Renderização no servidor

A situação de uso mais comum para a renderização do lado do servidor, é manipular a renderização inicial quando um usuário (ou rastreador de mecanismo de pesquisa) solicita sua aplicação.

Quando o servidor recebe a solicitação, ele renderiza o(s) componente(s) requerido(s) em uma cadeia HTML e o envia como uma resposta ao cliente. A partir desse momento, o cliente assume as funções de renderização.

Material-UI no servidor

O Material-UI foi projetado em base com garantias de renderização no servidor, mas cabe a você certificar-se de que ele será integrado corretamente. É importante fornecer a página com o CSS necessário, caso contrário a página irá renderizar somente o HTML até o CSS ser injetado pelo cliente, causando uma tremulação (FOUC). Para injetar o estilo no cliente, precisamos:

1. Create a fresh, new `emotion cache` instance on every request.
2. Renderize a árvore React com o coletor do lado do servidor.
3. Capture o CSS.
4. Passe o CSS junto ao cliente.

On the client-side, the CSS will be injected a second time before removing the server-side injected CSS.

Setting up

No passo a passo a seguir, vamos ver como configurar a renderização do lado do servidor.

O tema

Crie um tema que será compartilhado entre o cliente e o servidor:

`theme.js`

```
import { createTheme } from '@material-ui/core/styles';
import red from '@material-ui/core/colors/red';

// Cria a instância do tema.
const theme = createTheme({
  palette: {
    primary: {
      main: '#556cd6',
    },
    secondary: {
      main: '#19857b',
    },
```

```
    error: {
      main: red. A400,
    },
  },
});

export default theme;
```

O lado do servidor

A seguir um esboço para o aspecto do que o servidor deve lidar. We are going to set up an Express middleware using app.use to handle all requests that come into the server. If you're unfamiliar with Express or middleware, know that the `handleRender` function will be called every time the server receives a request.

`server.js`

```
import express from 'express';

// Vamos preenchê-las nas seções a seguir.
function renderFullPage(html, css) {
  /* ... */
}

function handleRender(req, res) {
  /* ... */
}

const app = express();

// Isso é acionado toda vez que o servidor recebe uma solicitação.
app.use(handleRender);

const port = 3000;
app.listen(port);
```

Manipulando as requests

A primeira coisa que nós precisamos fazer em cada request é criar um novo `emotion cache`.

When rendering, we will wrap `App`, the root component, inside a `CacheProvider` and `ThemeProvider` to make the style configuration and the `theme` available to all components in the component tree.

The key step in server-side rendering is to render the initial HTML of the component **before** we send it to the client-side. Para fazer isso, usamos React-DOMServer.renderToString().

Material-UI is using emotion as its default styled engine. We need to extract the styles from the emotion instance. For this, we need to share the same cache configuration for both the client and server:

getCache.js

```
import createCache from '@emotion/cache';

export default function getCache() {
  const cache = createCache({ key: 'css' });
  cache.compat = true;
  return cache;
}
```

With this we are creating new emotion cache instance and using this to extract the critical styles for the html as well.

Vamos ver como isso é passado na função `renderFullPage`.

```
import express from 'express';
import * as React from 'react';
import ReactDOMServer from 'react-dom/server';
import CssBaseline from '@material-ui/core/CssBaseline';
import { ThemeProvider } from '@material-ui/core/styles';
import createEmotionServer from '@emotion/server/create-instance';
import App from './App';
import theme from './theme';
import getCache from './getCache';

function handleRender(req, res) {
  const cache = getCache();
  const { extractCriticalToChunks, constructStyleTagsFromChunks } =
    createEmotionServer(cache);

  // Render the component to a string.
  app.use(handleRender);

const port = 3000;
app.listen(port);
  const html = ReactDOMServer.renderToString(
    <CacheProvider value={cache}>
      <ThemeProvider theme={theme}>
        {/* CssBaseline kickstart an elegant, consistent, and simple baseline to build upon
        <CssBaseline />
        <App />
      </ThemeProvider>
    </CacheProvider>,
  );
```

```
  // Grab the CSS from emotion
  const emotionChunks = extractCriticalToChunks(html);
  const emotionCss = constructStyleTagsFromChunks(emotionChunks);

  // Send the rendered page back to the client.
  res.send(renderFullPage(html, emotionCss));
}

const app = express();

app.use('/build', express.static('build'));

// This is fired every time the server-side receives a request.
app.use(handleRender);

const port = 3000;
app.listen(port);
```

Inject initial component HTML and CSS

The final step on the server-side is to inject the initial component HTML and CSS into a template to be rendered on the client-side.

```
function renderFullPage(html, css) {
  return `
    <!DOCTYPE html>
    <html>
      <head>
        <title>My page</title>
        ${css}
        <meta name="viewport" content="initial-scale=1, width=device-width" />
        <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,
      </head>
      <body>
        <div id="root">${html}</div>
      </body>
    </html>
  `;
}
```

The client-side

The client-side is straightforward. All we need to do is use the same cache configuration as the server-side. Vamos dar uma olhada no arquivo do cliente:

`client.js`

```
import * as React from 'react';
import ReactDOM from 'react-dom';
import CssBaseline from '@material-ui/core/CssBaseline';
import { ThemeProvider } from '@material-ui/core/styles';
import { CacheProvider } from '@emotion/react';
import App from './App';
import theme from './theme';
import getCache from './getCache';

function Main() {
  return (
    <CacheProvider value={getCache}>
      <ThemeProvider theme={theme}>
        {/* CssBaseline kickstart an elegant, consistent, and simple baseline to build upon.
        <CssBaseline />
        <App />
      </ThemeProvider>
    </CacheProvider>
  );
} ReactDOM.hydrate(<Main />, document.querySelector('#root'));
```

Implementações de referência

We host different reference implementations which you can find in the GitHub repository under the `/examples` folder:

- A implementação de referência deste tutorial
- Gatsby
- Next.js (TypeScript version)

Resolução de problemas

Confira a resposta no FAQ: Minha aplicação não é renderizada corretamente no servidor.