代码分割

打包是个非常棒的技术，但随着应用增长，代码包也将随之增长。尤其是在整合了体积巨大的第三方库的情况下。需要关注代码包中所包含的代码，以避免因体积过大而导致加载时间过长。

对应用进行代码分割能够帮助“懒加载”当前用户所需要的内容，能够显著地提高应用性能。尽管并没有减少应用整体的代码体积，但可以避免加载用户永远不需要的代码，并在初始加载的时候减少所需加载的代码量。

import()

import() 方法可以用于动态加载资源，返回一个 promise。

使用前：

```
import { add } from './math';

console.log(add(16, 26));
```

使用后：

```
import("./math").then(math => {
  console.log(math.add(16, 26));
});
```

React.lazy

React.lazy 函数能像渲染常规组件一样处理动态引入的组件。

使用之前：

```
import OtherComponent from './OtherComponent';
```

将 Suspense 组件置于懒加载组件之上的任何位置。你甚至可以用一个 Suspense 组件包裹多个懒加载组件。fallback 属性接受任何在组件加载过程中你想展示的 React 元素。

```
import React, { Suspense } from 'react';

const OtherComponent = React.lazy(() => import('./OtherComponent'));
const AnotherComponent = React.lazy(() => import('./AnotherComponent'));

function MyComponent() {
  return (
    <div>
```

```
        <Suspense fallback={<div>Loading...</div>}>
          <section>
            <OtherComponent />
            <AnotherComponent />
          </section>
        </Suspense>
      </div>
    );
  }
```

异常捕获边界（Error Boundaries）

如果模块加载失败（如网络问题），它会触发一个错误。你可以通过异常捕获边界（Error boundaries）技术来处理这些情况，以显示良好的用户体验并管理恢复事宜。

基于路由的代码分割

从路由开始分隔代码是比较好的选择，大多数网络用户习惯于页面之间能有个加载切换过程。