组件实践（2）：组件的内部实现

上一小节中，通过设计一个“秒表”应用，我们学习了组件的接口定义的原则和方法，但那只是搭建了一个骨架，在这一小节中，我们就给这个骨架填充血肉，制造出能够运转的“秒表”。

我们不大可能一次就写出完美的代码，软件开发本来就是一个逐渐精进的过程，但是我们应该努力让代码达到这样的要求：

1. 功能正常；
2. 代码整洁；
3. 高性能。

初始化应用框架

我们使用 Facebook 提供的 create-react-app 来创建我们的 React 应用。严格说来，如果要开发一款真正大型的应用，用 create-react-app 并不合适，或者至少 create-react-app 的默认配置并不合适，真正的应用需要对 webpack 等做更精细的配置，但是，create-react-app 足够简单易用，从学习 React 的角度来看，真的是非常合适了。

如果你的机器上还没有安装 create-react-app，那么就使用下面的命令来全局安装：

```
npm install -g create-react-app
```

然后，找一个合适的目录，使用下面的命令来创建我们的应用框架，在这里，我们的应用名字叫 `basic_stop_watch`。

```
create-react-app basic_stop_watch --use-npm
```

create-react-app 会优先使用 yarn 来安装依赖的 npm 包，但是不知道读者机器上有没有安装 yarn，所以用 `--use-npm` 参数强制 create-react-app 使用传统的 npm 工具来安装依赖的包，这样能够保持一致。如果你更喜欢 yarn，不使用 `--use-npm` 这个参数就可以了。

创建应用框架需要花费一些时间，因为现在依赖的包实在太多太细碎了，在完成之后，会创建一个 `basic_stop_watch` 目录，进入这个目录，运行下面给的命令，就可以启动给一个基本的 React 应用。

```
npm start
```

我们不会花费太多时间介绍 create-react-app 的其他功能和代码结构，因为我们有更重要的事情要做，那就是根据上一小节的组件设计来制造“秒表”相关组件。

构建 StopWatch

在上一小节中，我们已经确定了要用四个组件组合来实现“秒表”，这四个组件分别是 StopWatch、MajorClock、ControlButtons 和 SplitTimes。

我们在写代码之前要做的第一个决定就是，要把这些组件放在哪里？是放在不同的文件中？还是放在一个文件中就好？

表面上看，把所有组件放在一个文件中也行得通，但是，将来维护代码的朋友可能会很抓狂，想要修改 ControlButtons 这个组件，但是从文件目录里找不到对应文件，这样很不方便。

所以，从达到“代码整洁”的目的来说，应该每个组件都有一个独立的文件，然后这个文件用 `export default` 的方式导出单个组件。

比如，我们会在 `src` 目录下为 ControlButtons 创建一个 `ControlButtons.js` 文件，最初内容像下面这样：

```
import React from 'react';

const ControlButtons = () => {
  //TODO: 实现ControlButtons
};

export default ControlButtons;
```

第一行导入 React，虽然目前没有派上什么用场，但是任何 JSX 都需要 React，很快我们在实现 ControlButtons 这个控件的内容时，就要写 JSX，所以导入 React 是必需的。

最后一行用 `export default` 的方式导出 ControlButtons，这样，在其他组件中就可以用下面的方式导入：

```
import ControlButtons from './ControlButtons';
```

在上一节中我们已经设计好了 ControlButtons 可以接受的 props，我们重试 ControlButtons 的实现代码，如下：

```
const ControlButtons = (props) => {
  const {activated, onStart, onPause, onReset, onSplit} = props;
  if (activated) {
    return (
      <div>
        <button onClick={onSplit}>计次</button>
        <button onClick={onPause}>停止</button>
      </div>
    );
  } else {
    return (
      <div>
        <button onClick={onReset}>复位</button>
        <button onClick={onStart}>启动</button>
      </div>
    );
  }
};
```

在这里用了一个 ES6 功能，叫做**解构赋值**（Destructuring Assignment）。因为 ControlButtons 是一个函数类型的组件，所以 props 以参数形式传递进来，props 中的属性包含 `activated` 这样的值，利用大括号，就可以完成对 props 的“解构”，把 props.activated 赋值给同名的变量 `activated`。

如果没有解构赋值，就只能用下面的代码，很明显，与使用了解构赋值的代码相比，真是啰嗦得让人难以忍受。

```
  const activated = props.activated;
  const onStart = props.onStart;
  const onPause = props.onPause;
  const onReset = props.onReset;
  const onSplit = props.onSplit;
```

我们可以更进一步，把解构赋值提到参数中，这样连 props 的对象都看不见，就像下面这样：

```
const ControlButtons = ({activated, onStart, onPause, onReset, onSplit}) => {

}
```

在 ControlButtons 的实现部分，我们根据 activated 的值返回不同的 JSX，当 activated 为 `true` 时，返回的是“计次”和“停止”；当 activated 为 `false` 时，返回的是“复位”和“启动”，对应分别使用了传入的 `on` 开头的函数类型 props。

可以看到，ControlButtons 除了显示内容和分配 props，没有做什么实质的工作，实质工作会在 StopWatch 中介绍。

接下来我们做 MajorClock，根据传入 props 的 milliseconds 来显示数字时钟一样的时分秒。在 MajorClock.js 文件中，我们这样定义 MajorClock：

```
const MajorClock = ({milliseconds=0}) => {
  return <h1>{ms2Time(milliseconds)}</h1>
};
```

在这里，我们不光直接解构了参数，而且使用了默认值。如果使用 MajorClock 时没有传入 `milliseconds` 这个 props，那么 `milliseconds` 的值就是 `0`。

因为把毫秒数转为 `HH:mm:ss:mmm` 这样的格式和 JSX 没什么关系，所以，我们不在组件中直接编写，而是放在 `ms2Time` 函数中，`ms2Time` 就是 `ms-to-Time`，代码如下：

```
import padStart from 'lodash/padStart';

const ms2Time = (milliseconds) => {
  let time = milliseconds;
  const ms = milliseconds % 1000;
  time = (milliseconds - ms) / 1000;
  const seconds = time % 60;
  time = (time - seconds) / 60;
  const minutes = time % 60;
  const hours = (time - minutes) / 60;

  const result = padStart(hours, 2, '0') + ":" + padStart(minutes, 2, '0') + ":" + padStart(seconds,
  return result;
}
```

通过逐步从 milliseconds 中抽取毫秒、秒、分、时的信息，最终拼出人类容易理解的时间。不过，为了和数字时钟显示一致，需要补齐，比如 2 秒 23 毫秒，显示成 `2:23` 可不好看，不够的位数要补上 0，显示成 `00:00:02:023`。这个补齐的工作和 React 无关，我们也不深究，直接使用 `lodash` 中的 `padStart` 实现。

为了在项目中使用 `lodash`，请先用 npm 完成对应的库安装。

最后是 SplitTimes 这个组件，在 `SplitTimes.js` 这个文件中，我们需要这样定义 SplitTimes：

```
import MajorClock from './MajorClock';

const SplitTimes = ({value=[]}) => {
  return value.map((v, k) => (
    <MajorClock key={k} milliseconds={v} />
  ));
};
```

因为根据毫秒数显示数字时钟的功能在 MajorClock 中已经做到了，所以我们直接导入 MajorClock 使用就好，这符合“重用代码”的原则。

值得一提的是，利用循环或者数组 map 而产生的动态数量的 JSX 元件，必须要有 `key` 属性。这个 `key` 属性帮助 React 搞清楚组件的顺序，如果不用 `key`，那 React 会在开发模式下在 console 上输出红色警告。

一般来说，key 不应该取数组的序号，因为 key 要唯一而且稳定，也即是每一次渲染过程中，key 都能唯一标识一个内容。对于 StopWatch 这个例子，倒是可以直接使用数组序号，因为计次时间的数组顺序不会改变，使用数组序号足够唯一标识内容。

StopWatch 状态管理

在实现了 MajorClock、ControlButtons 和 SplitTimes 之后，我们需要把这些子组件串起来，这就是 StopWatch。

StopWatch 是一个有状态的组件，所以，不能只用一个函数实现，而是做成一个继承自 `React.Component` 的类，如下：

```
class StopWatch extends React.Component {
  render() {
    return (
      <Fragment>
        <MajorClock />
        <ControlButtons />
        <SplitTimes />
      </Fragment>
    );
  }
}
```

对于一个 React 组件类，最少要有一个 `render` 函数实现，不过，上面的 `render` 只是一个大概的代码框架，引用了相关子组件，但是没有传入 props。

传入什么 props 呢？当然是 StopWatch 记录的 state。

StopWatch 的 state 需要有这些信息：

1. isStarted，是否开始计时；
2. startTime，计时开始时间，Date 对象；
3. currentTime，当前时间，也是 Date 对象；
4. splits，所有计次时间的数组，每个元素是一个毫秒数。

React 组件的 state 需要初始化，一般来说，初始化 state 是在构造函数中，代码如下：

```
  constructor() {
    super(...arguments);

    this.state = {
      isStarted: false,
      startTime: null,
      currentTime: null,
      splits: [],
    };
  }
```

如果定义构造函数 constructor，一定要记得通过 super 调用父类 `React.Component` 的构造函数，不然，功能会不正常。

React官方网站上的代码示例是这样调用super函数：

```
  constructor(props) {
    super(props); //目前可行，但有更好的方法
  }
```

在早期版本中，React.Component 的构造函数参数有两个，第一个是 `props`，第二个是 `context`，如果忽略掉 `context` 参数，那么这个组件的 context 功能就不能正常工作，不过，现在React的行为已经变了，第二个参数传递不传递都能让context正常工作，看起来React.Component 的构造函数只有第一个参数被用到，但是，没准未来还会增加新的参数呢，所以，以不变应万变的方法，就是使用扩展操作符（spread operator）来展开 arguments，这样不管 React 将来怎么变，这样的代码都正确。

```
  constructor() {
    super(...arguments); //永远正确!
  }
```

扩展操作符的作用，在 React 开发中会经常用到，在 JSX 中展开 props 的时候会用到。

属性初始化方法

不过，其实我们也可以完全避免编写 constructor 函数，而直接使用属性初始化（Property Initializer），也就是在 class 定义中直接初始化类的成员变量。

不用 constructor，可以这样初始化 state，效果是完全一样的：

```
class StopWatch extends React.Component {
  state = {
    isStarted: false,
    startTime: null,
    currentTime: null,
    splits: [],
  }
}
```

接下来，我们要考虑如何实现传递给 ControlButtons 的一系列函数。

首先，明确一点，尽量不要在 JSX 中写内联函数（inline function），比如这样写，是很不恰当的：

```
        <ControlButtons
          activated={this.state.isStarted}
          onStart={() => { /* TODO */}}
          onPause={() => { /* TODO */}}
          onReset={() => { /* TODO */}}
          onSplit={() => { /* TODO */}}
        />
```

当然，按照上面那种写法，也可以完成程序的功能，但是，会带来性能的代价。首先，每一次渲染这段 JSX，都会产生全新的函数对象，这是一种浪费；其次，因为每一次传给 ControlButtons 的都是新的 props，这样 ControlButtons 也无法通过 `shouldComponentUpdate` 对 props 的检查来避免重复渲染。

在本小册中后续章节中，为了代码简洁会使用内联函数，但只是为了示例方便，在实际工作中，在 JSX 中应用的函数 props 应该尽量使用类成员函数，不要用内联函数。

以最容易实现的 `onSplit` 为例，这个函数响应用户点击“计次”按钮的事件，代码如下：

```
  onSplit() {
    this.setState({
      splits: [...this.state.splits, this.state.currentTime - this.state.startTime]
    });
  }
```

在 onSplit 中，利用 `this.setState` 来修改组件的状态。不过问题来了，这个函数执行时，`this` 是什么呢？

很可惜，对于 ES6 的类成员函数，并不自动绑定 `this`，也就是说，onSplit 中的 `this`，可不保证就是当前组件对象。

至于 `render` 这些生命周期函数，里面访问的 `this` 就是当前组件本身，完全是因为这些函数是 React 调用的，React 对它们进行了特殊处理，对于其他普通的成员函数，特殊处理就要靠我们自己了。

通常的处理方法，就是在构造函数中对函数进行绑定，然后把新产生的函数覆盖原有的函数，就像这样：

```
  constructor() {
    super(...arguments);

    this.onSplit = this.onSplit.bind(this);
  }
```

如果可以使用 bind operator，也可以这样写：

```
    this.onSplit = ::this.onSplit;
```

只可惜 bind operator 并不是稳定的标准语法，而 create-react-app 又不想依赖不稳定的语法，所以在我们的应用中还不能这么写。

我们的 StopWatch 需要给 ControlButtons 传递四个函数类型的 props，分别是 `onStart`、`onPause`、`onReset` 和 `onSplit`，对每一个函数都在构造函数里加一个 bind，也是够累的，还容易出错，所以，我们肯定会寻求更好的方法。

更好的方法依然是使用属性初始化，就和初始化 state 一样，利用等号直接初始化 onSplit，代码如下：

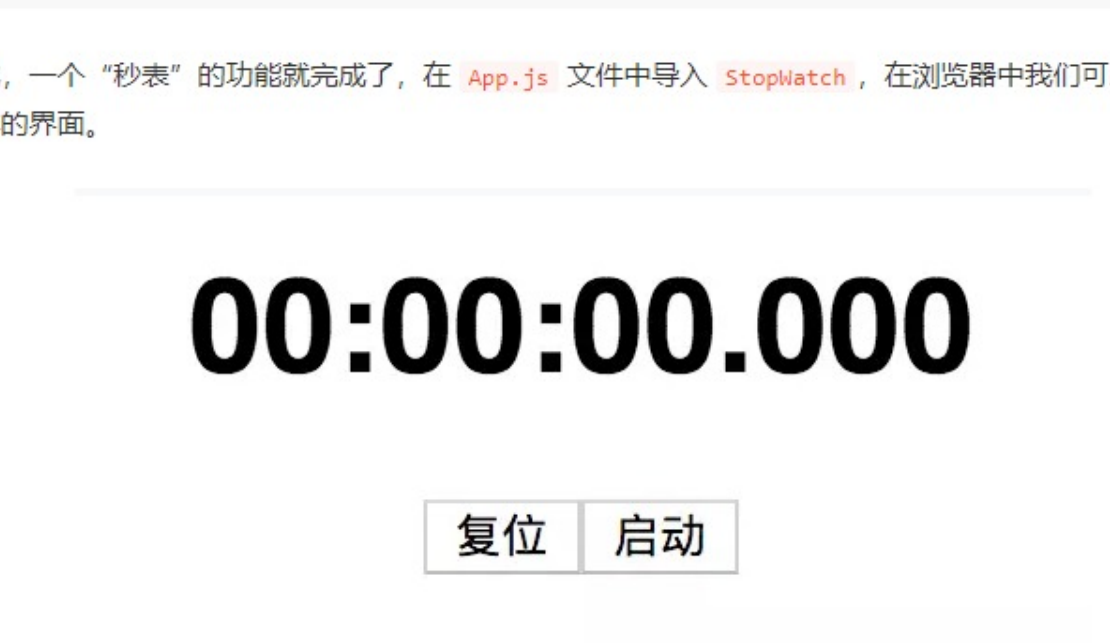
```
  onSplit = () => {
    this.setState({
      splits: [...this.state.splits, this.state.currentTime - this.state.startTime]
    });
  }
```

像上面这样写，就不需要 constructor，函数体内的 `this` 绝对就是当前组件对象。

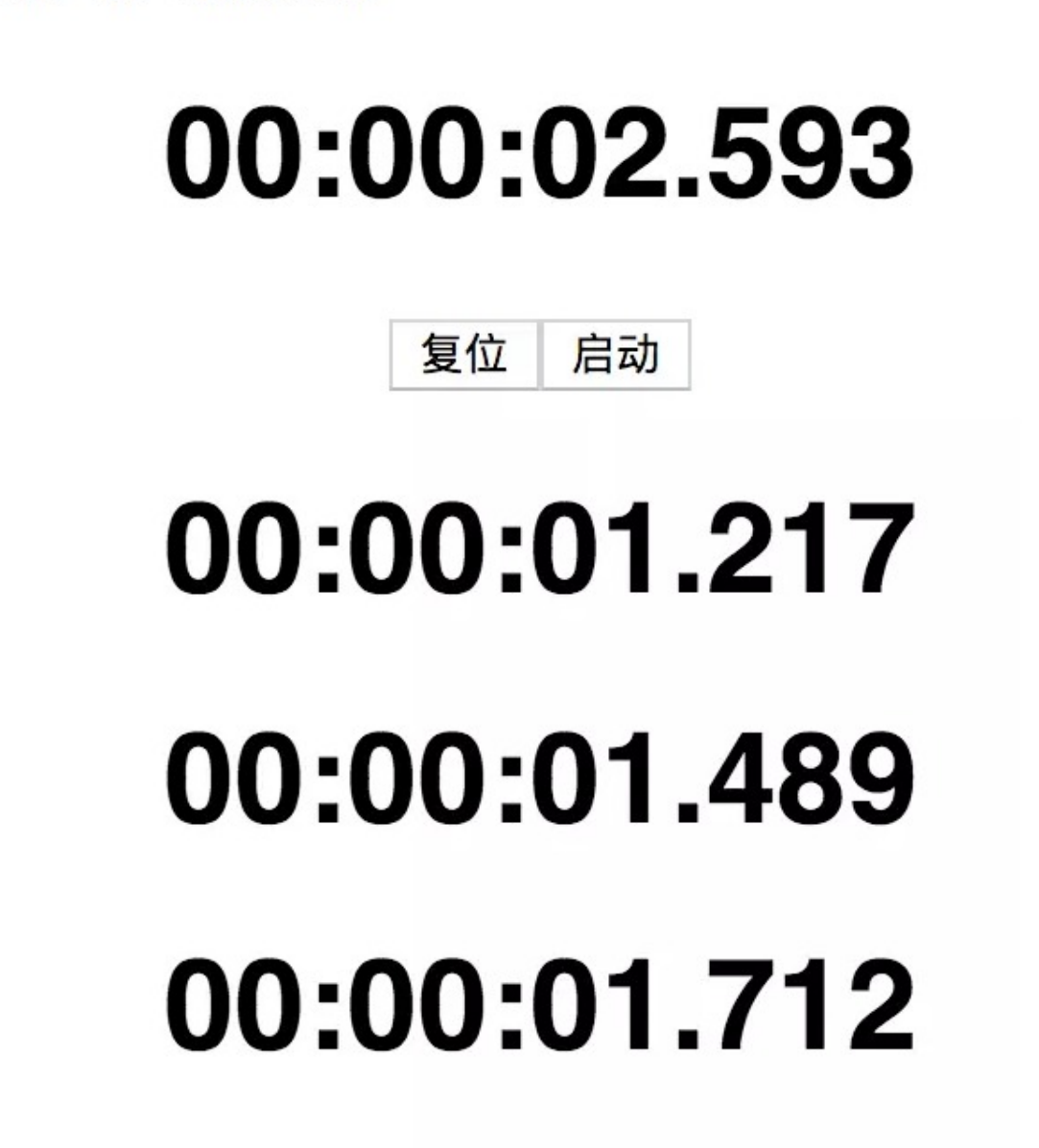
用同样的方法，我们一起实现其他函数成员。

```
  onStart = () => {
    this.setState({
      isStarted: true,
      startTime: new Date(),
      currentTime: new Date(),
    });

    this.intervalHandle = setInterval(() => {
      this.setState({currentTime: new Date()});
    }, 1000 / 60);
  }

  onPause = () => {
    clearInterval(this.intervalHandle);
    this.setState({
      isStarted: false,
    });
  }

  onReset = () => {
    this.setState({
      startTime: null,
      currentTime: null,
      splits: [],
    });
  }
```

至此，一个“秒表”的功能就完成了，在 `App.js` 文件中导入 `StopWatch`，在浏览器中我们可以看到这样的界面。



点击“启动”按钮，可以看见数字时钟开始运转；点击“计次”按钮，在按钮下方可以看到点击瞬间的时间；点击“停止”，时钟停止运转。



当然，现在这个“秒表”的界面还非常粗糙，和 iPhone 上的秒表应用差远了。但是，它该有的功能一个都不缺，只有功能完整而且正确，样式才有意义。

在后面的章节中，我们会用 React 的方式来来美化界面。

小结

在这一小节中，我们完成 StopWatch 秒表组件的实现，在这个过程中，读者应该学习到这些技巧：

1. 尽量每个组件都有自己专属的源代码文件；
2. 用**解构赋值**（destructuring assignment）的方法获取参数 props 的每个属性值；
3. 利用**属性初始化**（property initializer）来定义 state 和成员函数。

留言

评论将在后台进行审核，审核通过后对所有人可见

十二不死灰 前端 @ 上海汉得研发
点击停止后再继续点启动会重新从0开始.. 这个功能bug。 加个变量处理下吧 大佬。
▲ 0 收起评论 1月前

程墨 Hulu
如果能够复现这个问题，可以提一个PR到 https://github.com/mocheng/react-practice
1月前

评论审核通过后显示 评论

东东东寅
虽然使用成员函数的方法来代替bind更加简洁，但有一个副作用，比如有一个todolist的功能，分别实现添加和删除的功能，这时候如果我删除需要传递索引，正常情况下，是我点击具体要删除的项目才打印索引，但实际情况是，当我增加的时候，会改变state，通过this.setState从而render函数重新渲染，这时候诸如this.xx(index)会执行
▲ 0 收起评论 1月前

xiari 高级前端工程师 @ 拼多多
onClick = () => this.xx(index)
1月前

sanseo 前端工程师
没太听明白，能用代码演示下吗
24天前

晨风明悟 前端 @ Now.Now.Now
回复 sanseo：render 里面的 onClick 事件绑定的成员函数包裹到另一个函数中，这样就不会立即执行了
19天前

sanseo 前端工程师
回复 晨风明悟：这样每次render都要重新生成匿名函数，是对资源的浪费吧
19天前

晨风明悟 前端 @ Now.Now.Now
回复 sanseo：为了优化性能，我们在类中定义箭头函数也是可以的，例如：onClick = () => this.xx(index) 在类中定义如下：

class Example extends React.Component {
  handleClick = index => {
}
}
17天前

评论审核通过后显示 评论

东东东寅
回
▲ 0 评论 1月前

东东东寅
没有讲为啥避免在构造函数体声明类的初始成员变量
▲ 0 收起评论 1月前

程墨 Hulu
你说的是成员变量还是state？很多React类组件根本不需要state之外的成员变量。
1月前

东东东寅
回复 程墨：哈哈，就是构造函数里初始化状态和不在构造函数里初始化状态有没有性能上的差别
1月前

程墨 Hulu
回复 东东东寅：用property intializer的方法初始化成员，最后一样被babel进到构造时初始化，没有任何性能差别。
1月前

东东东寅
回复 程墨：没啦，其实如果是需要响应视图的更新，应该放在构造函数里初始化状态，而如果当成是作为一些常量等直接作为成员变量
1月前

程墨 Hulu
回复 东东东寅：抱歉，我没大明白的意思，你能用code说明一下吗？
1月前

1 2 下一页

评论审核通过后显示 评论

公仔qpee 前端
> 所以，从达到“代码整洁”的目的来说，应该每个组件都有一个独立的文件，然后这个文件用 export default 的方式导出单个组件。

为啥 react 的 component 喜欢用 default 的导出形式，而不是 named export，我看 Angular 的推荐一般都是习惯使用 named export.

我觉得 default 的方式很容易就被人把导出名改写了，而且 named export 的话，很多 ide 工具就能很好地分析并且自动引入（当然 default... 展开全部
▲ 0 收起评论 1月前

程墨 Hulu
如果你需要导出多个元素，除了named export别无他法；如果你的文件模块就只有一个东西要导出，你用named export会显得很啰嗦而且很怪，import {StopWatch} from './StopWatch.js'，肯定没有import StopWatch from './StopWatch.js' 看着清楚。
1月前

我爱吃南瓜 菜鸡前端菜鸡Python
回复 程墨：可能是因为写NG习惯了的原因，个人倒是觉得named export更清楚~😅
1月前

程墨 Hulu
回复 我爱吃南瓜：不同框架有不同的convention，也倒不必强求一致
1月前

公仔qpee 前端
回复 程墨：所以其实也没有什么官方推荐或者理由，只不过是 airbnb 非要搞一个 default export 的 lint 规则，才导致社区把这个当做最佳实践了
1月前

评论审核通过后显示 评论

snowLu 前端小洋葱 @ lg
如果没有解构赋值，就只能用上面的代码 这里有个笔误，应该是下面的代码
▲ 0 收起评论 1月前

程墨 Hulu
改过来了，谢谢指正
1月前

评论审核通过后显示 评论

上上忍
可以提供下源码链接嘛
▲ 0 收起评论 1月前

程墨 Hulu
在这 https://github.com/mocheng/react-practice
1月前

评论审核通过后显示 评论

Artyhacker 前端 @ 北京某小国企
写了一年多的react项目了,之前几节觉得也太基础了,看到这里发现以前还真没用过属性初始化,光这一点就值了,以后可以节省好多时间,期待后面的章节
▲ 0 收起评论 1月前

程墨 Hulu
感谢支持。
1月前

评论审核通过后显示 评论

KK_FE FE @ Shopee
<SplitTimes /> 参数传的是什么呢
▲ 0 收起评论 1月前

sizeof不想说话
values={splits}
1月前

评论审核通过后显示 评论

Mr.HoTwo
00:00:02:023 最后多了一个 0
▲ 1 收起评论 1月前

一個抗爭的少年
1 second === 1000 ms
1月前

评论审核通过后显示 评论

jscheng
onStart 方法中需要设置 currentTime: new Date() 否则第一次获取的milliseconds会有问题
▲ 6 评论 1月前

Clementines
onStart()中少设置了currentTime, 不设置的话ms2Time会出一个负4万多的时间。
▲ 2 评论 1月前

jscheng
"这个 key 属性帮助 React 搞清楚元件的顺序" 应该是 "这个 key 属性帮助 React 搞清楚组件的顺序"
▲ 3 评论 1月前