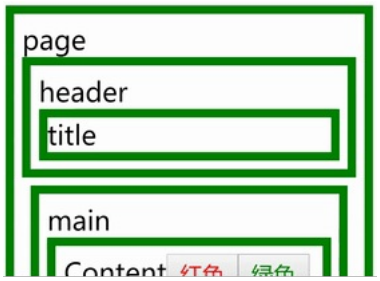
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=80 sentences=147, words=1416

## 1.Context(上下文) #

- 在某些场景下，你想在整个组件树中传递数据，但却不想手动地在每一层传递属性。你可以直接在 React 中使用强大的 `context`API解决上述问题
- 在一个典型的 React 应用中，数据是通过 props 属性自上而下（由父及子）进行传递的，但这种做法对于某些类型的属性而言是极其繁琐的（例如：地区偏好，UI 主题），这些属性是应用程序中许多组件都需要的。Context 提供了一种在组件之间共享此类值的方式，而不必显式地通过组件树的逐层传递 props




### 1.1 使用 #

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface PageProps {
}
interface PageState {
    color: string
}
interface ContextValue {
    color: string;
    changeColor: (color: string) => void
}
let ThemeContext = React.createContextnull>(null);
let root: HTMLElement | null = document.querySelector('#root');
class Header extends Component {
    render() {
        return (
            <ThemeContext.Consumer>
                {
                    (value: ContextValue | null) => (
                        <div style={{ border: `5px solid ${value!.color}`, padding: 5 }}>
                            header
                          <Title />
                        div>
                    )
                }
            ThemeContext.Consumer>
        )
    }
}
class Title extends Component {
    render() {
        return (
            <ThemeContext.Consumer>
                {
                    (value: ContextValue | null) => (
                        <div style={{ border: `5px solid ${value!.color}` }}>
                            title
                        div>
                    )
                }
            ThemeContext.Consumer>
        )
    }
}
class Main extends Component {
    render() {
        return (
            <ThemeContext.Consumer>
                {
                    (value: ContextValue | null) => (
                        <div style={{ border: `5px solid ${value!.color}`, margin: 5, padding: 5 }}>
                            main
                        <Content />
                        div>
                    )
                }
            ThemeContext.Consumer>
        )
    }
}
class Content extends Component {
    render() {
        return (
            <ThemeContext.Consumer>
                {
                    (value: ContextValue | null) => (
                        <div style={{ border: `5px solid ${value!.color}`, padding: 5 }}>
                            Content
                            <button onClick={() => value!.changeColor('red')} style={{ color: 'red' }}>红色button>
                            <button onClick={() => value!.changeColor('green')} style={{ color: 'green' }}>绿色button>
                        div>
                    )
                }
            ThemeContext.Consumer>

        )
    }
}

class Page extends Component<PageProps, PageState> {
    constructor(props: PageProps) {
        super(props);
        this.state = { color: 'red' };
    }
    changeColor = (color: string) => {
        this.setState({ color });
    }
    render() {
        let contextVal: ContextValue = { changeColor: this.changeColor, color: this.state.color };
        return (
            <ThemeContext.Provider value={contextVal}>
                <div style={{ margin: '10px', border: `5px solid ${this.state.color}`, padding: 5, width: 200 }}>
                    page
                    <Header />
                    <Main />
                div>
            ThemeContext.Provider>

        )
    }
}
ReactDOM.render(<Page />, root);
```

### 1.2 实现 #

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface PageProps {
}
interface PageState {
    color: string
}
+interface ContextValue {
+    color: string;
+    changeColor: (color: string) => void
+}
+interface ContextProps {
+    value: T
+}
+function createContext() {
+    let value;
+    function Provider(props) {
+        value = props.value;
+        Provider.value = value;
+        return props.children;
+    }
+    function Consumer(props) {
+        return props.children(value);;
+    }
+    return {
+        Provider,
+        Consumer
+    }
+}
+let ThemeContext = createContext(null);
let root: HTMLElement | null = document.querySelector('#root');
class Header extends Component {
    render() {
        return (

                {
                    (value: ContextValue | null) => (

                            header

                    )
                }

        )
    }
}
class Title extends Component {
    render() {
        return (

                {
                    (value: ContextValue | null) => (

                            title

                    )
                }

        )
    }
}
class Main extends Component {
    render() {
        return (

                {
                    (value: ContextValue | null) => (

                            main

                    )
                }

        )
    }
}
class Content extends Component {
    render() {
        return (

                {
                    (value: ContextValue | null) => (

                            Content
                             value!.changeColor('red')} style={{ color: 'red' }}>红色
                             value!.changeColor('green')} style={{ color: 'green' }}>绿色

                    )
                }

        )
    }
}

class Page extends Component {
    constructor(props: PageProps) {
        super(props);
        this.state = { color: 'red' };
    }
    changeColor = (color: string) => {
        this.setState({ color });
```

```
    }
    render() {
        let contextVal: ContextValue = { changeColor: this.changeColor, color: this.state.color };
        return (

                    page

        )
    }
}
ReactDOM.render(, root);
```

```
class Header extends Component {
+    static contextType = ThemeContext
    render() {
+        this.context = Header.contextType.Provider.value;
        return (

                Header

        )
    }
}
```

## 2. 高阶组件 #

- 高阶组件就是一个函数，传给它一个组件，它返回一个新的组件
- 高阶组件的作用其实就是为了组件之间的代码复用

```
const NewComponent = higherOrderComponent(OldComponent)
```

### 2.1 日志组件 #

```
import hoistNonReactStatics from 'hoist-non-react-statics';
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
const logger = (WrappedComponent: React.FC) => {
    class LoggerComponent extends Component {
        start: number | null = null;
        componentWillMount() {
            this.start = Date.now();
        }
        componentDidMount() {
            console.log((Date.now() - this.start!) + 'ms')
        }
        render() {
            return <WrappedComponent />
        }
    }
    hoistNonReactStatics(LoggerComponent, WrappedComponent);
    return LoggerComponent;
}
let Hello = logger((props) => <h1>hellohl>);
ReactDOM.render(<Hello />, document.getElementById('root'));
```

### 2.2 多层高阶组件 #

#### 2.2.1 从localStorage中加载 #

- localStorage中有 username=zhansan
- 从 localStorage中根据key加载对应的值

```
localStorage.setItem('username','zhangsan');
```

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface WrappedComponentProps {
    value: string;
}

const fromLocal = (WrappedComponent: React.FC | React.ComponentClass) => {
    interface FromLocalComponentProps {
        field: string
    }
    interface State {
        value: string;
    }
    class FromLocalComponent extends Component<FromLocalComponentProps, State> {
        constructor(props: FromLocalComponentProps) {
            super(props);
            this.state = { value: '' };
        }
        componentWillMount() {
            let value: string | null = localStorage.getItem(this.props.field);
            if (value)
                this.setState({ value });
        }
        render() {
            return <WrappedComponent value={this.state.value} />
        }
    }
    return FromLocalComponent;
}

const UserName = (props: WrappedComponentProps) => (
    <input defaultValue={props.value} />
)
const UserNameFromLocal = fromLocal(UserName);

ReactDOM.render(<UserNameFromLocal field="username" />, document.getElementById('root'));
```

#### 2.2.2 从ajax中加载 #

- 如果我们得到的用户名 zhangsan,但是要显示中文张三，需要包裹二次

- 包裹的时候是从内往外一层层包裹
- 渲染的时候是从外往内渲染

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface WrappedComponentProps {
    value: string;
}

+const fromLocal = (WrappedComponent: React.ComponentClass) => {
    interface FromLocalComponentProps { //最终返回的组件的属性对象
        field: string
    }
    interface State {  //状态对象
        value: string;
    }
    class FromLocalComponent extends Component {
        constructor(props: FromLocalComponentProps) {
            super(props);
            this.state = { value: '' };
        }
        componentWillMount() {
            let value: string | null = localStorage.getItem(this.props.field);
            if (value)
                this.setState({ value });
        }
        render() {
            return
        }
    }
    return FromLocalComponent;
}
+const fromAjax = (WrappedComponent: React.FC) => {
+    interface FromAjaxComponentProps { //最终返回的组件的属性对象
+        value: string
+    }
+    interface State {
+        value: string;
+    }

+    class FromAjaxComponent extends Component {
+        constructor(props: WrappedComponentProps) {
+            super(props);
+            this.state = { value: '' };
+        }
+        componentDidMount() {
+            fetch(`/translate.json`).then(response => response.json()).then((data) => {
+                let value = data[this.props.value];
+                this.setState({ value });
+            });
+        }
+        render() {
+            return
+        }
+    }
+    return FromAjaxComponent;
+}
+const UserName = (props: WrappedComponentProps) => (
+
+)
+const UserNameFromAjax = fromAjax(UserName);
+const UserNameFromLocal = fromLocal(UserNameFromAjax);

ReactDOM.render(, document.getElementById('root'));
```

## 3. render props #

- render-props (https://zh-hans.reactjs.org/docs/render-props.html)
- `render prop` 是指一种在 React 组件之间使用一个值为函数的 prop 共享代码的简单技术
- 具有 render prop 的组件接受一个函数，该函数返回一个 React 元素并调用它而不是实现自己的渲染逻辑
- render prop 是一个用于告知组件需要渲染什么内容的函数 prop
- 这也是逻辑复用的一种方式

```
 (
  <h1>Hello {data.target}h1>
)}/>
```

### 3.1 原生实现 #

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface Props {

}
interface State {
    x: number;
    y: number;
}
class MouseTracker extends React.Component<Props, State> {
    constructor(props: Props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event: React.MouseEvent) => {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (
            <div onMouseMove={this.handleMouseMove}>
                <h1>移动鼠标!h1>
                <p>当前的鼠标位置是 ({this.state.x}, {this.state.y})p>
            div>
        );
    }
}
ReactDOM.render(<MouseTracker />, document.getElementById('root'));
```

### 3.2 children #

- children是一个渲染的方法

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';

interface State {
    x: number;
    y: number;
}
interface Props {
    children: (state: State) => React.ReactNode
}
class MouseTracker extends React.Component {
    constructor(props: Props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event: React.MouseEvent) => {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (

                {this.props.children(this.state)}

        );
    }
}
ReactDOM.render(
    {
        (props: State) => (
            <>
                移动鼠标!

                当前的鼠标位置是 ({props.x}, {props.y})
            </>
        )
    }
, document.getElementById('root'));
```

### 3.3 render属性 #

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface Props {
    render: (state: State) => React.ReactNode
}
interface State {
    x: number;
    y: number;
}
class MouseTracker extends React.Component<Props, State> {
    constructor(props: Props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event: React.MouseEvent) => {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (
            <div onMouseMove={this.handleMouseMove}>
                {this.props.render(this.state)}
            div>
        );
    }
}

ReactDOM.render(< MouseTracker render={params => (
    <>
        <h1>移动鼠标!h1>
        <p>当前的鼠标位置是 ({params.x}, {params.y})p>
    </>
)} />, document.getElementById('root'));
```

### 3.4 HOC #

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
interface Props {
    render: (state: State) => React.ReactNode
}
interface State {
    x: number;
    y: number;
}
class MouseTracker extends React.Component {
    constructor(props: Props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event: React.MouseEvent) => {
        this.setState({
            x: event.clientX,
            y: event.clientY
        });
    }

    render() {
        return (

                {this.props.render(this.state)}

        );
    }
}
+function withMouse(Component: React.FC) {
+    return (
+        (props: State) =>  } />
+    )
+}
+let App = withMouse((props: State) => (
+    <>
+        移动鼠标!

+        当前的鼠标位置是 ({props.x}, {props.y})
+    </>
+));
ReactDOM.render(, document.getElementById('root'));
```