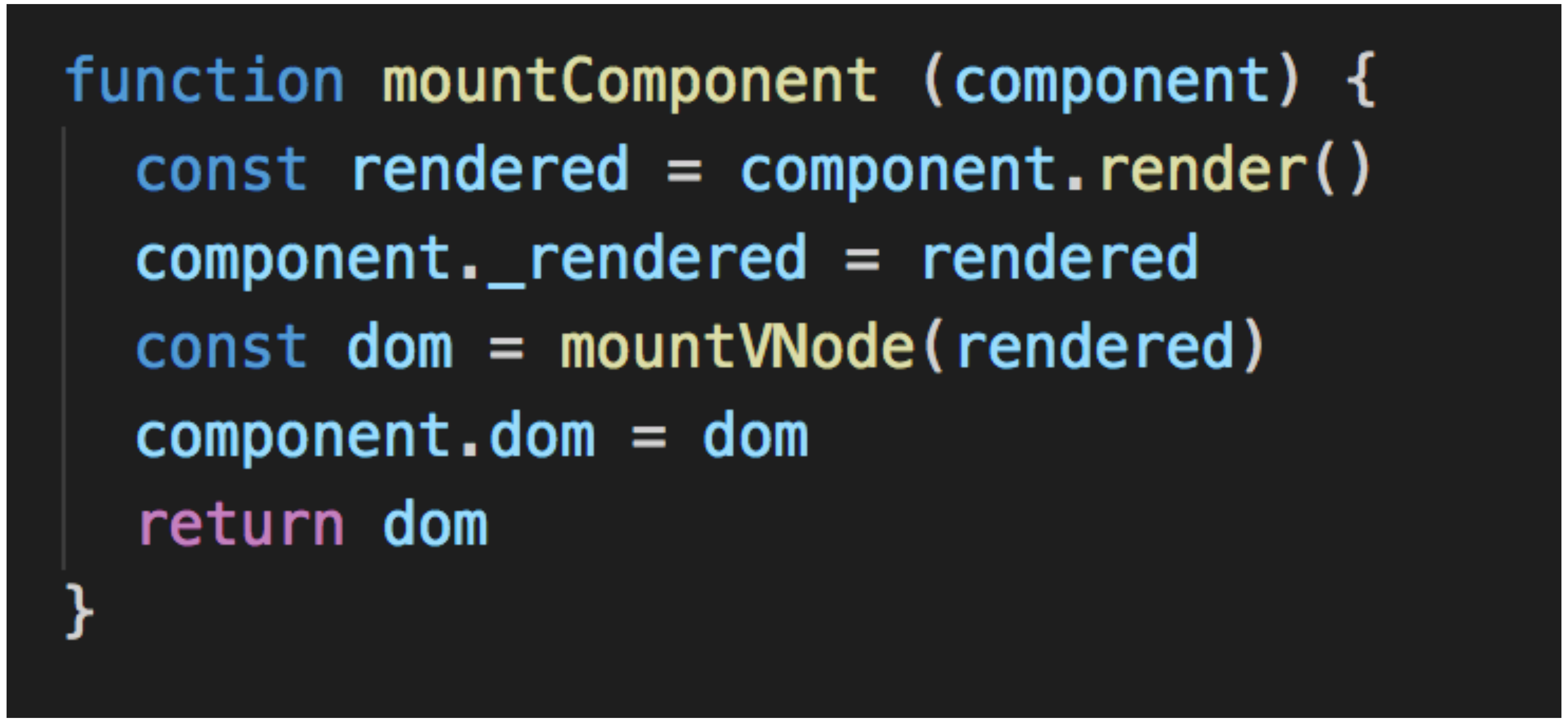
```
function mountVNode (vnode) {
  if (vnode instanceof Component && typeof vnode.render === 'function') {
    return mountComponent(vnode)
  }
  const tagName = vnode.tagName
  const props = vnode.props
  const namespace = props ? props.namespace : null
  if (typeof vnode === 'string' || typeof vnode === 'number') {
    return document.createTextNode(vnode)
  }
  const domNode = namespace ?
    document.createElementNS(namespace, tagName) :
    document.createElement(tagName)
  setProps(domNode, props)
  const children = vnode.children
  if (children.length) {
    children.forEach(child => domNode.appendChild(mountVNode(child)))
  }
  return domNode
}
```

挂载虚拟node

改写createDomNode

组件框架设计

mountComponent



```
function mountComponent (component) {
  const rendered = component.render()
  component._rendered = rendered
  const dom = mountVNode(rendered)
  component.dom = dom
  return dom
}
```

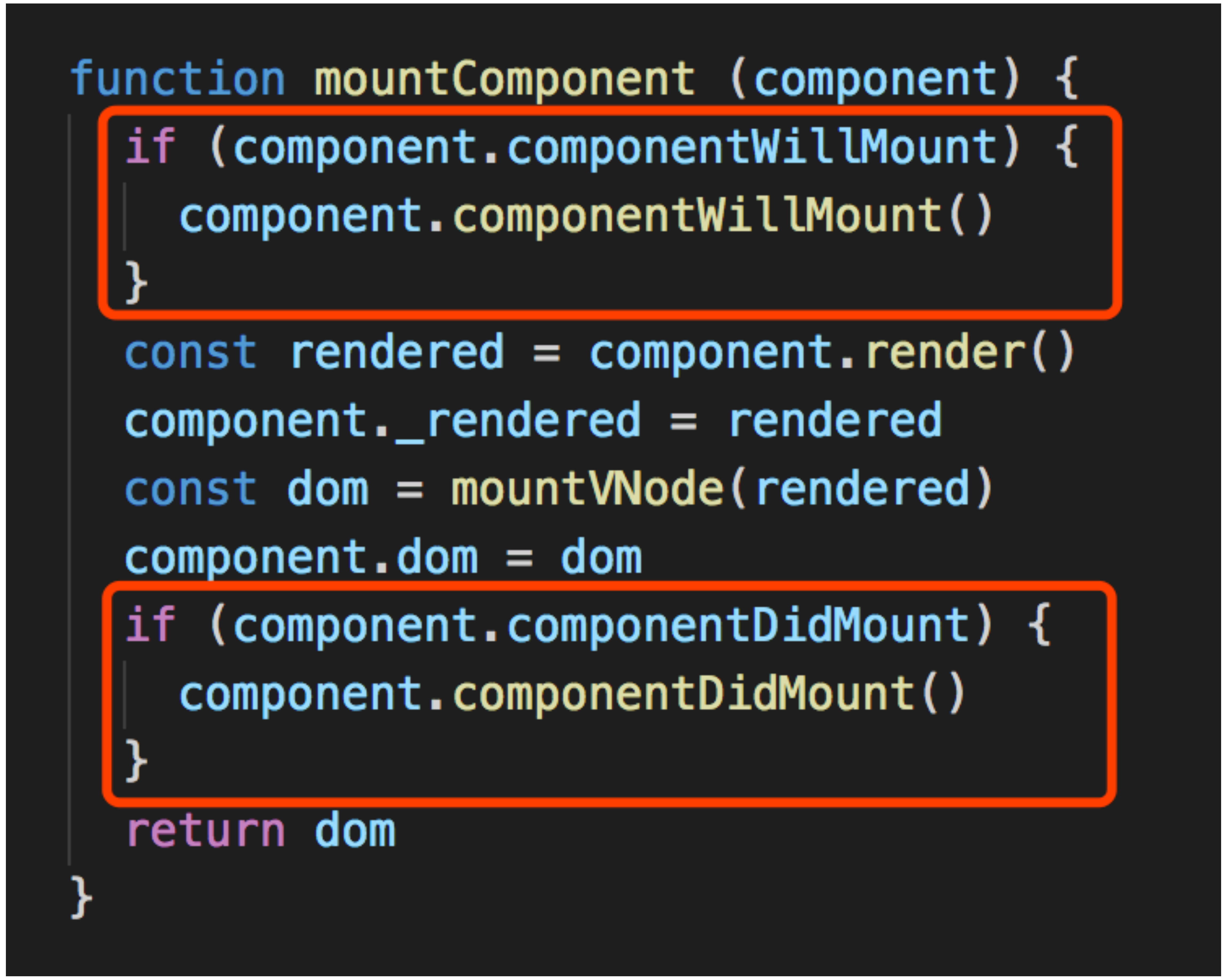
挂载组件

调用render方法得到虚拟dom

调用挂载虚拟node方法

组件框架设计

生命周期函数



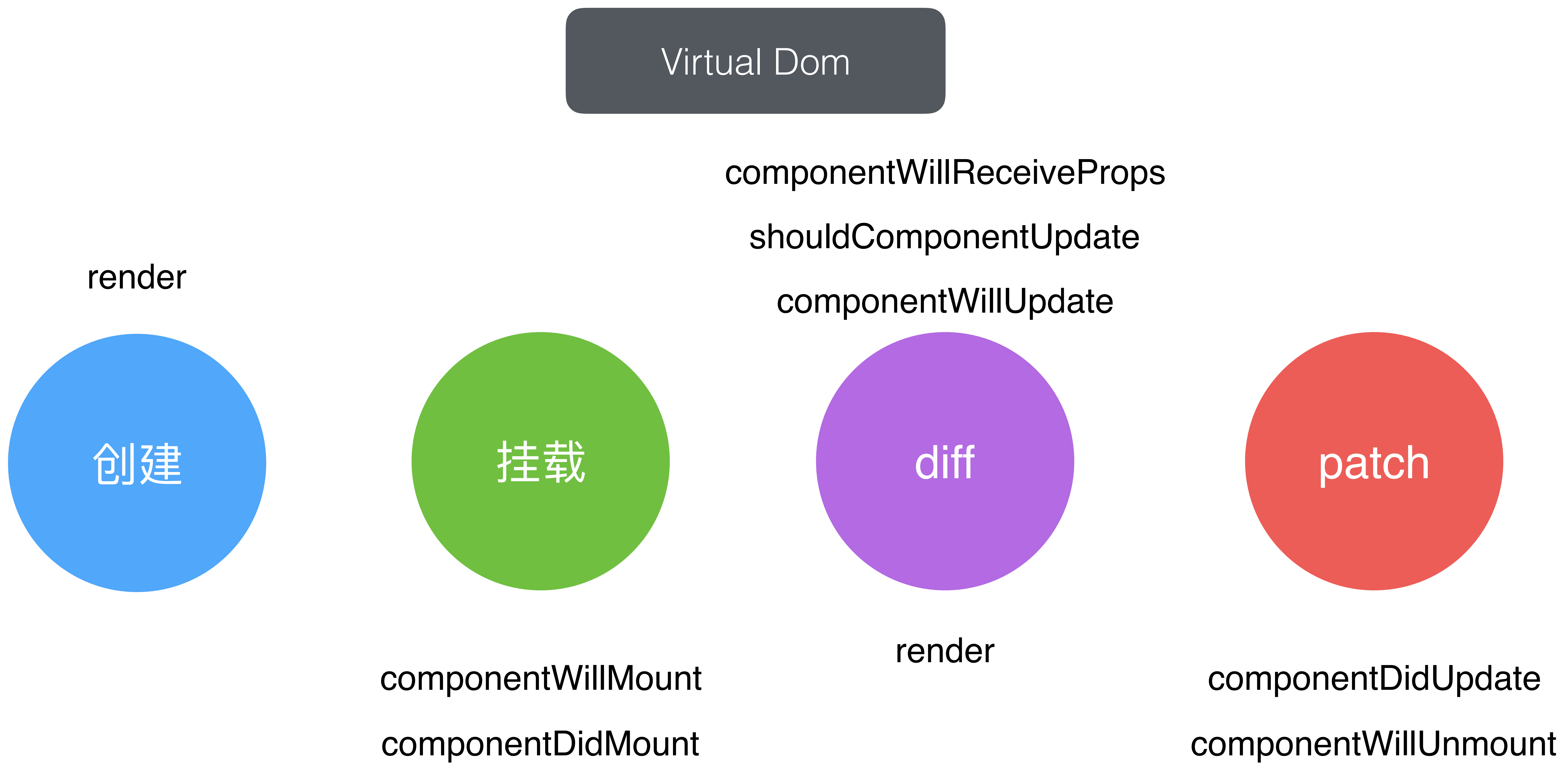
```
function mountComponent (component) {
  if (component.componentWillMount) {
    component.componentWillMount()
  }
  const rendered = component.render()
  component._rendered = rendered
  const dom = mountVNode(rendered)
  component.dom = dom
  if (component.componentDidMount) {
    component.componentDidMount()
  }
  return dom
}
```

组件框架设计

挂载流程

创建虚拟dom
render挂载
createElement
是否vnode
yes
createVNode
no
createComponent
渲染挂载组件
yes
挂载虚拟dom
是否是组件
no
是否是vnode或
yes
递归创建dom

组件框架设计


Virtual Dom
render
创建
挂载
componentWillMount
componentDidMount
componentWillReceiveProps
shouldComponentUpdate
componentWillUpdate
diff
render
patch
componentDidUpdate
componentWillUnmount

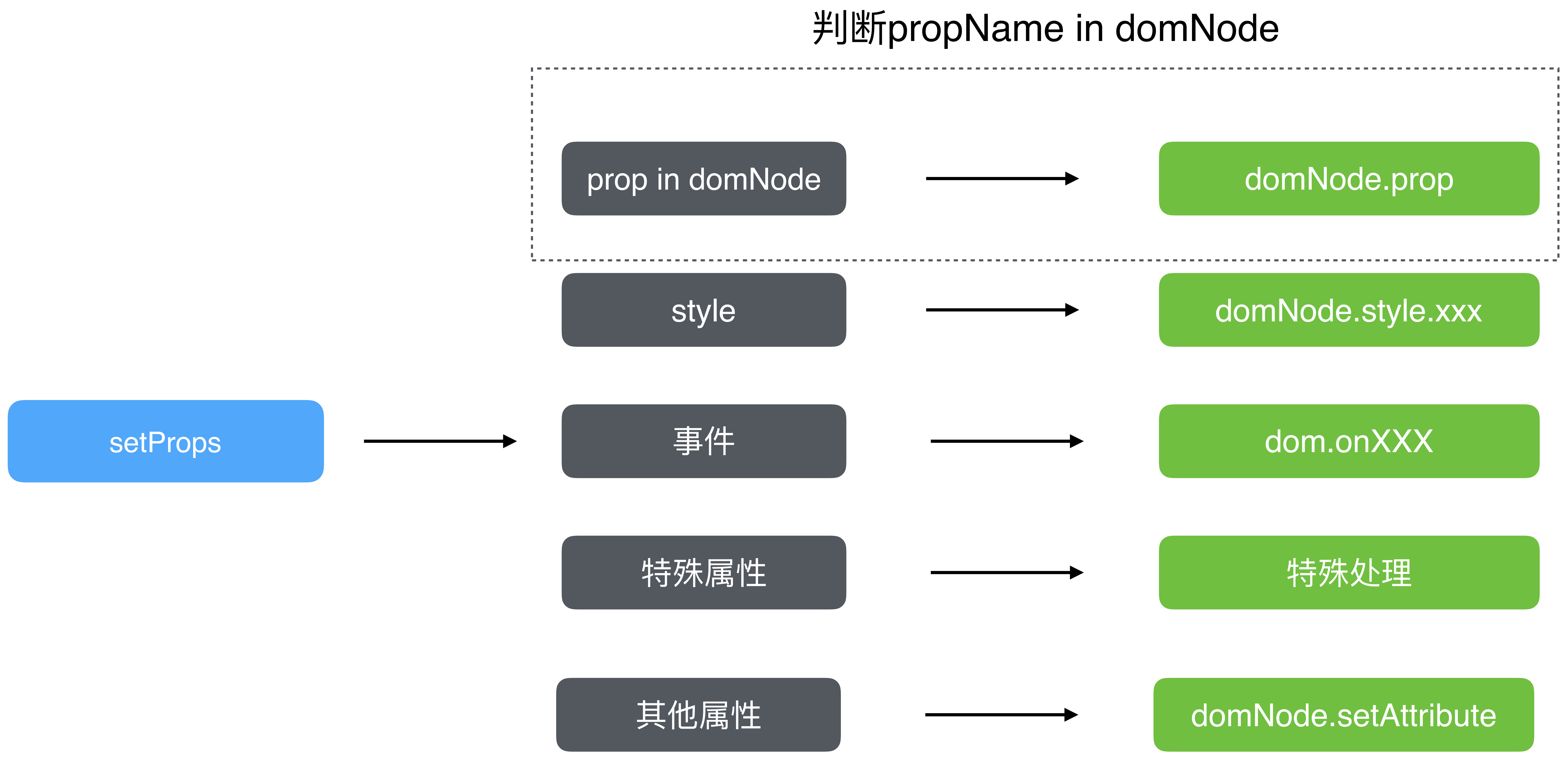
组件框架设计

组件更新设计请听下回分解

3

属性处理

属性处理


判断propName in domNode
setProps
prop in domNode
domNode.prop
style
domNode.style.xxx
事件
dom.onXXX
特殊属性
特殊处理
其他属性
domNode.setAttribute

属性处理

特殊属性

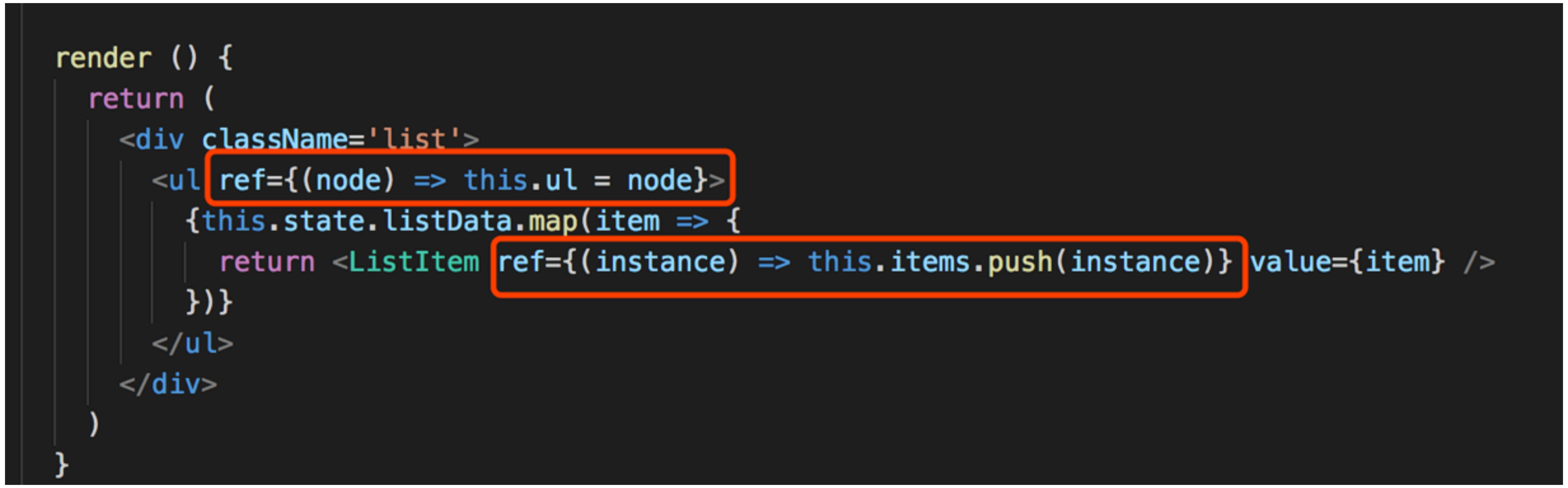
ref

dangerousSetInnerHTML

属性处理

ref

获取组件实例或dom元素



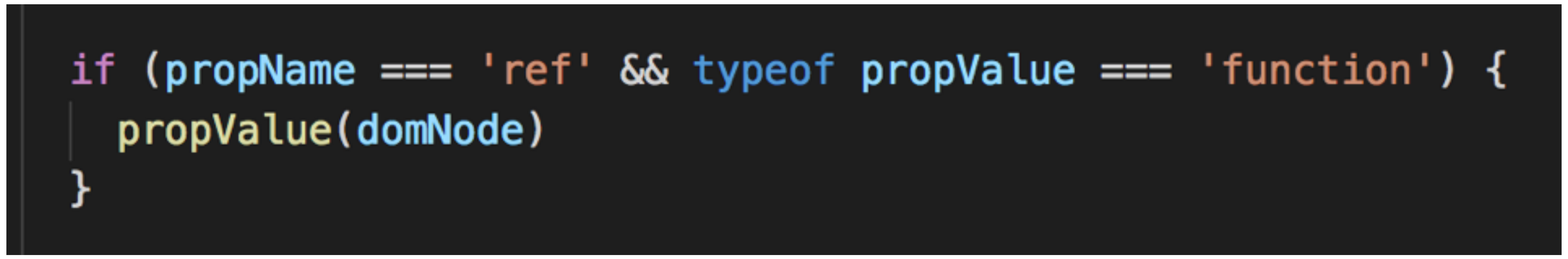
```
render () {
  return (
    <div className='list'>
      <ul ref={(node) => this.ul = node}>
        {this.state.listData.map(item => {
          return <ListItem ref={(instance) => this.items.push(instance)} value={item} />
        })}
      </ul>
    </div>
  )
}
```

ref最好为函数

属性处理

ref

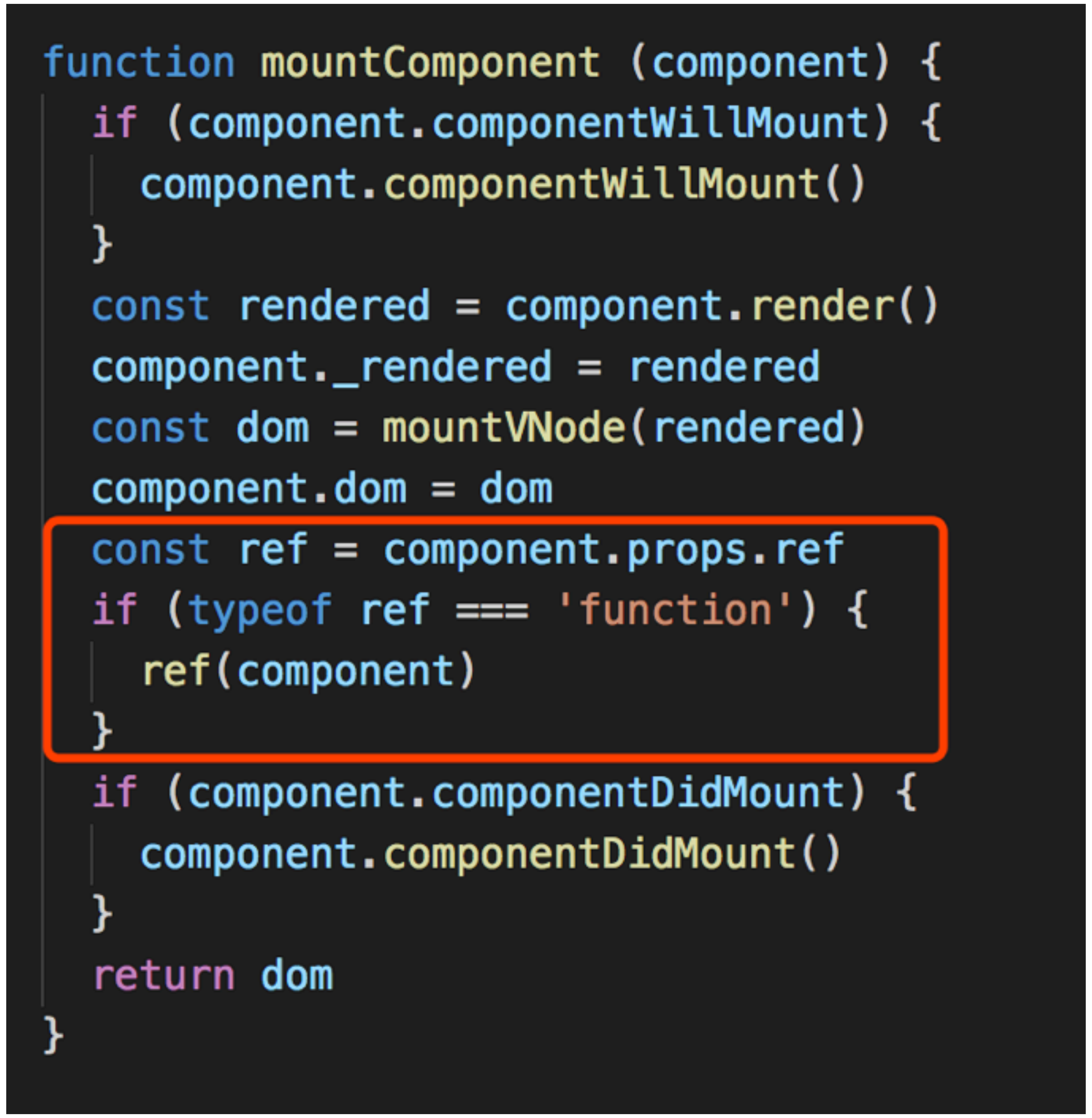
setProps



```
if (propName === 'ref' && typeof propValue === 'function') {
  propValue(domNode)
}
```

属性处理

ref



```
function mountComponent (component) {
  if (component.componentWillMount) {
    component.componentWillMount()
  }
  const rendered = component.render()
  component._rendered = rendered
  const dom = mountVNode(rendered)
  component.dom = dom
  const ref = component.props.ref
  if (typeof ref === 'function') {
    ref(component)
  }
  if (component.componentDidMount) {
    component.componentDidMount()
  }
  return dom
}
```

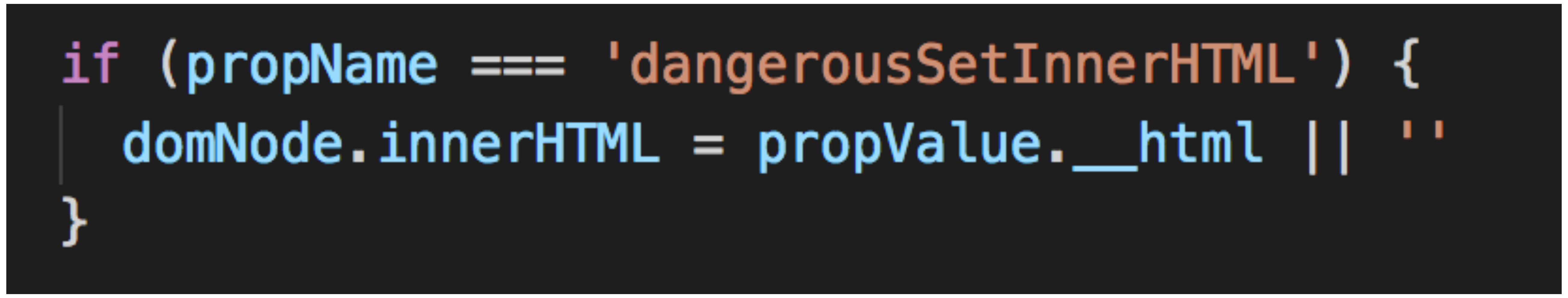
挂载组件

调用属性的ref方法

属性处理

dangerousSetInnerHTML

setProps



```
if (propName === 'dangerousSetInnerHTML') {
  domNode.innerHTML = propValue.__html || ''
}
```

课后作业

实现一个能一次性挂载渲染的组件系统

OPEN ORIENTED

凹 凸 实 验 室