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=73 sentences=168, words=1304

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 类组件使用

```
import React from 'react';
import ReactDOM from 'react-dom';
let ThemeContext = React.createContext();

class Title extends React.Component {
    static contextType = ThemeContext
    render() {
        return (
            <div style={{ border: `5px solid ${this.context.color}` }}>
                Title
            div>
        )
    }
}
class Header extends React.Component {
    static contextType = ThemeContext
    render() {
        return (
            <div style={{ border: `5px solid ${this.context.color}` }}>
                Header
                <Title />
            div>
        )
    }
}
class Content extends React.Component {
    static contextType = ThemeContext
    render() {
        return (
            <div style={{ border: `5px solid ${this.context.color}` }}>
                Content
                <button onClick={() => this.context.changeColor('red')}>变红button>
                <button onClick={() => this.context.changeColor('green')}>变绿button>
            div>
        )
    }
}
class Main extends React.Component {
    static contextType = ThemeContext
    render() {
        return (
            <div style={{ border: `5px solid ${this.context.color}` }}>
                Main
                <Content />
            div>
        )
    }
}
class Panel extends React.Component {
    state = { color: 'green' }
    changeColor = (color) => {
        this.setState({ color });
    }
    render() {
        let value = { color: this.state.color, changeColor: this.changeColor };

        return (
            <ThemeContext.Provider value={value}>
                <div style={{ border: `5px solid ${this.state.color}`, width: 300 }}>
                    Panel
                    <Header />
                    <Main />
                div>
            ThemeContext.Provider>
        )
    }
}
ReactDOM.render(<Panel />, document.getElementById('root'));
```

1.2 函数组件使用

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
let ThemeContext = React.createContext('theme');

class Header extends Component {
    render() {
        return (
             <ThemeContext.Consumer>
                {
                    value => (
                        <div style={{ border: `5px solid ${value.color}`, padding: 5 }}>
                          header
                          <Title />
                        div>
                    )
                }
            ThemeContext.Consumer>
        )
    }
}
class Title extends Component {
    static contextType = ThemeContext;
    render() {
        return (
            <ThemeContext.Consumer>
                {
                    value => (
                       <div style={{border: `5px solid ${value.color}` }}>
                            title
                        div>
                    )
                }
            ThemeContext.Consumer>
        )
    }
}
class Main extends Component {
    static contextType = ThemeContext;
    render() {
        return (
            <ThemeContext.Consumer>
                {
                    value => (
                      <div style={{ border: `5px solid ${value.color}`, margin: 5, padding: 5 }}>
                        main
                        <Content />
                      div>
                    )
                }
            ThemeContext.Consumer>
        )
    }
}
class Content extends Component {
    static contextType = ThemeContext;
    render() {
        return (
            <ThemeContext.Consumer>
                {
                    value => (
                     <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 {
    constructor() {
        super();
        this.state = { color: 'red' };
    }
    changeColor = (color) => {
        this.setState({ color })
    }
    render() {
        let contextVal = {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 />, document.querySelector('#root'));
```

1.3 函数组件实现

```
function createContext() {
    let value;
    class Provider extends React.Component {
        constructor(props) {
            super(props);
            value = props.value
            this.state = {};
        }
        static getDerivedStateFromProps(nextProps, prevState) {
            value = nextProps.value;
            return {};
        }
        render() {
            return this.props.children;
        }
    }
    class Consumer extends React.Component {
        constructor(props) {
            super(props);
        }
        render() {
            return this.props.children(value);
        }
    }
    return {
        Provider,
        Consumer
    }
}
let ThemeContext = createContext('theme');
```

3. 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)
```

3.1 日志组件

3.1 手工实现

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
class App extends Component {
    componentWillMount() {
        this.start = Date.now();
    }
    componentDidMount() {
        console.log((Date.now() - this.start) + 'ms')
    }
    render() {
        return <div>Appdiv>
    }
}

ReactDOM.render(<App />, document.getElementById('root'));
```

3.2 高阶组件

```
import React,{Component} from 'react';
import ReactDOM from 'react-dom';
const logger = (WrappedComponent) => {
    class LoggerComponent extends Component {
      componentWillMount(){
          this.start = Date.now();
      }
      componentDidMount(){
          console.log((Date.now() - this.start)+'ms')
      }
      render () {
        return <WrappedComponent />
      }
    }
    return LoggerComponent;
  }
let Hello = logger(props=><h1>helloh1>);

ReactDOM.render(<Hello />, document.getElementById('root'));
```

3.2 多层高阶组件

3.2.1 从localStorage中加载

```
import React,{Component} from 'react';
import ReactDOM from 'react-dom';
const fromLocal = (WrappedComponent,name) =>{
    class NewComponent extends Component{
        constructor(){
            super();
            this.state = {value:null};
        }
        componentWillMount(){
            let value = localStorage.getItem(name);
             this.setState({value});
        }
        render(){
            return <WrappedComponent value={this.state.value}/>
        }
    }
    return NewComponent;
}
const UserName = ({value})=>(
    <input defaultValue = {value}/>
)
const UserNameFromLocal = fromLocal(UserName,'username');

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

3.2.2 从ajax中加载

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

const fromLocal = (WrappedComponent,name) =>{
    class NewComponent extends Component{
        constructor(){
            super();
            this.state = {id:null};
        }
        componentWillMount(){
            let id = localStorage.getItem(name);
            this.setState({id});
        }
        render(){
            return <WrappedComponent id={this.state.id}/>
        }
    }
    return NewComponent;
}
const fromAjax = (WrappedComponent) =>{
    class NewComponent extends Component{
        constructor(){
            super();
            this.state = {value:{}};
        }
        componentDidMount(){
            fetch(`/${this.props.id}.json`).then(response=>response.json()).then(value=>{
                this.setState({value});
            });
        }
        render(){
            return <WrappedComponent value={this.state.value}/>
        }
    }
    return NewComponent;
}
const UserName = ({value})=>{
  return <input defaultValue = {value.username}/>;
}

const UserNameFromAjax = fromAjax(UserName);
const UserNameFromLocal = fromLocal(UserNameFromAjax,'id');

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

translate.json

```
{
    "zhangsan": "张三"
}
```

4. 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>
)}/>
```

4.1 原生实现

```
class MouseTracker extends React.Component {
    constructor(props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

    handleMouseMove = (event) => {
        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>
        );
    }
}
```

4.2 children

```
class MouseTracker extends React.Component {
    constructor(props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

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

    render() {
        return (

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

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

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

4.3 render属性

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

    handleMouseMove = (event) => {
        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'));
```

4.4 HOC

```
class MouseTracker extends React.Component {
    constructor(props) {
        super(props);
        this.state = { x: 0, y: 0 };
    }

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

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

5. 插槽(Portals)

- Portals 提供了一种很好的方法，将子节点渲染到父组件 DOM 层次结构之外的 DOM 节点。

```
ReactDOM.createPortal(child, container)
```

- 第一个参数（child）是任何可渲染的 React 子元素，例如一个元素，字符串或 片段(fragment)
- 第二个参数（container）则是一个 DOM 元素

index.html

```
 <div id="modal-root">div>
```

index.js

```
import React,{Component} from 'react';
import ReactDOM from 'react-dom';
import './modal.css';

class Modal extends Component{
    constructor() {
        super();
        this.modal=document.querySelector('#modal-root');
    }
    render() {
        return ReactDOM.createPortal(this.props.children,this.modal);
    }
}
class Page extends Component{
    constructor() {
        super();
        this.state={show:false};
    }
    handleClick=() => {
        this.setState({show:!this.state.show});
    }
    render() {
        return (
            <div>
                <button onClick={this.handleClick}>显示模态窗口button>
                {
                    this.state.show&&<Modal>
                    <div id="modal" className="modal">
                        <div className="modal-content" id="modal-content">
                                内容
                                <button onClick={this.handleClick}>关闭button>
                        div>
                    div>
                Modal>
                }
            div>
        )
    }
}
ReactDOM.render(<Page/>,document.querySelector('#root'));
```

modal.css

```css
.modal{
    position: fixed;
    left:0;
    top:0;
    right:0;
    bottom:0;
    background: rgba(0,0,0,.5);
    display: block;
}

@keyframes zoom{
    from{transform:scale(0);}
    to{transform:scale(1);}
}

.modal .modal-content{
    width:50%;
    height:50%;
    background: white;
    border-radius: 10px;
    margin:100px auto;
    display:flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
    animation: zoom .6s;
}
```