# **[React context基本用法](https://www.cnblogs.com/mengff/p/9511419.html)**

React的context就是一个全局变量，可以从根组件跨级别在React的组件中传递。React context的API有两个版本，React16.x之前的是  
老版本的context，之后的是新版本的context。

**1.老版本的context**

getChildContext 根组件中声明，一个函数，返回一个对象，就是context  
childContextTypes 根组件中声明，指定context的结构类型，如不指定，会产生错误  
contextTypes 子孙组件中声明，指定要接收的context的结构类型，可以只是context的一部分结构。contextTypes 没有定义，context将是一个空对象。  
this.context 在子孙组件中通过此来获取上下文  
(注:从React v15.5开始 ，React.PropTypes 助手函数已被弃用，可使用 prop-types 库 来定义contextTypes)

举例如下：

[IMG_256](https://www.cnblogs.com/mengff/p/javascript:void(0);)

//根组件class MessageList extends React.Component {

getChildContext() {

return {color: "purple",text: "item text"};

}

render() {

const children = this.props.messages.map((message) =>

<Message text={message.text} /> );

return <div>{children}</div>; }

}

MessageList.childContextTypes = {

color: React.PropTypes.string

text: React.PropTypes.string

};

//中间组件class Message extends React.Component {

render() {

return (

<div>

<MessageItem />

<Button>Delete</Button>

</div> );

}

}

//孙组件(接收组件)class MessageItem extends React.Component {

render() {

return (

<div>

{this.props.text}

</div> );

}

}

MessageItem.contextTypes = {

text: React.PropTypes.string

};

class Button extends React.Component {

render() {

return (

<button style={{background: this.context.color}}>

{this.props.children}

</button> );

}

}

Button.contextTypes = {

color: React.PropTypes.string

};

[IMG_257](https://www.cnblogs.com/mengff/p/javascript:void(0);)

**2.新版本的context**

新版本的React context使用了Provider和Customer模式，和redux-react的模式非常像。在顶层的Provider中传入value，  
在子孙级的Consumer中获取该值，并且能够传递函数，用来修改context，如下代码所示：

[IMG_258](https://www.cnblogs.com/mengff/p/javascript:void(0);)

//创建Context组件

const ThemeContext = React.createContext({

theme: 'dark',

toggle: () => {}, //向上下文设定一个回调方法});

//运行APPclass App extends React.Component {

constructor(props) {

super(props);

this.toggle = () => { //设定toggle方法，会作为context参数传递

this.setState(state => ({

theme:

state.theme === themes.dark

? themes.light

: themes.dark,

}));

};

this.state = {

theme: themes.light,

toggle: this.toggle,

};

}

render() {

return (

<ThemeContext.Provider value={this.state}> //state包含了toggle方法

<Content />

</ThemeContext.Provider> );

}

}

//中间组件function Content() {

return (

<div>

<Button />

</div> );

}

//接收组件function Button() {

return (

<ThemeContext.Consumer>

{({theme, toggle}) => (

<button

onClick={toggle} //调用回调

style={{backgroundColor: theme}}>

Toggle Theme

</button> )}

</ThemeContext.Consumer> );

}

[IMG_259](https://www.cnblogs.com/mengff/p/javascript:void(0);)

详细用法可以参考官方文档：https://react.docschina.org/docs/context.html#reactcreatecontext

**3. context在如下的生命周期钩子中可以使用**

constructor(props, context)  
componentWillReceiveProps(nextProps, nextContext)  
shouldComponentUpdate(nextProps, nextState, nextContext)  
componentWillUpdate(nextProps, nextState, nextContext)  
componentDidUpdate(prevProps, prevState, prevContext)

**4. 在无状态组件中可以通过参数传入**

[IMG_260](https://www.cnblogs.com/mengff/p/javascript:void(0);)

function D(props, context) {

return (

<div>{this.context.user.name}</div> );

}

D.contextTypes = {

user: React.PropTypes.object.isRequired

}

[IMG_261](https://www.cnblogs.com/mengff/p/javascript:void(0);)

**5. React context的局限性**

1. 在组件树中，如果中间某一个组件 ShouldComponentUpdate returning false 了，会阻碍 context 的正常传值，导致子组件无法获取更新。  
2. 组件本身 extends React.PureComponent 也会阻碍 context 的更新。

注意点：

1. Context 应该是唯一不可变的  
2. 组件只在初始化的时候去获取 Context

参考：https://www.tuicool.com/articles/nUryimf  
　　   https://segmentfault.com/a/1190000012575622