link: null
title: 珠峰架构师成长计划
description: render\client.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=115 sentences=160, words=1274

1.渲染模式

1.1 服务器渲染

- 页面上的内容是由服务器生产的

```
npm install express --save
```

render\client.js

```
let express = require('express');
let app = express();
app.get('/', (req, res) => {
  res.send(`

            hello

    `);
});
app.listen(8080);
```

1.2 客户端渲染

- 页面上的内容由于浏览器运行JS脚本而渲染到页面上的
  - 浏览器访问服务器
  - 服务器返回一个空的HTML页面，里面有一个JS资源链接，比如 `client`
  - 浏览器下载JS代码并在浏览器中运行
  - 内容呈现在页面上

```
let express = require('express');
let app = express();
app.get('/', (req, res) => {
  res.send(`

            root.innerHTML = 'hello'

    `);
});
app.listen(8090);
```

2. 什么是同构

- 客户端渲染缺点
  - 首屏速度加载慢
  - 不支持SEO和搜索引擎优化
  - 首页需要通过请求初始化数据
- 同构渲染
  - 同构的项目支持客户端渲染和服务器端渲染
  - 第一次访问页面是SSR，后面的访问是SPA，而且支持SEO
  - 客户端和服务器端同构可以实现(尽可能复用代码)

3. 客户端渲染(CSR)

3.1 安装

```
npm install react react-dom --save
npm install webpack webpack-cli babel-loader  @babel/preset-react --save-dev
npm install express cross-env nodemon @babel/register @babel/plugin-transform-modules-commonjs --save-dev
```

3.2 src\index.js

src\index.js

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

3.3 App.js

src\App.js

```
import React from 'react';
import Header from './Header';
import User from './User';
import Footer from './Footer';
function App() {
  return (
    <>
      <Header />
      <User />
      <Footer />
    </>
  );
}
export default App
```

src\Header.js

```
import React from 'react';
function Header() {
  return (
    <div onClick={() => alert('Header')}>Headerdiv>
  );
}
export default Header;
```

3.5 User.js

src\User.js

```
import React from 'react';
function User() {
  return (
    <div>Userdiv>
  );
}
export default User;
```

src\Footer.js

```
import React from 'react';
function Footer() {
  return (
    <div>Footerdiv>
  );
}
export default Footer;
```

3.7 webpack.config.js

webpack.config.js

```
const path = require('path');
module.exports = {
  mode: 'development',
  devtool: false,
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, './build'),
    filename: 'main.js'
  },
  watch: true,
  module: {
    rules: [
      {
        test: /\.js$/,
        enforce: 'pre',
        use: {
          loader: 'source-map-loader',
        }
      },
      {
        test: /\.js$/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ["@babel/preset-react"]
          }
        },
        exclude: /node_modules/
      },
    ],
  },
}
```

3.8 package.json

package.json

```
{
  "scripts": {
    "build": "webpack"
  }
}
```

3.9 index.html

build\index.html

```

<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Documenttitle>
head>
<body>
  <div id="root">div>
  <script src="main.js">script>
body>
html>
```

4. 水合

- 水合原本是指物质溶解在水里时与水发生的化学作用
- 此处水合指的是后端内容打到前端后，JS需要将事件绑定，才能够响应用户的交互或者DOM的更新行为
- 工作流
    - 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

这个是晒干之后的银耳
呈自然的淡黄色

4.1 index.html

build\index.html

```
  Document

+ HeaderUserFooter
```

4.2 src\index.js

src\index.js

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

5. React18之前的服务器端渲染(SSR)

- 工作流
  - 服务器内部获取数据
  - 服务器内部渲染 HTML
  - 客户端从远程加载代码
  - 客户端开始水合
- 一切都是串行的，前一个任务没完成之前，后一任务都无法开始
- 这个操作必须是整体性的，而水合的过程可能比较慢，会引起卡顿
- react-dom-server (https://zh-hans.reactjs.org/docs/react-dom-server.html)
- @babel/register (https://babeljs.io/docs/en/babel-register)在底层改写了 node 的 require 办法，在代码里引入 @babel/register模块后，所有通过require引入并且以 .es6、 .es、 .jsx、 .mjs和 .js为后缀名的模块都会被 Babel 转译
- @babel/plugin-transform-modules-commonjs (https://babeljs.io/docs/en/babel-plugin-transform-modules-commonjs)转换ES模块为CommonJS

5.1 package.json

```
{
  "scripts": {
    "build": "webpack",
+   "start": "cross-env NODE_ENV=development nodemon -- server.js"
  }
}
```

5.2 server.js

server.js

```
const babelRegister = require('@babel/register');
babelRegister({
  ignore: [/node_modules/],
  presets: ["@babel/preset-react"],
  plugins: ['@babel/plugin-transform-modules-commonjs'],
});
const webpack = require('webpack');
const express = require('express');
const static = require('serve-static');
const webpackConfig = require('./webpack.config');
const render = require('./render');
webpack(webpackConfig, (err, stats) => {
  let statsJSON = stats.toJson({ assets: true });
  const assets = statsJSON.assets.reduce((memo, { name }) => {
    memo[name] = `/${name}`
    return memo;
  }, {});
  const app = express();
  app.get('/', async function (req, res) {
    render(req, res, assets);
  });
  app.use(static('build'));
  app.listen(8080, () => console.log('server started on port 8080'));
});
```

5.3 render.js

render.js

```
import React from 'react';
import App from "./src/App";
import { renderToString } from "react-dom/server";
function render(req, res, assets) {
  const html = renderToString(<App />);
  res.statusCode = 200;
  res.setHeader("Content-type", "text/html;charset=utf8");
  res.send(`

        Document

        ${html}
        ${assets[<span class="hljs-string">'main.js'</span>]}</span>">

  `);
}
module.exports = render;
```

6. React18之后的服务器端渲染(SSR)

- Streaming SSR with selective hydration(选择性水合)
- 像流水一样，打造一个从服务端到客户端持续不断的渲染管线，而不是 `renderToString`那样一次性渲染机制
- 在React18之前，这个操作必须是整体性的，而水合的过程可能比较慢，会引起局部卡顿，所以选择性水合可以在局部进行水合
- 服务端渲染把简单的 `res.send`改为 `res.socket`,这样的渲染就从单次行为转变为持续性行为
- 打破了以前串行的限制，优化前端的加载速度和可交互所需等待时间
- 服务器端的流式HTML使用 `renderToPipeableStream`
- 客户端的选择性 `Hydration` 使用 `<suspense></suspense>`

6.1 render.js

render.js

```
import React from 'react';
import App from "./src/App";
import { renderToPipeableStream } from "react-dom/server";
function render(req, res, assets) {
+ const { pipe } = renderToPipeableStream(
+   ,
+   {
+     bootstrapScripts: [assets['main.js']],
+     onShellReady() {
+       res.statusCode = 200;
+       res.setHeader("Content-type", "text/html;charset=utf8");
+       res.write(`
+
+
+
+
+
+
+             Document
+
+           `);
+       pipe(res);
+       res.write(``)
+     }
+   }
+ );
}
module.exports = render;
```

6.2 App.js

src\App.js

```
+import React, { Suspense } from 'react';
import Header from './Header';
import Footer from './Footer';
+import User from './User';
//const LazyUser = React.lazy(() => import('../User'));
function App() {
  return (
    <>

+     loading User...}>
+
+

    </>
  );
}
export default App
```

```

  ID:1

document.getElementById('B:0').replaceChildren(document.getElementById('S:0'));
```

6.3 User.js

src\User.js

```
import React from 'react';
+const userPromise = fetchUser(1);
+const userResource = wrapPromise(userPromise);
+function User() {
+  const user = userResource.read();
+  return ID:{user.id}
+}
+export default User;
+
+function fetchUser(id) {
+  return new Promise((resolve) => {
+    setTimeout(() => {
+      resolve({ id });
+    }, 80000);
+  });
+}
+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;
+      }
+    }
+  };
+}
```

7. useId

- useId #22644 (https://github.com/facebook/react/pull/22644)
- `Math.random()`生成的ID在客户端、服务端不匹配
- `useId`可以生成稳定、唯一的id
- 每个id代表该组件在组件树中的层级结构
- 层级本身就能作为服务端、客户端之间不变的标识

src\Footer.js

```
import React, { useId } from 'react';
function Footer() {
+  const id = useId();
  return (

+      are you ok?

+

  );
}
export default Footer;
```

8. 支持路由

8.1 安装

```
npm install react-router-dom --save
```

8.2 routesConfig.js

src\routesConfig.js

```
import React from 'react';
import Home from './routes/Home';
import Counter from './routes/Counter';
export default [
  {
    path: '/',
    element: <Home />,
    index: true
  },
  {
    path: '/counter',
    element: <Counter />
  }
]
```

8.3 Home.js

src\routes\Home.js

```
import React from 'react';
function Home() {
  return (
    <div>
      Home
    div>
  )
}
export default Home;
```

8.4 Counter.js

src\routes\Counter.js

```
import React, { useState } from 'react';
function Counter() {
  const [number, setNumber] = useState(0);
  return (
    <div>
      <p>{number}p>
      <button onClick={() => setNumber(number + 1)}>+button>
    div>
  )
}
export default Counter;
```

src\components\Header.js

```
import React from 'react';
import { Link } from 'react-router-dom';
function Header() {
  return (
    <ul>
      <li><Link to="/">HomeLink>li>
      <li><Link to="/counter">CounterLink>li>
    ul>
  )
}
export default Header
```

8.6 src\App.js

src\App.js

```
import React from 'react';
+import { useRoutes } from 'react-router-dom';
+import routesConfig from './routesConfig';
+import Header from './components/Header';
function App() {
  return (
+   <>
+
+     {useRoutes(routesConfig)}
+   </>
  );
}
export default App
```

8.7 src\index.js

src\index.js

```
import React from 'react';
import { hydrateRoot } from 'react-dom/client';
import App from './App';
+import { BrowserRouter } from 'react-router-dom';
const root = document.getElementById('root');
+hydrateRoot(root, );
```

8.8 render.js

render.js

```
import React from 'react';
+import { StaticRouter } from "react-router-dom/server";
import { renderToPipeableStream } from "react-dom/server";
import App from "./src/App";
function render(req, res, assets) {
  const { pipe } = renderToPipeableStream(
+   ,
    {
      bootstrapScripts: [assets['main.js']],
      onShellReady() {
        res.statusCode = 200;
        res.setHeader("Content-type", "text/html;charset=utf8");
        res.write(`

             Document

           `);
        pipe(res);
        res.write(``)
      }
    }
  );
}
module.exports = render;
```