**使用生成版本**

在部署应用时，请确认使用了生产版本。

### 单文件构建

### 我们提供压缩好的生产版本的 React 和 React DOM 文件:

<script src="https://unpkg.com/react@15/dist/react.min.js"></script>

<script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"></script>

### webpack

为了创建最高效的Webpack生产版本，需要在生产版本的配置中添加这些插件：

**使用 Chrome 性能分析工具 分析组件性能**

在 ****开发模式**** 中，你可以在支持相关功能的浏览器中使用性能工具来可视化组件 装载(mount) ，更新(update) 和 卸载(unmount) 的各个过程。例如：

**避免重新渲染**

React 构建并维护渲染 UI 的内部表示。它包括你从组件中返回的 React 元素。这些内部状态使得 React 只有在必要的情况下才会创建DOM节点和访问存在DOM节点，因为对 JavaScript 对象的操作是比 DOM 操作更快。这被称为”虚拟DOM”，React Native 也是基于上述原理。

**避免更新**

通过重写生命周期函数 shouldComponentUpdate 来优化性能，该函数会在重新渲染流程前触发。该函数的默认实现中返回的是 true，如果你的组件在部分场景下不需要更新，你可以在 shouldComponentUpdate 返回 false来跳过整个渲染流程，包括调用render() 和之后流程。

示例：通过 shouldComponentUpdate 函数来检查：

class CounterButton extends React.Component {

constructor(props) {

super(props);

this.state = {count: 1};

}

shouldComponentUpdate(nextProps, nextState) {

if (this.props.color !== nextProps.color) {

return true;

}

if (this.state.count !== nextState.count) {

return true;

}

// 返回false之后，render() 就不会被执行

return false;

}

render() {

return (

<button

color={this.props.color}

onClick={() => this.setState(state => ({count: state.count + 1}))}>

Count: {this.state.count}

</button>

);

}

}