React.js 小书

<-- 返回首页

挂载阶段的组件生命周期（二）

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson19

这一节我们来讨论一下对于一个组件来说，`constructor`、`componentWillMount`、`componentDidMount`、`componentWillUnmount` 这几个方法在一个组件的出生到死亡的过程里面起了什么样的作用。

一般来说，所有关于组件自身的状态的初始化工作都会放在 `constructor` 里面去做。你会发现本书所有组件的 `state` 的初始化工作都是放在 `constructor` 里面的。假设我们现在在做一个时钟应用：

我们会在 `constructor` 里面初始化 `state.date`，当然现在页面还是静态的，等下一会让时间动起来。

```
class Clock extends Component {
  constructor () {
    super()
    this.state = {
      date: new Date()
    }
  }

  render () {
    return (
      <div>
        <h1>
          <p>现在的时间是</p>
          {this.state.date.toLocaleTimeString()}
        </h1>
```

```
      </div>
    )
  }
}
```

一些组件启动的动作，包括像 Ajax 数据的拉取操作、一些定时器的启动等，就可以放在 `componentWillMount` 里面进行，例如 Ajax：

```
...
  componentWillMount () {
    ajax.get('http://json-api.com/user', (userData) => {
      this.setState({ userData })
    })
  }
...
```

当然在我们这个例子里面是定时器的启动，我们给 `Clock` 启动定时器：

```
class Clock extends Component {
  constructor () {
    super()
    this.state = {
      date: new Date()
    }
  }

  componentWillMount () {
    this.timer = setInterval(() => {
      this.setState({ date: new Date() })
    }, 1000)
  }
  ...
}
```

我们在 `componentWillMount` 中用 `setInterval` 启动了一个定时器：每隔 1 秒更新中的 `state.date`，这样页面就可以动起来了。我们用一个 `Index` 把它用起来，并且插入页面：

```
class Index extends Component {
  render () {
    return (
      <div>
        <Clock />
      </div>
    )
  }
}

ReactDOM.render(
  <Index />,
```

```
  document.getElementById('root')
)
```

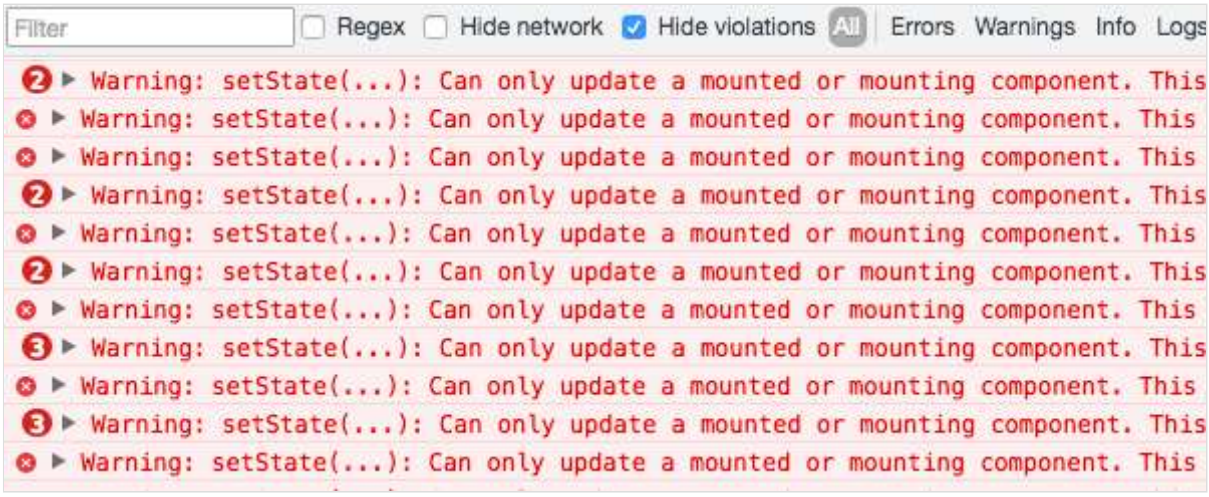
像上一节那样，我们修改这个 `Index` 让这个时钟可以隐藏或者显示：

```
class Index extends Component {
  constructor () {
    super()
    this.state = { isShowClock: true }
  }

  handleShowOrHide () {
    this.setState({
      isShowClock: !this.state.isShowClock
    })
  }

  render () {
    return (
      <div>
        {this.state.isShowClock ? <Clock /> : null }
        <button onClick={this.handleShowOrHide.bind(this)}>
          显示或隐藏时钟
        </button>
      </div>
    )
  }
}
```

现在页面上有个按钮可以显示或者隐藏时钟。你试一下显示或者隐藏时钟，虽然页面上看起来功能都正常，在控制台你会发现报错了：



这是因为，**当时钟隐藏的时候，我们并没有清除定时器**。时钟隐藏的时候，定时器的回调函数还在不停地尝试 `setState`，由于 `setState` 只能在已经挂载或者正在挂载的组件上调用，所以 React.js 开始疯狂报错。

多次的隐藏和显示会让 React.js 重新构造和销毁 `Clock` 组件，每次构造都会重新构建一个定时器。而销毁组件的时候没有清除定时器，所以你看到报错会越来越多。

而且因为 JavaScript 的闭包特性，这样会导致严重的内存泄漏。

这时候 `componentWillUnmount` 就可以派上用场了，它的作用就是在组件销毁的时候，做这种清场的工作。例如清除该组件的定时器和其他的数据清理工作。我们给 `Clock` 添加 `componentWillUnmount`，在组件销毁的时候清除该组件的定时器：

```
...
  componentWillUnmount () {
    clearInterval(this.timer)
  }
...
```

这时候就没有错误了。

总结

我们一般会把组件的 `state` 的初始化工作放在 `constructor` 里面去做；在 `componentWillMount` 进行组件的启动工作，例如 Ajax 数据拉取、定时器的启动；组件从页面上销毁的时候，有时候需要一些数据的清理，例如定时器的清理，就会放在 `componentWillUnmount` 里面去做。

说一下本节没有提到的 `componentDidMount` 。一般来说，有些组件的启动工作是依赖 DOM 的，例如动画的启动，而 `componentWillMount` 的时候组件还没挂载完成，所以没法进行这些启动工作，这时候就可以把这些操作放在 `componentDidMount` 当中。`componentDidMount` 的具体使用我们会在接下来的章节当中结合 DOM 来讲。

课后练习

- React.js 加载、刷新数据

> 因为第三方评论工具有问题，对本章节有任何疑问的朋友可以移步到 React.js 小书的论坛 发帖，我会回答大家的疑问。

下一节：更新阶段的组件生命周期

上一节：挂载阶段的组件生命周期（一）