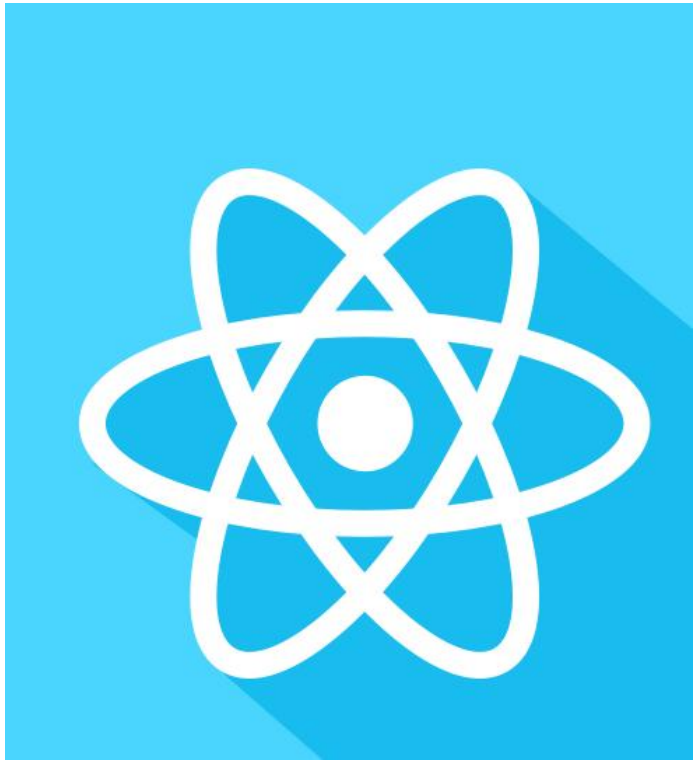
数据可视化 实践课11

1

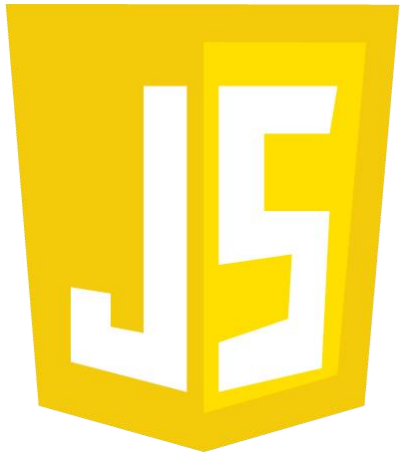
React.js介绍

为什么需要使用框架？

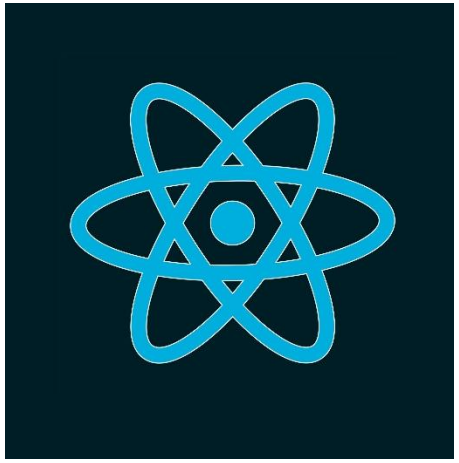
- HTML，CSS，JS
  - 大型的项目很难做到代码的复用
  - 写法过于自由
- 框架
  - **组件**这一概念使得代码结构更加清晰
  - 封装和复用
  - 方便使用各种JS库



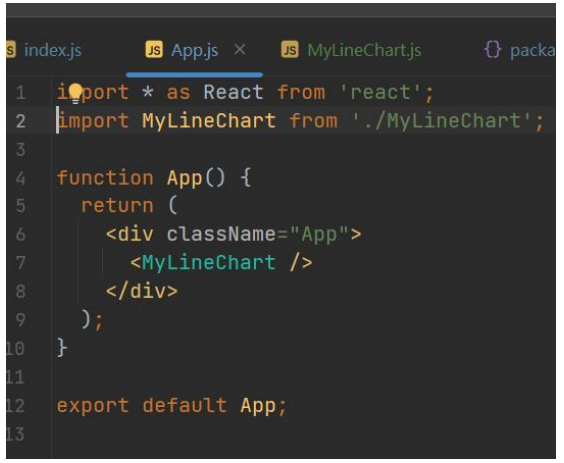
什么是React?



JavaScript Library



Facebook开发



对于初学者友好

JSX语法

- 可以视为JS与HTML的结合
- 下面的例子中函数的返回值：一些HTML标签的组合

```
function AboutPage() {
  return (
    <>
      <h1>About</h1>
      <p>Hello there.<br />How do you do?</p>
    </>
  );
}
```

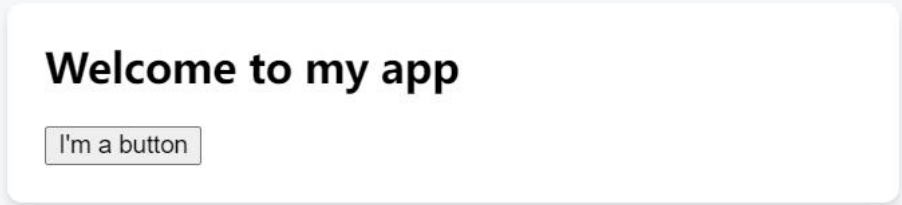
组件（component）

- 拥有了自身的逻辑，小到一个button，大到一整个界面
- 根据传入的参数（props）和自身的状态（state），渲染出UI界面

App.js

Download Reset Fork

```
1  function MyButton() {
2    return (
3      <button>
4        I'm a button
5      </button>
6    );
7  }
8
9  export default function MyApp() {
10   return (
11     <div>
12       <h1>Welcome to my app</h1>
13       <MyButton />
14     </div>
15   );
16 }
```



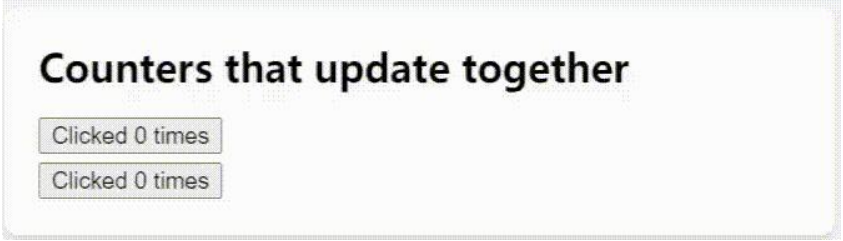
状态（State）

- State为组件某一时刻自身的状态（snapshot），State的改变会导致组件的重新渲染
- useState这个函数返回某个State，以及改变这个State数值的函数（不可直接对State赋值）
- handleClick为button的回调函数，响应click事件，使count加1

```
function MyButton() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      Clicked {count} times
    </button>
  );
}
```

```
export default function MyApp() {
  return (
    <div>
      <h1>Counters that update separately</h1>
      <MyButton />
      <MyButton />
    </div>
  );
}
```

父组件传入的参数（props）

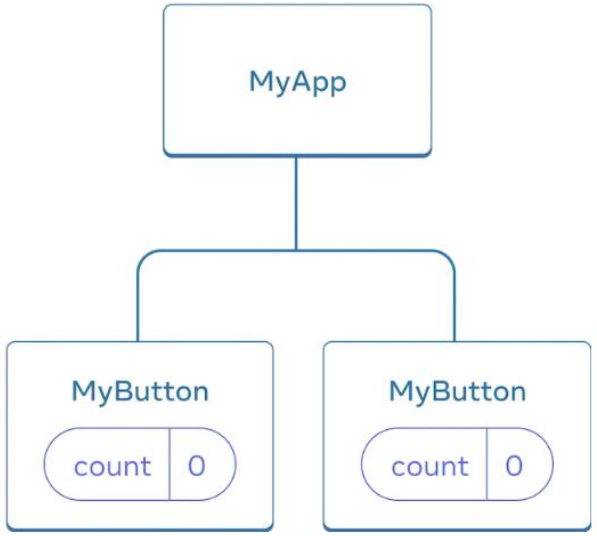


- 父组件（MyApp）维护一个公共的State，作为参数传给子组件，**单向数据流**
- 子组件（两个MyButton）调用传入的onClick函数，改变父组件的状态，实现共同更新

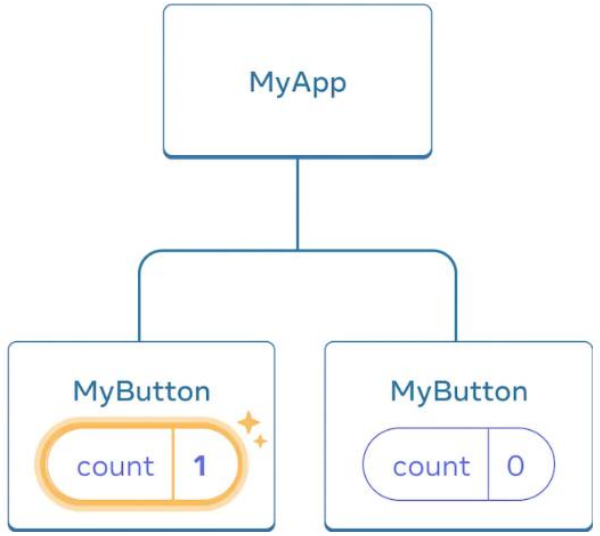
```
function MyButton({ count, onClick }) {
  return (
    <button onClick={onClick}>
      Clicked {count} times
    </button>
  );
}
```

```
export default function MyApp() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <div>
      <h1>Counters that update together</h1>
      <MyButton count={count} onClick={handleClick} />
      <MyButton count={count} onClick={handleClick} />
    </div>
  );
}
```

组件间数据的共享

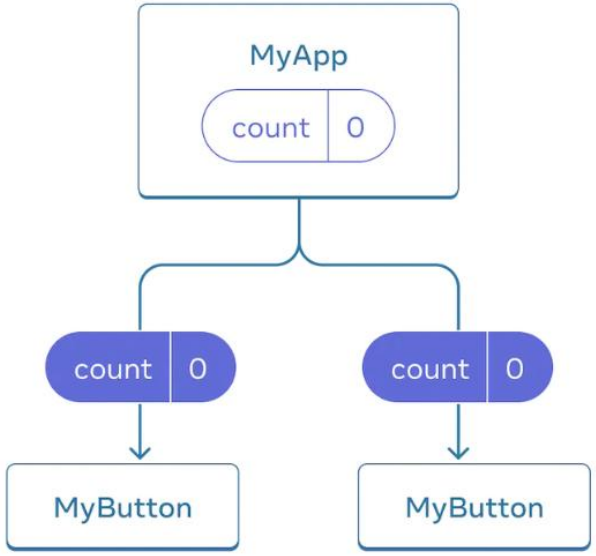


Initially, each `MyButton`'s `count` state is `0`

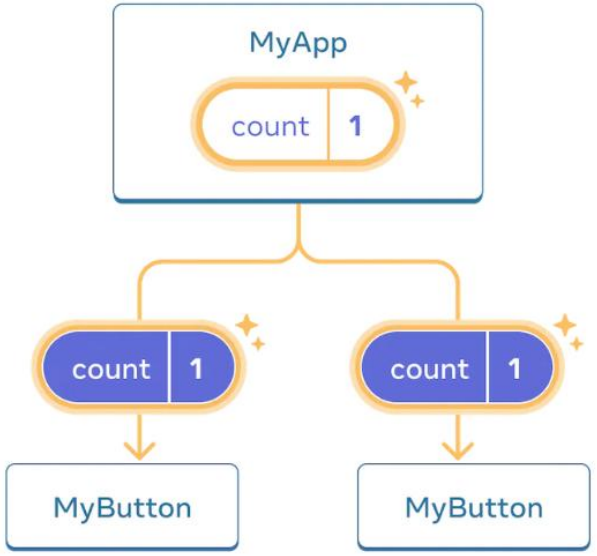


The first `MyButton` updates its `count` to `1`

子组件的状态是独立的



Initially, `MyApp`'s `count` state is `0` and is passed down to both children



On click, `MyApp` updates its `count` state to `1` and passes it down to both children

将状态提升至父组件，实现数据共享

2

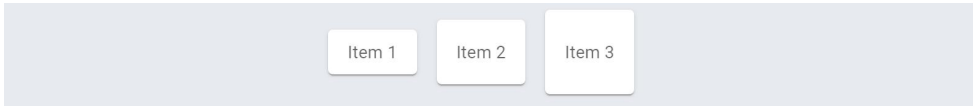
简单的DashBoard的构建

使用组件库MUI进行复杂的布局

- MUI　https://mui.com/material-ui/　Google的UI库
  - Box（相当于div）

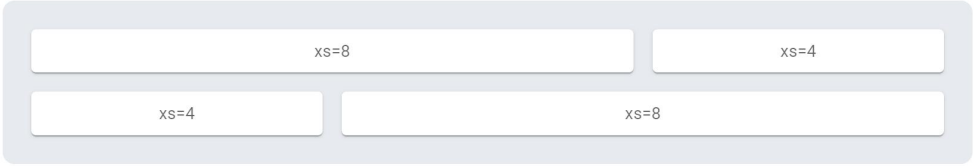
```
<Box component="span" sx={{ p: 2, border: '1px dashed grey' }}>
  <Button>Save</Button>
</Box>
```

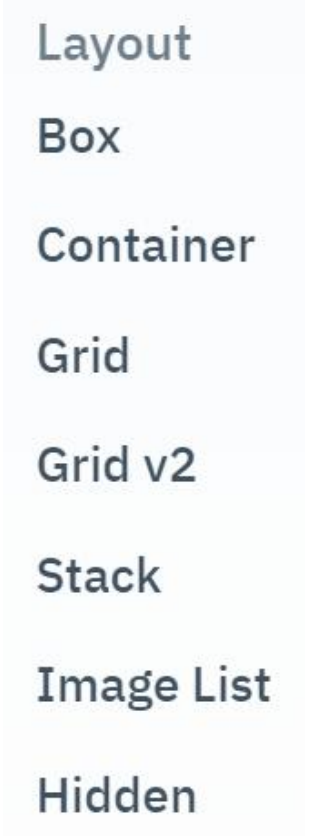
  - Stack（一维布局，flex）https://mui.com/material-ui/react-stack/



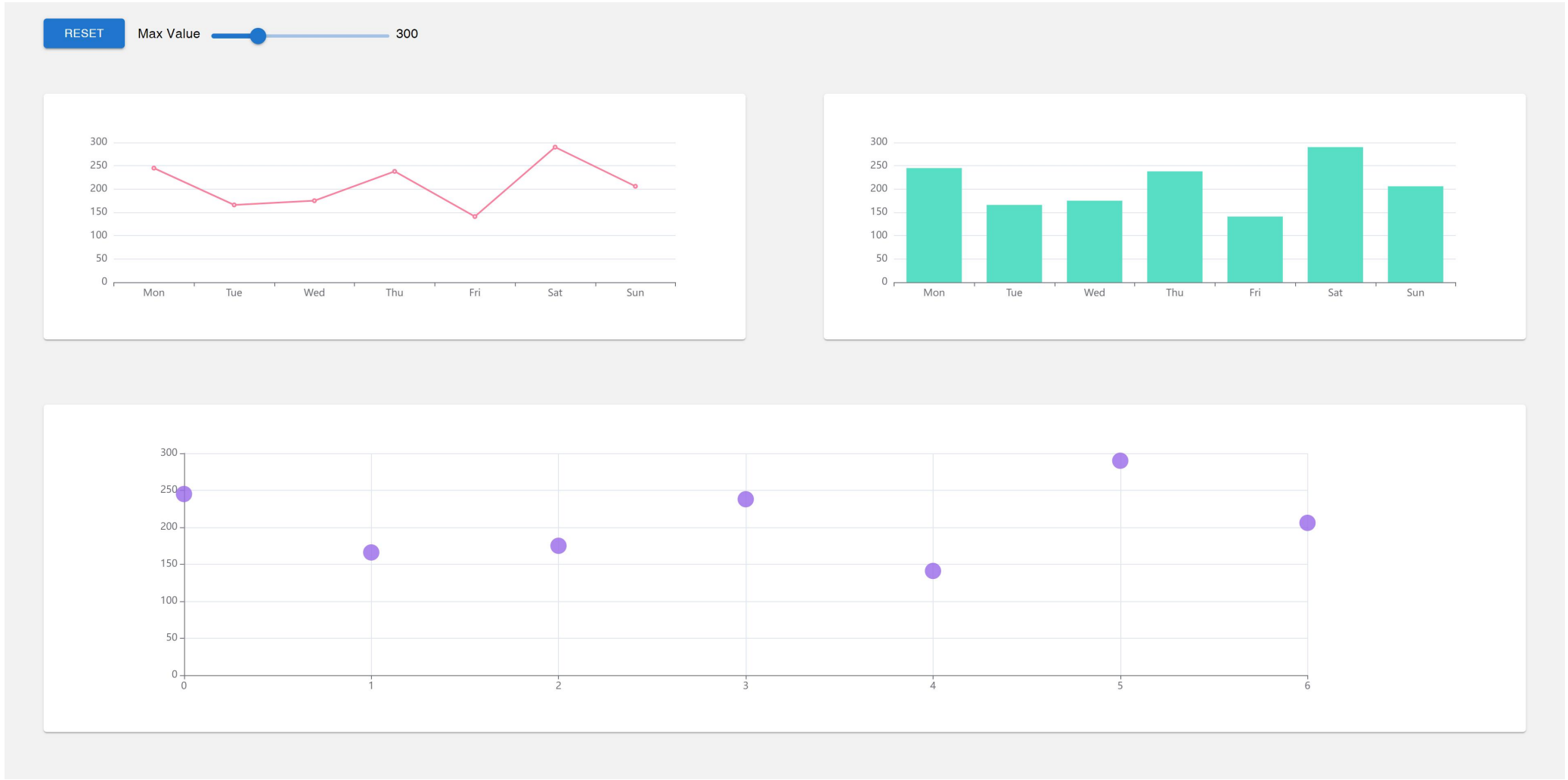

```
<Stack
  direction="row"
  justifyContent="center"
  alignItems="center"
  spacing={2}
>
```

  - Grid（二维布局）





简单的dashboard



目录结构

```
28  const DashBoard = () => {
29    const [seriesData, setSeriesData] = React.useState([150, 230, 224, 218, 135, 147, 260]);
30    const [maxChartValue, setMaxChartValue] = React.useState(300);
```

index.html： 28行的DashBoard为最大的组件

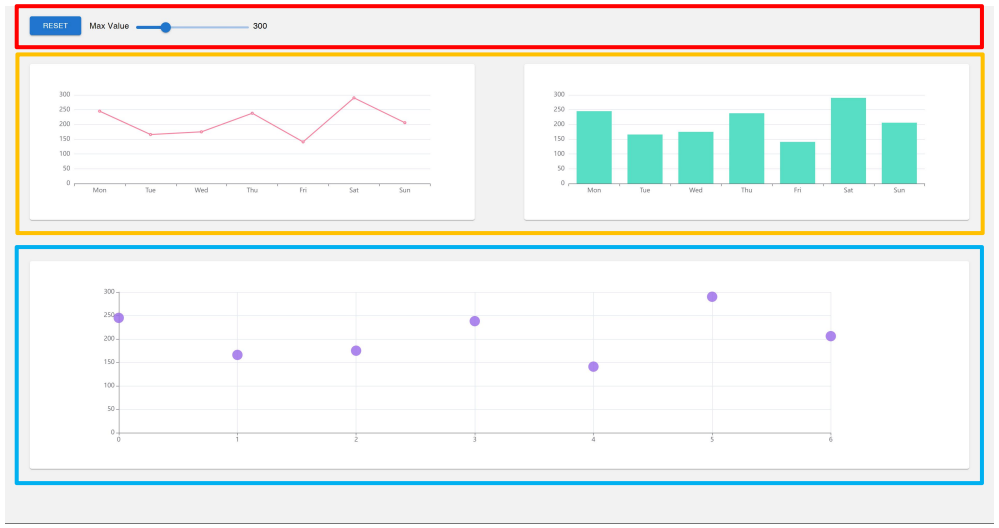
```
23  <body>
24    <div id="root"></div>
25    <script type="text/babel">
132     ReactDOM.render(
133       <DashBoard />,
134       document.getElementById('root')
135     );
```

js文件夹： 包含一些框架的代码，（已经压缩过，只有计算机看得懂）

components文件夹： 包含了一些自定义的小组件

布局代码

```
return (
  <Stack>
    <Stack direction={'row'} alignItems={'flex-end'} mx={5} my={2} spacing={2}>
      <Button sx={{width: 100}} variant={'contained'} onClick={handleReset}>
        Reset
      </Button>
      <Stack direction={'row'} width={400} spacing={1}...>
    </Stack>
    <Stack direction={'row'} justifyContent={'space-between'} m={5}>
      <Card>
        <LineChart option={lineChartOption} />
      </Card>
      <Card>
        <BarChart option={barChartOption} />
      </Card>
    </Stack>
    <Box m={5}>
      <Card>
        <ScatterChart option={scatterChartOption} />
      </Card>
    </Box>
  </Stack>
);
}
```

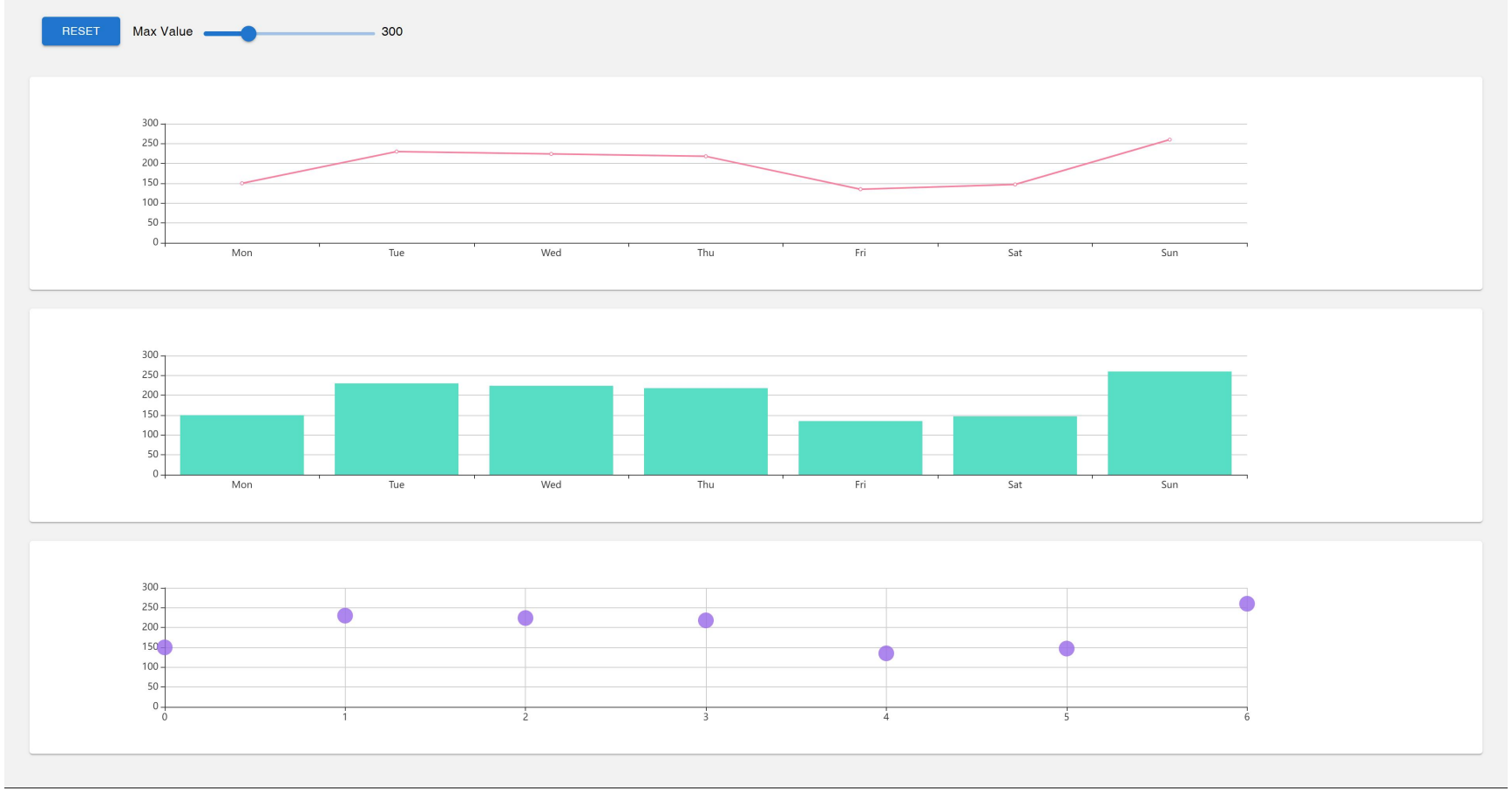


ScatterChart.js

```
const ScatterChart = (props) => {
  const { option } = props;

  return (
    <ReactEcharts option={option} style={{ height: '40vh', width: '90vw' }} />
  );
}

export default ScatterChart;
```

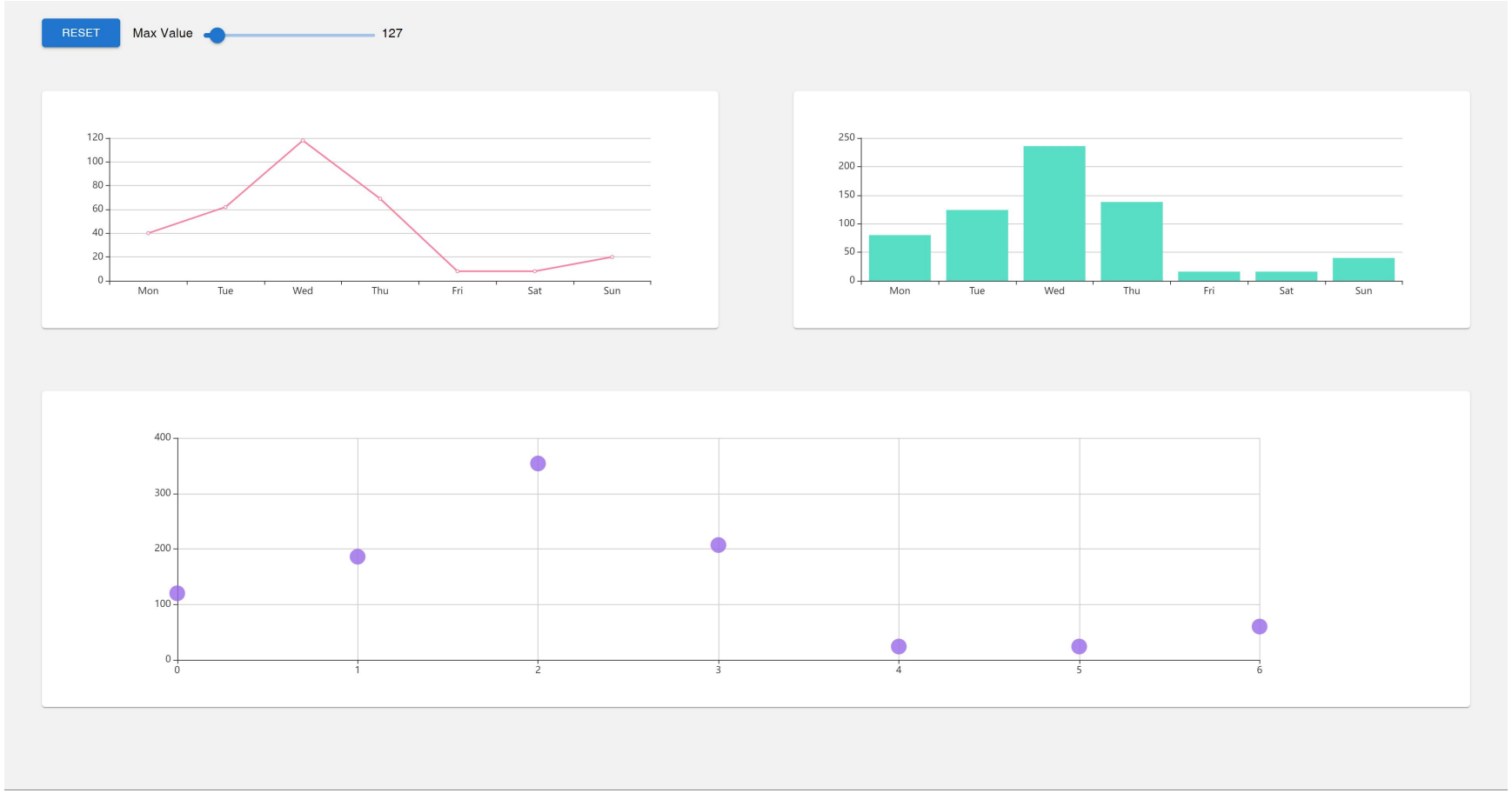
练习2-1

改变三个图表的布局为纵向或横向布局，要求有一定间距



练习2-2

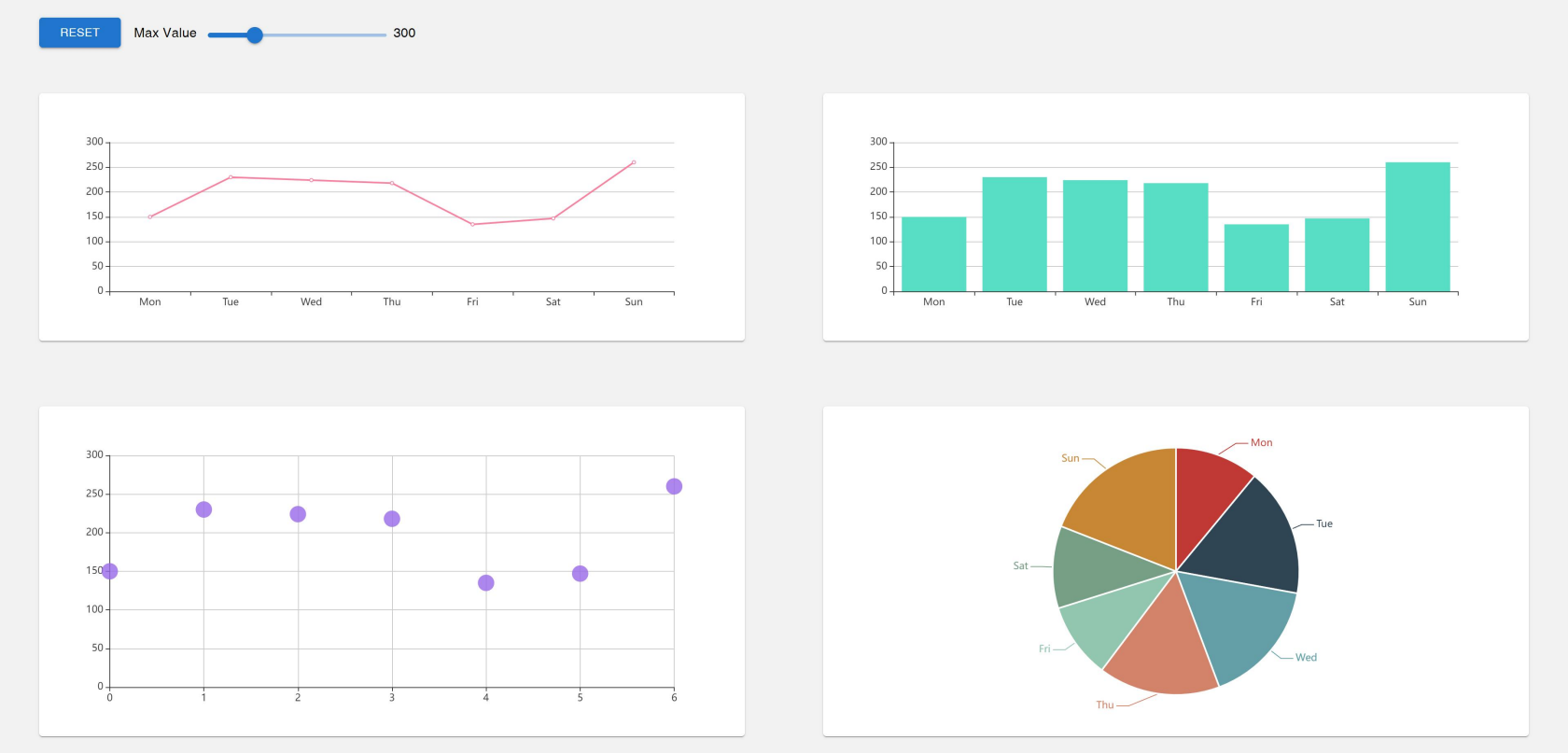
使柱状图的所有数值为折线图对应的2倍，使散点图的所有数值为折线图对应的3倍



练习2-3

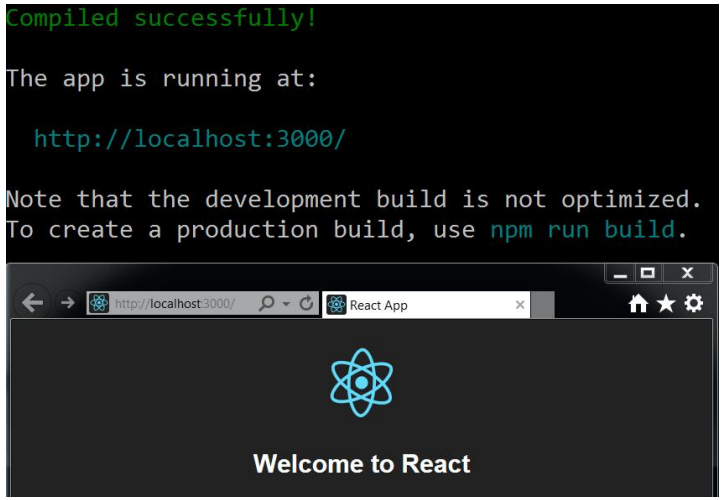
添加一个饼状图，使用其它子图同样的数据

- 需要新建一个PieChart.js表示饼状图，并在index.html中导入PieChart.js
- 添加pieChartOption，传入PieChart（echarts饼状图配置可参考官网）



NPM安装相关（了解）

- Node： 一个JS后端，设置环境变量 https://nodejs.org/zh-cn/download
- Npm：包管理工具
- 换国内源：npm config set registry https://registry.npm.taobao.org
- 运行 npx create-react-app my-app 快速创建React项目
- npm start 运行项目
- 添加需要的package，如npm install echarts-for-react
- 从github拷贝下来的项目，npm install 安装package.json中的所有依赖



package.json

```
{
  "name": "dashboard-demo",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.10.6",
    "@emotion/styled": "^11.10.6",
    "@mui/material": "^5.12.0",
    "echarts-for-react": "^3.0.2",
    "gh-pages": "^5.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "predeploy": "npm run build",
    "deploy": "gh-pages -d build"
  },
```