link: null
title: 珠峰架构师成长计划
description: vite.config.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=116 sentences=261, words=1873

1.React18

- react-18 (https://github.com/reactwg/react-18/discussions)
- New Suspense SSR Architecture in React 18 (https://github.com/reactwg/react-18/discussions/37)
- Upgrading to React 18 on the server (https://github.com/reactwg/react-18/discussions/22)
- hooks-reference (https://zh-hans.reactjs.org/docs/hooks-reference.html)

1.1 大纲

- 批量更新
- Suspense
- startTransition
- useDeferredValue
- useTransition
- Suspense SSR
- 并发渲染
- 更新优先级
- 双缓冲
- 水合
- React18升级指南

2.创建项目

- vitejs (https://cn.vitejs.dev/)
- plugin-react-refresh (https://www.npmjs.com/package/@vitejs/plugin-react-refresh)

2.1 安装依赖

```
npm install react  react-dom  @types/react @types/react-dom --save
npm install vite  @vitejs/plugin-react-refresh --save-dev
```

2.2 vite.config.js

vite.config.js

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

2.2 package.json

package.json

```
{
  "scripts": {
    "start": "vite"
  },
}
```

2.3 index.html

index.html

```

<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Vite Apptitle>
head>

<body>
  <div id="root">div>
  <script src="/src/main.jsx" type="module">script>
body>

html>
```

2.4 main.jsx

src\main.jsx

```
import React from 'react'
import { render } from 'react-dom'
import { createRoot } from 'react-dom/client'
const root = document.getElementById('root');
const element = <div>appdiv>
render(element, root);
createRoot(root).render(element);
```

3. 批量更新

- automatic batching (https://github.com/reactwg/react-18/discussions/21)
- 在React中多次的setState合并到一次进行渲染
- 在React18中更新是以优先级为依据进行合并的

3.1 老的批量更新

3.1.1 main.jsx

src\main.jsx

```
import React from 'react'
import { render } from 'react-dom'
import { createRoot } from 'react-dom/client'
const root = document.getElementById('root');
+import OldBatchUpdatePage from './routes/OldBatchUpdatePage';
+const element =
render(element, root);
//createRoot(root).render(element);
```

3.1.2 OldBatchUpdatePage.jsx

src\routes\OldBatchUpdatePage.jsx

```
import React, { Component } from 'react'
import { unstable_batchedUpdates } from 'react-dom'
class OldBatchUpdatePage extends Component {
  state = { number: 0 }
  handleCLick = () => {
    this.setState({ number: this.state.number + 1 });
    console.log("number", this.state.number);
    this.setState({ number: this.state.number + 1 });
    console.log("number", this.state.number);
    setTimeout(() => {

      this.setState({ number: this.state.number + 1 });
      console.log("number", this.state.number);
      this.setState({ number: this.state.number + 1 });
      console.log("number", this.state.number);

    }, 0);
  };

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

3.1.3 1.批量更新老实现.js

```
let isBatchingUpdate = false;
let updateQueue = [];
let state = { number: 0 };
function setState(newState) {
  if (isBatchingUpdate) {
    updateQueue.push(newState);
  } else {
    state = newState;
  }
}
const handleCLick = () => {
  setState({ number: state.number + 1 });
  console.log("number", state.number);
  setState({ number: state.number + 1 });
  console.log("number", state.number);
  setTimeout(() => {
    setState({ number: state.number + 1 });
    console.log("number", state.number);
    setState({ number: state.number + 1 });
    console.log("number", state.number);
  }, 0);
};
function batchedUpdates(fn) {
  isBatchingUpdate = true;
  fn();
  isBatchingUpdate = false;
  updateQueue.forEach(newState => {
    state = newState
  });
}
batchedUpdates(handleCLick);
```

3.2 新的批量更新

3.2.1 main.jsx

src\main.jsx

```
import React from 'react'
import { render } from 'react-dom'
import { createRoot } from 'react-dom/client'
const root = document.getElementById('root');
import OldBatchUpdatePage from './routes/OldBatchUpdatePage';
+import NewBatchUpdatePage from './routes/NewBatchUpdatePage';
+const element =
render(element, root);
//createRoot(root).render(element);
```

3.2.2 NewBatchUpdatePage.jsx

src\routes\NewBatchUpdatePage.jsx

```
import React, { Component } from 'react'
import { flushSync } from 'react-dom'
class NewBatchUpdatePage extends Component {
  state = { number: 0 }
  handleCLick = () => {
    this.setState({ number: this.state.number + 1 });
    console.log("number", this.state.number);
    this.setState({ number: this.state.number + 1 });
    console.log("number", this.state.number);
    setTimeout(() => {
      flushSync(() => {
        this.setState({ number: this.state.number + 1 });
      });
      console.log("number", this.state.number);
      flushSync(() => {
        this.setState({ number: this.state.number + 1 });
      });
      console.log("number", this.state.number);
    }, 0);
  };

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

3.2.3 2.批量更新新实现.js

```
let updateQueue = [];
let lastPriority = -1;
let state = { number: 0 };
const InputPriority = 1;
const NormalPriority = 1;
let lastUpdatePriority;

function setState(newState, priority) {
  updateQueue.push(newState);
  if (lastUpdatePriority === priority) {
    return;
  }
  lastUpdatePriority = priority;
  setTimeout(() => {
    updateQueue.forEach(newState => {
      state = newState
    });
  });
}
function flushSync(fn) {
  lastUpdatePriority = null;
  fn();
}

const handleCLick = () => {
  setState({ number: state.number + 1 }, InputPriority);
  console.log("number", state.number);
  setState({ number: state.number + 1 }, InputPriority);
  console.log("number", state.number);
  setTimeout(() => {
    setState({ number: state.number + 1 }, NormalPriority);
    console.log("number", state.number);
    setState({ number: state.number + 1 }, NormalPriority);
    console.log("number", state.number);
  }, 500);
};
handleCLick();
```

4. 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
- 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
- 官方demo (https://codesandbox.io/s/frosty-hermann-bztrp)

4.1 main.jsx

src\main.jsx

```
import React from 'react'
-import { render } from 'react-dom'
import { createRoot } from 'react-dom/client'
const root = document.getElementById('root');
import OldBatchUpdatePage from './routes/OldBatchUpdatePage';
import NewBatchUpdatePage from './routes/NewBatchUpdatePage';
+import SuspensePage from './routes/SuspensePage';
+const element =
-render(element, root);
createRoot(root).render(element);
```

4.2 SuspensePage.jsx

src\routes\SuspensePage.jsx

```
import React, { Component, Suspense } from 'react'
import ErrorBoundary from './components/ErrorBoundary';
import { fetchUser } from '../fakeApi';
import { wrapPromise } from '../utils';
const userPromise = fetchUser('1');
const userResource = wrapPromise(userPromise);
class SuspensePage extends Component {
  render() {
    return (
      <Suspense fallback={<h3>Loading User......h3>}>
        <ErrorBoundary>
          <User />
        ErrorBoundary>
      Suspense >
    );
  }
}
function User() {
  const user = userResource.read();
  return <div>{user.id}:{user.name}div>
}

export default SuspensePage;
```

4.3 ErrorBoundary.jsx

src\routes\components\ErrorBoundary.jsx

```
import React from 'react'
class ErrorBoundary extends React.Component {
  state = { hasError: false, error: null };
  static getDerivedStateFromError(error) {
    return {
      hasError: true,
      error
    };
  }
  render() {
    if (this.state.hasError) {
      return (
        <>
          {this.state.error.msg}
        </>
      );
    }
    return this.props.children;
  }
}
export default ErrorBoundary
```

4.4 fakeApi.jsx

src\fakeApi.jsx

```
export function fetchUser(id) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve({ id, name: `姓名${id}` });

    }, 1000 * Number(id));
  });
}
```

4.5 utils.jsx

src\utils.jsx

```
export function wrapPromise(promise) {
  let status = "pending";
  let result;
  let suspender = promise.then(
    (r) => {
      status = "success";
      result = r;
    },
    (e) => {
      status = "error";
      result = e;
    }
  );
  return {
    read() {
      if (status === "pending") {
        throw suspender;
      } else if (status === "error") {
        throw result;
      } else if (status === "success") {
        return result;
      }
    }
  };
}
```

5. startTransition

- startTransition主要为了能在大量的任务下也能保持 UI 响应
- startTransition可以通过将特定更新标记为 过渡来显着改善用户交互

5.1 main.jsx

src\main.jsx

```
import React from 'react'
import { render } from 'react-dom'
import { createRoot } from 'react-dom/client'
const root = document.getElementById('root');
import OldBatchUpdatePage from './routes/OldBatchUpdatePage';
import NewBatchUpdatePage from './routes/NewBatchUpdatePage';
import SuspensePage from './routes/SuspensePage';
+import StartTransitionPage from './routes/StartTransitionPage';
+const element =
//render(element, root);
createRoot(root).render(element);
```

5.2 StartTransitionPage.jsx

src\routes\StartTransitionPage.jsx

```
import React, { startTransition, useEffect, useState } from 'react';
function getSuggestions(keyword) {
  let items = new Array(10000).fill(keyword)
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(items);
    }, 1000 * keyword.length);
  });
}

function Suggestion(props) {
  const [suggestions, setSuggestions] = useState([]);
  useEffect(() => {
    if (props.keyword && props.keyword.length > 0) {
      getSuggestions(props.keyword).then(suggestions => {
        startTransition(() => {
          setSuggestions(suggestions);
        })
      })
    }
  }, [props.keyword]);
  useEffect(() => {
    console.log(props.keyword);
  })
  return (
    <ul>
      {
        suggestions.map((item, index) => (<li key={index}>{item}li>))
      }
    ul>
  )
}
function StartTransitionPage() {
  const [keyword, setKeyword] = useState < string > ("");
  const handleChange = (event) => {
    setKeyword(event.target.value);
  };
  return (
    <div>
      <div>关键字<input value={keyword} onChange={handleChange} />div>
      <Suggestion keyword={keyword} />
    div>
  );
}
export default StartTransitionPage
```

5.3 UpdatePriorityPage.jsx

UpdatePriorityPage.js

```
import React, { startTransition, useEffect } from 'react'
import { flushSync } from 'react-dom';
function UpdatePriorityPage() {
  let [state, setState] = React.useState('');
  useEffect(() => {
    console.log(state);
  });
  const update = () => {
    flushSync(() => startTransition(() => setState(state => state + 'A')));
    flushSync(setState(state => state + 'B'));
    flushSync(() => startTransition(() => setState(state => state + 'C')));
    flushSync(setState(state => state + 'D'));
  }
  return (
    <>
      <div>{state}div>
      <button onClick={update}>更新button>
    />
  )
}
export default UpdatePriorityPage;
```

6. useDeferredValue

- 如果说某些渲染比较消耗性能，比如存在实时计算和反馈，我们可以使用这个 `useDeferredValue`降低其计算的优先级，使得避免整个应用变得卡顿

6.1 main.jsx

src\main.jsx

```
import React from 'react'
import { render } from 'react-dom'
import { createRoot } from 'react-dom/client'
const root = document.getElementById('root');
import OldBatchUpdatePage from './routes/OldBatchUpdatePage';
import NewBatchUpdatePage from './routes/NewBatchUpdatePage';
import SuspensePage from './routes/SuspensePage';
import StartTransitionPage from './routes/StartTransitionPage';
+import UseDeferredValuePage from './routes/UseDeferredValuePage';
+const element =
createRoot(root).render(element);
```

6.2 src\routes\UseDeferredValuePage.jsx

src\routes\UseDeferredValuePage.jsx

```
import React, { useDeferredValue, useEffect, useState } from 'react';
function getSuggestions(keyword) {
  let items = new Array(10000).fill(0).map((item, index) => keyword + index);
  return Promise.resolve(items);
}

function Suggestion(props) {
  const [suggestions, setSuggestions] = useState < Array < string >> ([]);
  useEffect(() => {
    getSuggestions(props.keyword).then(suggestions => {
-      startTransition(() => {
        setSuggestions(suggestions);
-      })
    })
  }, [props.keyword]);

  return (

      {
        suggestions.map((item) => ({item}))
      }

  )
}
function StartTransitionPage() {
  const [keyword, setKeyword] = useState < string > ("");
+ const deferredText = useDeferredValue(keyword);
  const handleChange = (event) => {
    setKeyword(event.target.value);
  };
  return (

      关键字
+

  );
}
export default StartTransitionPage
```

7. 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
- useTransition 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

7.1 main.jsx

src\main.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
const root = document.getElementById('root');
import OldBatchUpdatePage from './routes/OldBatchUpdatePage';
import NewBatchUpdatePage from './routes/NewBatchUpdatePage';
import SuspensePage from './routes/SuspensePage';
import StartTransitionPage from './routes/StartTransitionPage';
import UseDeferredValuePage from './routes/UseDeferredValuePage';
+import UseTransitionPage from './routes/UseTransitionPage';
+const element =
createRoot(root).render(element);
```

7.2 UseTransitionPage.jsx

src\routes\UseTransitionPage.jsx

```
import React, { Suspense, useState, useTransition } from 'react'
import ErrorBoundary from './components/ErrorBoundary';
import { fetchUser } from '../fakeApi';
import { wrapPromise } from '../utils';
const user1Resource = wrapPromise(fetchUser('1'));
const user5Resource = wrapPromise(fetchUser('5'));
function UseTransitionPage() {
  const [resource, setResource] = useState(user1Resource);
  const [isPending, startTransition] = useTransition();
  return (
    <>
      <Suspense fallback={<h3>Loading User......h3>}>
        <ErrorBoundary>
          <User resource={resource} />
        ErrorBoundary>
      Suspense>
      <button onClick={() => {
        startTransition(() => {
          setResource(user5Resource)
        });
      }}>user5button>
      <h3>{isPending && <p>isPending...p>}h3>
    </>
  );
}
function User({ resource }) {
  const user = resource.read();
  return <div>{user.id}:{user.name}div>
}
export default UseTransitionPage;
```

9.基础知识

9.1. 并发更新

- What happened to concurrent "mode"? (https://github.com/reactwg/react-18/discussions/64)
- 并发更新就是一种可中断渲染的架构
- 什么时候中断渲染呢？当一个更高优先级渲染到来时，通过放弃当前的渲染，立即执行更高优先级的渲染，换来视觉上更快的响应速度
- 在React18中以是否使用并发特性作为是否开启并发更新的依据

9.2 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

9.3 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

9.4 水合

- 水合反应（ hydrated reaction），也叫作水化
- 是指物质溶解在水里时，与水发生的化学作用,水合分子的过程
- 组件在服务器端拉取数据(水)，并在服务器端首次渲染
- 脱水: 对组件进行脱水，变成HTML字符串，脱去动态数据，成为风干标本快照
- 注水：发送到客户端后，重新注入数据(水)，重新变成可交互组件

10. React18升级指南

- 18.0.0 (https://github.com/facebook/react/releases/tag/v18.0.0)
- How to Upgrade to React 18 (https://reactjs.org/blog/2022/03/08/react-18-upgrade-guide.html)