link: null
title: 珠峰架构师成长计划
description: 用数组保存所有列表元素的位置，只渲染可视区内的列表元素，当可视区滚动时，根据滚动的offset大小以及所有列表元素的位置，计算在可视区应该渲染哪些元素
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=40 sentences=119, words=974

1. 使用React.Fragment

- 使用 `React.Fragment`来避免向 DOM 添加额外的节点

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
class Users extends React.Component {
  render() {
    return (
      <React.Fragment>
        <div>用户1div>
        <div>用户2div>
      React.Fragment>
    );
  }
}
ReactDOM.render(<Users />, document.querySelector('#root'));
```

2. 使用 React.Lazy 延迟加载组件

- `React.Lazy`帮助我们按需加载组件，从而减少我们应用程序的加载时间，因为只加载我们所需的组件。
- `React.lazy` 接受一个函数，这个函数内部调用 import() 动态导入。它必须返回一个 Promise，该 Promise 需要 resolve 一个 defalut export 的 React 组件。
- React.Suspense 用于包装延迟组件以在加载组件时显示后备内容。

```
import React, { Component,lazy, Suspense } from 'react'
import ReactDOM from 'react-dom';
import Loading from './components/Loading';
const AppTitle  = lazy(()=>import('./components/Title'))

class App extends Component{
    state = {visible:false}
    show = ()=>{
        this.setState({visible:true});
    }
    render() {
        return (
            <>
             {this.state.visible&&(
                 <Suspense fallback={<Loading/>}>
                    <AppTitle/>
                 Suspense>
             )}
             <button onClick={this.show}>加载button>
            </>
        )
    }
}
ReactDOM.render(<App />, document.querySelector('#root'));
```

3. 错误边界(Error Boundaries)

- 如果当一个组件异步加载下载js文件时，网络错误，无法下载 js 文件
- Suspense 无法处理这种错误情况， 在 react 中有一个 错误边界 （Error Boundaries）的概念，用来解决这种问题，它是利用了 react 生命周期的 componetDidCatch 方法来处理
- 有两种方式，一种是 生命周期 componentDidCatch 来处理错误，还有一种 是 静态方法 static getDerivedStateFromError 来处理错误，
- 请使用 `static getDerivedStateFromError()`渲染备用 UI ，使用 componentDidCatch() 打印错误信息。

```
import React, { Component,lazy, Suspense } from 'react'
import ReactDOM from 'react-dom';
import Loading from './components/Loading';
const AppTitle  = lazy(()=>import('./components/Title'))

class App extends Component{
    state = {visible:false,isError: false}
    show = ()=>{
        this.setState({visible:true});
    }

    static getDerivedStateFromError(error) {
      return { isError: true };
    }

    componentDidCatch (err, info) {
      console.log(err, info)
    }
    render() {
        if (this.state.isError) {
            return (<div>errordiv>)
        }
        return (
            <>
             {this.state.visible&&(
                 <Suspense fallback={<Loading/>}>
                    <AppTitle/>
                 Suspense>
             )}
             <button onClick={this.show}>加载button>
            </>
        )
    }
}
ReactDOM.render(<App />, document.querySelector('#root'));
```

4. PureComponent

- 当一个组件的 props或 state变更，React 会将最新返回的元素与之前渲染的元素进行对比，以此决定是否有必要更新真实的 DOM，当它们不相同时 React 会更新该 DOM。
- 如果渲染的组件非常多时可以通过覆盖生命周期方法 shouldComponentUpdate 来进行优化
- shouldComponentUpdate 方法会在重新渲染前被触发。其默认实现是返回 true,如果组件不需要更新，可以在 shouldComponentUpdate中返回 false 来跳过整个渲染过程。其包括该组件的 render 调用以及之后的操作
- PureComponent通过prop和state的浅比较来实现 shouldComponentUpdate

3.1 App.js

```
import React from 'react';
import {Button,message} from 'antd';
import PureComponent from './PureComponent';
export default class App extends PureComponent{
  state = {
    title:'计数器',
    number:0
  }
  add = ()=>{
    this.setState({number:this.state.number+parseInt(this.amount.value)});
  }
  render(){
    console.log('App render');
    return (
      <div>
        <Title2 title={this.state.title}/>
        <Counter number={this.state.number}/>
        <input ref={inst=>this.amount = inst}/>
        <button onClick={this.add}>+button>
      div>
    )
  }
}
class Counter extends PureComponent{
  render(){
    console.log('Counter render');
    return (
     <p>{this.props.number}p>
    )
  }
}

class Title extends PureComponent{
  render(){
    console.log('Title render');
    return (
     <p>{this.props.title}p>
    )
  }
}

const Title2 = React.memo(props=>{
  console.log('Title2 render');
  return  <p>{props.title}p>;
});

function memo(func){
  class Proxy extends PureComponent{
    render(){
      return func(this.props);
    }
  }
  return Proxy;
}

function memo2(Func){
  class Proxy extends PureComponent{
    render(){
      return <Func {...this.props}/>
    }
  }
  return Proxy;
}
```

3.2 PureComponent

```
import React from 'react';
function shallowEqual(obj1,obj2){
    if(obj1 === obj2){
        return true;
    }
    if(typeof obj1 != 'object' || obj1 === null ||typeof obj2 != 'object' || obj2 === null ){
        return false;
    }
    let keys1 = Object.keys(obj1);
    let keys2 = Object.keys(obj2);
    if(keys1.length != keys2.length){
        return false;
    }
    for(let key of keys1){
        if(!obj2.hasOwnProperty(key) || obj1[key] !== obj2[key]){
            return false;
        }
    }
    return true;
}
export default class PureComponent extends React.Component{
    isPureReactComponent = true
    shouldComponentUpdate(nextProps,nextState){
        return !shallowEqual(this.props,nextProps)||!shallowEqual(this.state,nextState)
    }
}
```

5. 长列表优化

- 用数组保存所有列表元素的位置，只渲染可视区内的列表元素，当可视区滚动时，根据滚动的offset大小以及所有列表元素的位置，计算在可视区应该渲染哪些元素
- react-window (https://www.npmjs.com/package/react-window)
- fixed-size (https://react-window.now.sh/#/examples/list/fixed-size)
- react-virtualized (https://www.npmjs.com/package/react-virtualized)

```
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>长列表优化title>
    <style>
    *{
        margin: 0;
        padding: 0;
    }
    ul,li{
        list-style: none;
    }
    style>
  head>
  <body>
    <div id="container" style="height:150px;overflow:auto">
      <ul id="list">ul>
      <div id="content-placeholder">div>
    div>
    <script>
      const ITEM_HEIGHT = 30;
      const ITEM_COUNT = 10;

      window.onload = function() {
        const container = document.querySelector("#container");
        const containerHeight = container.clientHeight;
        const list = document.querySelector("#list");
        list.style.height = containerHeight+'px';
        const visibleCount = Math.ceil(containerHeight / ITEM_HEIGHT);
        const placeholder = document.querySelector("#content-placeholder");
        list.appendChild(renderNodes(0, visibleCount));
        placeholder.style.height = (ITEM_COUNT * ITEM_HEIGHT -containerHeight) + "px";
        container.addEventListener("scroll", function() {
          list.style.webkitTransform = 'translateY('+container.scrollTop+'px)';
          list.innerHTML = "";
          const firstIndex = Math.floor(container.scrollTop / ITEM_HEIGHT);
          list.appendChild(renderNodes(firstIndex, firstIndex + visibleCount));
        });
      };

      function renderNodes(from, to) {
        const fragment = document.createDocumentFragment();
        for (let i = from; i < to; i++) {
          const el = document.createElement("li");
          el.style.height='30px';
          el.innerHTML = i + 1;
          fragment.appendChild(el);
        }
        return fragment;
      }
    script>
  body>
html>
```

```
import React, { Component, lazy, Suspense } from "react";
import ReactDOM from "react-dom";
import { FixedSizeList as List } from 'react-window';

const Row = ({ index, style }) => (
  <div style={style}>Row {index}div>
);

const Container = () => (
  <List
    height={150}
    itemCount={1000}
    itemSize={35}
    width={300}
  >
    {Row}
  List>
);
ReactDOM.render(<Container/>, document.querySelector("#root"));
```

```
import React, { Component, lazy, Suspense } from "react";
import ReactDOM from "react-dom";

class List extends React.Component{
    state = {start:1}
    constructor(){
        super();
        this.containerRef = React.createRef();
    }
    componentDidMount(){
        this.containerRef.current.addEventListener('scroll',()=>{
            let scrollTop = this.containerRef.current.scrollTop;
            let start = Math.floor(scrollTop/this.props.itemSize);
            this.setState({start});
        });
    }
    render(){
        let {width,height,itemCount,itemSize} = this.props;
        let containerStyle = {height,width,position:'relative',border:'1px solid red',overflow:'auto'};
        let itemStyle = {height:itemSize,width:'100%',position:'absolute',left:0,top:0};
        let render = this.props.children;
        let children = [];
        let size = Math.floor(height/itemSize)+1;
        for(let index=this.state.start;indexthis.state.start+size;index++){
            let style = {...itemStyle,top:(index-1)*itemSize};
            children.push(render({index,style}));
        }
        let topStyle = {width:'100%',height:itemSize*this.start};
        return (
            <div style={containerStyle} ref={this.containerRef}>
                <div style={{width:'100%',height:itemSize*itemCount}}>
                    {children}
                div>
            div>
        )
    }
}
const Row = ({ index, style }) => (
  <div key={index} style={style}>Row{index}div>
);

const Container = () => (
  <List
    height={150}
    itemCount={100}
    itemSize={30}
    width={300}
  >
    {Row}
  List>
);
ReactDOM.render(<Container/>, document.querySelector("#root"));
```

6. react devtool

- react-devtools (https://github.com/facebook/react-devtools)
- profiler (http://react.html.cn/blog/2018/09/10/introducing-the-react-profiler.html)
- react-flame-graph (https://react-flame-graph.now.sh/)