---

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=128 sentences=227, words=1868

---

1. React18介绍

- React 18发布计划 (https://zh-hans.reactjs.org/blog/2021/06/08/the-plan-for-react-18.html)
- React 18 工作组 (https://github.com/reactwg/react-18/discussions)
- startTransition (https://github.com/reactwg/react-18/discussions/41)
- React.lazy的全新SSR 架构 (https://github.com/reactwg/react-18/discussions/37)

2. 并发模式(concurrent mode)

- 在React 18中新加入的可选的并发渲染(concurrent rendering) (https://zh-hans.reactjs.org/docs/concurrent-mode-intro.html)机制
- Concurrent 模式是一组 React 的新功能，可帮助应用保持响应，并根据用户的设备性能和网速进行适当的调整
- 在 Concurrent 模式中，渲染不是阻塞的。它是可中断的

2.1 更新优先级

- 以前更新没有优先级的概念,优先级高的更新并不能打断之前的更新,需要等前面的更新完成后才能进行
- 用户对不同的操作对交互的执行速度有不同的预期,所以我们可以根据用户的预期赋予更新不同的优先级
    - 高优先级 用户输入、窗口缩放和拖拽事件等
    - 低优先级 数据请求和下载文件等
- 高优先级的更新会中断正在进行的低优先级的更新
- 等高优先级更新完成后,低优先级基于高优先级更新的结果重新更新
- 对于 CPU-bound 的更新 (例如创建新的 DOM 节点和运行组件中的代码)，并发意味着一个更急迫的更新可以"中断"已经开始的渲染

2.2 双缓冲

- 当数据量很大时，绘图可能需要几秒钟甚至更长的时间，而且有时还会出现闪烁现象，为了解决这些问题，可采用双缓冲技术来绘图
- 双缓冲 (https://wiki.osdev.org/Double_Buffering)即在内存中创建一个与屏幕绘图区域一致的对象，先将图形绘制到内存中的这个对象上，再一次性将这个对象上的图形拷贝到屏幕上，这样能大大加快绘图的速度
- 对于 IO-bound的更新 (例如从网络加载代码或数据)，并发意味着 React 甚至可以在全部数据到达之前就在内存中开始渲染，然后跳过令人不愉快的空白加载状态

3.搭建开发环境

3.1 vite

- Vite是Vue的作者开发的Web开发构建工具
- 它是一个基于浏览器原生ES模块导入的开发服务器
- 在开发环境下，利用浏览器去解析import，在服务器端按需编译返回，完全跳过了打包这个概念
- 服务器随启随用
- 同时不仅对Vue文件提供了支持，还支持热更新，而且热更新的速度不会随着模块增多而变慢

3.2 安装

- plugin-react-refresh (plugin-react-refresh)支持react组件的热更新

```
npm install react@alpha react-dom@alpha @types/react @types/react-dom -S
npm install vite typescript @vitejs/plugin-react-refresh -D
node ./node_modules/esbuild/install.js
```

3.3 vite.config.ts

```
import { defineConfig } from 'vite'
import reactRefresh from '@vitejs/plugin-react-refresh'
export default defineConfig({
  plugins: [reactRefresh()]
})
```

3.4 tsconfig.json

tsconfig.json

```
{
  "compilerOptions": {
    "target": "ESNext",
    "lib": ["DOM", "DOM.Iterable", "ESNext"],
    "allowJs": false,
    "skipLibCheck": false,
    "esModuleInterop": false,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "module": "ESNext",
    "moduleResolution": "Node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react",
    "types": ["react/next", "react-dom/next"]
  },
  "include": ["./src"]
}
```

3.5 package.json

package.json

```
{
  "name": "zhufeng-react18",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react": "^18.0.0-alpha-ed6c091fe-20210701",
    "react-dom": "^18.0.0-alpha-ed6c091fe-20210701"
  },
  "devDependencies": {
    "@types/react": "^17.0.13",
    "@types/react-dom": "^17.0.8",
    "@vitejs/plugin-react-refresh": "^1.3.5",
    "typescript": "^4.3.5",
    "vite": "^2.4.1"
  }
}
```

3.6 index.html

- `type='module'`可以导入ES6模块 (https://www.sitepoint.com/using-es-modules/),可以启用ESM模块机制

index.html

```
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/src/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite Apptitle>
  head>
  <body>
    <div id="root">div>
    <script type="module" src="/src/main.tsx">script>
  body>
html>
```

3.7 src\main.tsx

- `legacy`模式 `ReactDOM.renders`会同步渲染
- `createRoot`会启用 `concurrent`并发模式

src\main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom'
ReactDOM.createRoot(
    document.getElementById('root')!

).render(<h1>hellohl>);
```

3.8 启动

```
npm run dev
```

4.批量更新

- automatic batching (https://github.com/reactwg/react-18/discussions/21)
- 在 `Concurrent`模式中更新是以优先级为依据进行合并的

4.1 安装路由

- npm强制安装可以使用 `-f` 或 `--force` 参数

```
npm install react-router-dom @types/react-router-dom --force -S
```

4.2 src\main.tsx

src\main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom'
+import {HashRouter as Router,Route,Link} from 'react-router-dom';
+import BatchState from './routes/BatchState';
ReactDOM.createRoot(
    document.getElementById('root')!

).render(
+
+
+           BatchState
+
+
+
);
```

4.3 BatchState.tsx

src\routes\BatchState.tsx

```tsx
import React, { Component } from 'react'
interface Props { }
interface State {
    count: number
}
export default class extends Component<Props, State> {
    state = { count: 0 }
    handleCLick = () => {
        setTimeout(() => {
            this.setState({ count: this.state.count + 1 });
            console.log("count", this.state.count);
            this.setState({ count: this.state.count + 1 });
            console.log("count", this.state.count);
        }, 0);
    };

    render() {
        return (
            <div>
                <p>{this.state.count}p>
                <button onClick={this.handleCLick}>+button>
            div>
        );
    }
}
```

5.Suspense

- Suspense 让你的组件在渲染之前进行 等待，并在等待时显示 fallback的内容
- Suspense内的组件子树比组件树的其他部分拥有更低的优先级
- 执行流程
    - 在render函数中我们可以使用异步请求数据
    - react会从我们缓存中读取这个缓存
    - 如果有缓存了，直接进行正常的render
    - 如果没有缓存，那么会抛出一个promise异常
    - 当这个promise完成后(比发请求数据完成)，react会继续回到原来的render中，把数据render出来
    - 完全同步写法，没有任何异步 callback之类的东西
- React提供了一个内置函数 componentDidCatch,如果 render() 函数抛出错误，则会触发该函数
- ErrorBoundary(错误边界)是一个组件，该组件会捕获到渲染期间(render)子组件发生的错误，并有能力阻止错误继续传播

5.1 src\main.tsx

src\main.tsx

```tsx
import React from 'react'
import ReactDOM from 'react-dom'
import {HashRouter as Router,Route,Link} from 'react-router-dom';
import BatchState from './routes/BatchState';
+import Suspense from './routes/Suspense';
ReactDOM.createRoot(
    document.getElementById('root')!

).render(

            BatchState
+           Suspense

+


);
```

5.2 ErrorBoundary.tsx

src\components\ErrorBoundary.tsx

```tsx
import React from "react";
interface Props{
    fallback:React.ReactNode
}
export default class ErrorBoundary extends React.Component<Props> {
  state = {hasError: false, error: null};
  static getDerivedStateFromError(error:any) {
    return {
      hasError: true,
      error,
    };
  }
  render() {
    if (this.state.hasError) {
      return this.props.fallback;
    }
    return this.props.children;
  }
}
```

5.3 Suspense.tsx

src\routes\Suspense.tsx

```
import React, { Component, Suspense } from 'react'
import ErrorBoundary from "../components/ErrorBoundary";
function createResource(promise: Promise) {
    let status = 'pending';
    let result: any;
    return {
        read() {
            if (status === 'success' || status === 'error') {
                return result;
            } else {
                throw promise.then((data: any) => {
                    status = 'success';
                    result = data;
                }, (error: any) => {
                    status = 'error';
                    result = error;
                });
            }
        }
    }
}

function fetchData(id: number) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve({ success: true, data: { id, name: '张三' } });

        }, 1000);
    });
}
const initialResource = createResource(fetchData(1));
function User() {
    const result = initialResource.read();
    if (result.success) {
        let user = result.data;
        return <p>{user.id}:{user.name}p>;
    } else {
        return <p>{result.message}p>;
    }
}

export default class extends Component {
    render() {
        return (
            <ErrorBoundary fallback={<h1>出错了h1>}>
                <Suspense fallback={<h1>加载中h1>}>
                    <User />
                Suspense>
            ErrorBoundary>
        );
    }
}
```

5.4 Suspense.tsx

src\components\Suspense.tsx

```
import React, { Component } from 'react'
interface SuspenseProps {
    fallback: React.ReactNode
}
interface SuspenseState {
    loading: boolean
}
export default class Suspense extends React.Component<SuspenseProps, SuspenseState> {
    mounted: any = null
    state = { loading: false };
    componentDidCatch(error: any) {
        if (typeof error.then === 'function') {
            this.setState({ loading: true });
            error.then(() => { this.setState({ loading: false }) });
        }
    }
    render() {
        const { fallback, children } = this.props;
        const { loading } = this.state;
        return loading ? fallback : children;
    }
}
```

6.SuspenseList

- SuspenseList 通过编排向用户显示这些组件的顺序，来帮助协调许多可以挂起的组件
- revealOrder (forwards, backwards, together) 定义了 SuspenseList 子组件应该显示的顺序
    - together 在所有的子组件都准备好了的时候显示它们，而不是一个接着一个显示
    - forwards 从前往后显示
    - backwards 从后往前显示
- tail (collapsed, hidden) 指定如何显示 SuspenseList 中未加载的项目
    - 默认情况下，SuspenseList 将显示列表中的所有 fallback
    - collapsed 仅显示列表中下一个 fallback
    - hidden 未加载的项目不显示任何信息

6.1 src\main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom'
import {HashRouter as Router,Route,Link} from 'react-router-dom';
import BatchState from './routes/BatchState';
import Suspense from './routes/Suspense';
+import SuspenseList from './routes/SuspenseList';
ReactDOM.createRoot(
    document.getElementById('root')!

).render(

            BatchState
            Suspense
+           SuspenseList

+

);
```

6.2 SuspenseList.tsx

src\routes\SuspenseList.tsx

```
import React, { Component, Suspense, SuspenseList } from 'react'
import ErrorBoundary from "../components/ErrorBoundary";
function createResource(promise: Promise) {
    let status = 'pending';
    let result: any;
    return {
        read() {
            if (status === 'success' || status === 'error') {
                return result;
            } else {
                throw promise.then((data: any) => {
                    status = 'success';
                    result = data;
                }, (error: any) => {
                    status = 'error';
                    result = error;
                });
            }
        }
    }
}

function fetchData(id: number) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve({ success: true, data: { id, name: '姓名' + id } });

        }, 1000 * id);
    });
}

let userResourceMap: any = {
    1: createResource(fetchData(1)),
    2: createResource(fetchData(2)),
    3: createResource(fetchData(3))
}
interface UserProps {
    id: number
}
function User(props: UserProps) {
    const result = userResourceMap[props.id].read();
    if (result.success) {
        let user = result.data;
        return <p>{user.id}:{user.name}p>;
    } else {
        return <p>{result.message}p>;
    }
}

export default class extends Component {
    render() {
        return (
            <ErrorBoundary fallback={<h1>出错了h1>}>
                <SuspenseList revealOrder="backwards" tail="collapsed" >
                    <Suspense fallback={<h1>加载用户3......h1>}>
                        <User id={3} />
                    Suspense>
                    <Suspense fallback={<h1>加载用户2......h1>}>
                        <User id={2} />
                    Suspense>
                    <Suspense fallback={<h1>加载用户1......h1>}>
                        <User id={1} />
                    Suspense>
                SuspenseList>

            ErrorBoundary>
        );
    }
}
```

7.startTransition

- startTransition (https://zh-hans.reactjs.org/docs/concurrent-mode-reference.html)
- startTransition 是一个接受回调的函数。我们用它来告诉 React 需要推迟的 state
- 允许组件将速度较慢的数据获取更新推迟到随后渲染，以便能够立即渲染更重要的更新

7.1 src\main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom'
import {HashRouter as Router,Route,Link} from 'react-router-dom';
import BatchState from './routes/BatchState';
import Suspense from './routes/Suspense';
import SuspenseList from './routes/SuspenseList';
+import StartTransition from './routes/StartTransition';
ReactDOM.createRoot(
    document.getElementById('root')!

).render(

            BatchState
            Suspense
            SuspenseList
+           StartTransition

+


);
```

7.2 StartTransition.tsx

src\routes\StartTransition.tsx

```
import React, { startTransition, useEffect, useState } from 'react';
function getSuggestions(keyword: string):Promise<Array<string>> {
  let items =  new Array(10000).fill(0).map((item: number, index: number) => keyword + index);
  return Promise.resolve(items);
}
interface SuggestionProps {
  keyword: string;
}
function Suggestion(props: SuggestionProps) {
  const [suggestions, setSuggestions] = useState<Array>([]);
  useEffect(() => {
    getSuggestions(props.keyword).then(suggestions => {
      startTransition(() => {
        setSuggestions(suggestions);
      })
    })
  }, [props.keyword]);

  return (
    <ul>
      {
        suggestions.map((item: string) => (<li key={item}>{item}li>))
      }
    ul>
  )
}

export default function () {
  const [keyword, setKeyword] = useState("");
  const handleChange = (event: React.ChangeEvent) => {
    setKeyword(event.target.value);
  };
  return (
    <div>
      请输入商品关键字<input value={keyword} onChange={handleChange} />
      <Suggestion keyword={keyword} />
    div>
  );
}
```

8.useDeferredValue

- 返回一个延迟响应的值
- 在 useDeferredValue内部会调用useState并触发一次更新,但此更新的优先级很低

8.1 src\main.tsx

src\main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom'
import {HashRouter as Router,Route,Link} from 'react-router-dom';
import BatchState from './routes/BatchState';
import Suspense from './routes/Suspense';
import SuspenseList from './routes/SuspenseList';
import StartTransition from './routes/StartTransition';
+import UseDeferredValue from './routes/UseDeferredValue';
ReactDOM.createRoot(
    document.getElementById('root')!

).render(

            BatchState
            Suspense
            SuspenseList
            StartTransition
+           UseDeferredValue

+


);
```

8.2 UseDeferredValue.tsx

src\routes\UseDeferredValue.tsx

```
import React, { startTransition, useEffect, useState,useDeferredValue } from 'react';
function getSuggestions(keyword: string):Promise<Array<string>> {
  let items =  new Array(10000).fill(0).map((item: number, index: number) => keyword + index);
  return Promise.resolve(items);
}
interface SuggestionProps {
  keyword: string;
}
function Suggestion(props: SuggestionProps) {
  const [suggestions, setSuggestions] = useState<Array>([]);
  useEffect(() => {
    getSuggestions(props.keyword).then(suggestions => {
       setSuggestions(suggestions);
    })
  }, [props.keyword]);

  return (
    <ul>
      {
        suggestions.map((item: string) => (<li key={item}>{item}li>))
      }
    ul>
  )
}

export default function () {
  const [keyword, setKeyword] = useState("");
  const deferredText = useDeferredValue(keyword);
  const handleChange = (event: React.ChangeEvent) => {
    setKeyword(event.target.value);
  };
  return (
    <div>
      请输入商品关键字<input value={keyword} onChange={handleChange} />
      <Suggestion keyword={deferredText} />
    div>
  );
}
```

9.useTransition

- useTransition允许组件在切换到下一个界面之前等待内容加载，从而避免不必要的加载状态
- 它还允许组件将速度较慢的数据获取更新推迟到随后渲染，以便能够立即渲染更重要的更新
- useTransition hook 返回两个值的数组
    - startTransition 是一个接受回调的函数。我们用它来告诉 React 需要推迟的 state
    - isPending 是一个布尔值。这是 React 通知我们是否正在等待过渡的完成的方式
- 如果某个 state 更新导致组件挂起，则该 state 更新应包装在 transition 中

9.1 src\main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom'
import {HashRouter as Router,Route,Link} from 'react-router-dom';
import BatchState from './routes/BatchState';
import Suspense from './routes/Suspense';
import SuspenseList from './routes/SuspenseList';
import StartTransition from './routes/StartTransition';
import UseDeferredValue from './routes/UseDeferredValue';
+import UseTransition from './routes/UseTransition';
ReactDOM.createRoot(
    document.getElementById('root')!

).render(

            BatchState
            Suspense
            SuspenseList
            StartTransition
            UseDeferredValue
+           UseTransition

+

);
```

9.2 UseTransition.tsx

src\routes\UseTransition.tsx

```
import React, { Component, Suspense, useTransition, useState } from 'react'
import ErrorBoundary from "../components/ErrorBoundary";
function fetchData(id: number) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve({ success: true, data: { id, name: '张三' + id } });

        }, 3000);
    });
}

interface UserProps {
    resource: any
}
function User(props: UserProps) {
    const result = props.resource.read();
    if (result.success) {
        let user = result.data;
        return <p>{user.id}:{user.name}p>;
    } else {
        return <p>{result.message}p>;
    }
}

function createResource(promise: Promise) {
    let status = 'pending';
    let result: any;
    return {
        read() {
            if (status === 'success' || status === 'error') {
                return result;
            } else {
                throw promise.then((data: any) => {
                    status = 'success';
                    result = data;
                }, (error: any) => {
                    status = 'error';
                    result = error;
                });
            }
        }
    }
}
const initialResource = createResource(fetchData(1));
export default function () {
    const [resource, setResource] = useState(initialResource);
    const [isPending, startTransition] = useTransition();
    return (
        <>
            <ErrorBoundary fallback={<h1>出错了h1>}>
                <Suspense fallback={<h1>加载中...h1>}>
                    <User resource={resource} />
                Suspense>
            ErrorBoundary>
            {isPending ? " 加载中..." : null}
            <button
                disabled={isPending}
                onClick={() => {
                    //startTransition(() => {
                        setResource(createResource(fetchData(2)));
                    //});
                }}
            >下一个用户button>
        </>
    )
}
```