useMediaQuery

这是 React 的 CSS 媒体查询 （Media queries）hook。 它监听与 CSS 媒体查询的匹配的内容。 它允许根据查询的结果是否匹配来渲染组件。

以下是一些重要的特点：

- ⚛️ 它有一个符合用户使用习惯的 React API。
- 🚀 它是高性能的，原理是通过观测文档的媒体查询值发生更改，而不是使用定期轮询的方法来监听其结果。
- 📦 1kB 已压缩的包。
- 🤘 它支持服务器端渲染。

{{"component": "modules/components/ComponentLinkHeader.js", "design": false}}

简单的媒体查询

你应该将媒体查询提供给 hook 作为第一个参数。 媒体查询的字符串可以是任何有效的 CSS 媒体查询，例如 `'(prefers-color-scheme: dark)'` 。

{{"demo": "SimpleMediaQuery.js", "defaultCodeOpen": true}}

⚠️ 由于每个浏览器的限制，你不能使用 `'print'` ，例如 Firefox 上的这个问题。

使用 Material-UI 的断点辅助功能

按照如下所示的例子，你可以这样使用 Material-UI 的 断点辅助功能：

```
import { useTheme } from '@material-ui/core/styles';
import useMediaQuery from '@material-ui/core/useMediaQuery';

function MyComponent() {
  const theme = useTheme();
  const matches = useMediaQuery(theme.breakpoints.up('sm'));

  return <span>{`theme.breakpoints.up('sm') matches: ${matches}`}</span>;
}
```

{{"demo": "ThemeHelper.js", "defaultCodeOpen": false}}

或者你也可以使用一个回调函数，其第一个参数是 theme：

```
import useMediaQuery from '@material-ui/core/useMediaQuery';

function MyComponent() {
  const matches = useMediaQuery((theme) => theme.breakpoints.up('sm'));

  return <span>{`theme.breakpoints.up('sm') matches: ${matches}`}</span>;
}
```

⚠️ 由于这个方法**没有默认的**主题支持，所以你必须将它注入到父级主题提供者（parent theme provider）中。

使用 JavaScript 的语法

你可以使用 json2mq 来从 JavaScript 对象中生成媒体查询字符串。

{{"demo": "JavaScriptMedia.js", "defaultCodeOpen": true}}

测试

你需要在测试环境中实现 matchMedia。

例如：暂时还不支持 jsdom。 所以你应来兼容（polyfill）它。 我们推荐使用 css-mediaquery 来创造一个模拟环境从而达到兼容的目的。

```
import mediaQuery from 'css-mediaquery';

function createMatchMedia(width) {
  return (query) => ({
    matches: mediaQuery.match(query, {
      width,
    }),
    addListener: () => {},
    removeListener: () => {},
  });
}

describe('MyTests', () => {
  beforeAll(() => {
    window.matchMedia = createMatchMedia(window.innerWidth);
  });
});
```

仅在客户端渲染

要和服务器进行同步使用（hydration），hook 需要渲染两次。 第一次使用 `false` 表示服务端的值，第二次使用已解析的值。 这个双向渲染周期带有一个缺点。 速度较慢。 如果你只需要**客户端渲染**，那么你可以设置 `noSsr` 选项为 `true`。

```
const matches = useMediaQuery('(min-width:600px)', { noSsr: true });
```

或者你可以通过全局主题设置来启用它：

```
const theme = createTheme({
  components: {
    MuiUseMediaQuery: {
      defaultProps: {
        noSsr: true,
      },
    },
  },
});
```

服务端渲染

> ⚠️ *从根本上来看，服务端渲染和客户端的媒体查询是矛盾的。 所以你需要在其中取舍。 支持只能是部分的。*

你可以先尝试依赖于客户端的 CSS 媒体查询。 例如，你可以使用：

- `<Box display>`
- `themes.breakpoints.up(x)`
- or `sx prop`

如果上述的方案都不可用，那么你也可以继续阅读本节文档的其余内容。

首先，你需要从服务端上猜测客户端请求的特征。 你可以选择使用：

- **用户代理（User agent）**。 解析客户端上用户代理的字符串来提取信息。 我们推荐使用 ua-parser-js 来解析用户代理信息。
- **客户端提示（Client hints）**。 读取客户端向服务器发送的提示。 请注意，并不是所有浏览器都会支持 此功能。

最后，你需要为 `useMediaQuery` 提供一个具有预先猜测特征的 matchMedia 来实现。 我们建议使用 css-mediaquery 来模拟 matchMedia 环境。

例如，在服务端上：

```
import ReactDOMServer from 'react-dom/server';
import parser from 'ua-parser-js';
import mediaQuery from 'css-mediaquery';
import { ThemeProvider } from '@material-ui/core/styles';

function handleRender(req, res) {
  const deviceType = parser(req.headers['user-agent']).device.type || 'desktop';
  const ssrMatchMedia = (query) => ({
    matches: mediaQuery.match(query, {
      // 浏览器的 css 宽度预计值
      width: deviceType === 'mobile' ? '0px' : '1024px',
    }),
  });

  const html = ReactDOMServer.renderToString(
    <ThemeProvider
      theme={{
        props: {
          // 更改 useMediaQuery 的默认选项
          MuiUseMediaQuery: {
            ssrMatchMedia,
          },
        },
      }}
    >
      <App />
    </ThemeProvider>,
  );
```

```
  // …
}
      width: deviceType === 'mobile' ? '0px' : '1024px',
    }),
  });

  const html = ReactDOMServer.renderToString(
    <ThemeProvider
      theme={{
        props: {
          // 更改 useMediaQuery 的默认选项
          MuiUseMediaQuery: {
            ssrMatchMedia,
          },
        },
      }}
    >
      <App />
    </ThemeProvider>,
  );

  // …
}
      width: deviceType === 'mobile' ? '0px' : '1024px',
    }),
  });

  const theme = createTheme({
    components: {
      // Change the default options of useMediaQuery
      MuiUseMediaQuery: {
        defaultProps: {
          ssrMatchMedia,
        },
      },
    },
  });

  const html = ReactDOMServer.renderToString(
    <ThemeProvider theme={theme}>
      <App />
    </ThemeProvider>,
  );

  // …
}
```

{{"demo": "ServerSide.js", "defaultCodeOpen": false}}

确保您提供相同的自定义匹配媒体实现到客户端，这样能够保证注水渲染的匹配。

从 `withWidth()` 迁移

`withWidth()` 高阶组件注入了页面的屏幕宽度。 您可以使用 `useWidth` hook 来实现相同的操作：

{{"demo": "UseWidth.js"}}

API

`useMediaQuery(query, [options]) => matches`

参数

1. `query` (*string* | *func*): A string representing the media query to handle or a callback function accepting the theme (in the context) that returns a string.
2. `options` (*object* [optional]):

- `options.defaultMatches` (*bool* [optional]): As `window.matchMedia()` is unavailable on the server, we return a default matches during the first mount. 默认值为 `false` 。 默认值为 `false` 。
- `options.matchMedia` (*func* [optional]): You can provide your own implementation of *matchMedia*. 用其您可以处理一个 iframe 内容窗口。 用其您可以处理一个 iframe 内容窗口。
- `options.noSsr` (*bool* [optional])：默认为 `false` 。 要和服务器进行同步使用（hydration），hook 需要渲染两次。 A first time with `false`, the value of the server, and a second time with the resolved value. 这个双向渲染周期带有一个缺点。 速度较慢。 如果你只需要 **客户端**渲染，那么可以将该选项设置为 `true` 。
- `options.ssrMatchMedia` (*func* [optional]): You can provide your own implementation of *matchMedia* in a server-side rendering context.

注意：你可以使用主题的 默认属性 功能和 `MuiUseMediaQuery` 键（key）来更改默认的选项。

返回结果

`matches` ：如果文档当前能够匹配这个媒体查询，Matches 则为 `true` ，否则为 `false` 。

例子

```
import * as React from 'react';
import useMediaQuery from '@material-ui/core/useMediaQuery';

export default function SimpleMediaQuery() {
  const matches = useMediaQuery('(min-width:600px)');

  return <span>{`(min-width:600px) matches: ${matches}`}</span>;
}
```