组件实践（1）：如何定义清晰可维护的接口

在第一节，我们已经知道，React 世界由组件构成，所以，如何设计组件的接口就成了组件设计最重要的事情。

设计原则

React 的组件其实就就是软件设计中的模块，所以其设计原则也遵从通用的组件设计原则，简单说来，就是要减少组件之间的耦合性（Coupling），让组件的界面简单，这样才能让整体系统易于理解、易于维护。

更具体一点，在设计 React 组件时，要注意以下原则：

1. 保持接口小，props 数量要少；
2. 根据数据边界来划分组件，充分利用组合（composition）；
3. 把 state 往上层组件提取，让下层组件只需要实现为纯函数。

说太多理论没意思，让我们用一个实际例子来解读这些原则，我们选择“秒表”这个程序来讲。

秒表曾经是一个体育专业人士才配备的工具，不过，现在大家的手机上肯定都有这样的应用，比如，在 iPhone 上的“时钟”里就如下图所示的秒表。

按下右侧“启动”按钮，这个按钮就会变成“停止”，同时上面的数字时钟开始计时；按下“停止”按钮，数字时钟停止计时。请注意左侧还有一个按钮，初始状态显示“复位”，点击该按钮会清空时钟；开始计时之后，这个左侧按钮会变成“计次”，按一下“计次”，秒表底部就会增加一列时间，记录下按下“计次”这一瞬间的时刻。

秒表可以用来测量运动员的训练时间，比如运动员起跑的时候按“启动”，每跑一圈回到起点，按一下“计次”。这样跑十圈下来，可以知道每一圈都分别花了多少时间，运动员和教练可以做对应调整。

秒表是一个很实用的应用，复杂度也适当，这本小册中会以秒表为例展示 React 应用的开发，不过，我们无需急着写代码，首先来规划一下秒表的 React 组件接口如何设计。

组件的划分

我们会做一个 React 组件来渲染整个秒表，这个组件可以叫做 StopWatch，目前看来这个组件不需要从外部获得什么输入，本着“props 数量要少”的原则，我们也不需要费心未来会用上什么 props，目前就当 StopWatch 不支持 props 好了。

此外，这个组件需要记录当前计时，还要记录每一次按下“计次”的时间，所以需要维持状态（state），可以肯定，StopWatch 是一个有状态的组件，不能只是一个纯函数，而是一个继承自 Component 的类。

```
class StopWatch extends React.Component {
  render() {
    //TODO: 返回所有JSX
  }
}
```

任何一个复杂组件都是从简单组件开始的，一开始我们在 render 函数里写的代码不多，但是随着逻辑的复杂，JSX 代码越来越多，于是，就需要拆分函数中的内容。

在 React 中，有一个误区，就是把 render 中的代码分拆到多个 renderXXXX 函数中去，比如下面这样：

```
class StopWatch extends React.Component {
  render() {
    const majorClock = this.renderMajorClock();
    const controlButtons = this.renderControlButtons();
    const splitTimes = this.renderSplitTimes();

    return (
      <div>
        {majorClock}
        {controlButtons}
        {splitTimes}
      </div>
    );
  }

  renderMajorClock() {
    //TODO: 返回数字时钟的JSX
  }

  renderControlButtons() {
    //TODO: 返回两个按钮的JSX
  }

  renderSplitTimes() {
    //TODO: 返回所有计次时间的JSX
  }
}
```

用上面的方法组织代码，当然比写一个巨大的 render 函数要强，但是，实现这么多 renderXXXX 函数并不是一个明智之举，因为这些 renderXXXX 函数访问的是同样的 props 和 state，这样代码依然耦合在了一起。更好的方法，是把这些 renderXXXX 重构成各自独立的 React 组件，像下面这样：

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

const MajorClock = (props) => {
  //TODO: 返回数字时钟的JSX
};

const ControlButtons = (props) => {
  //TODO: 返回两个按钮的JSX
};

const SplitTimes = (props) => {
  //TODO: 返回所有计次时间的JSX
}
```

我们创造了 `MajorClock`、`ControlButtons` 和 `SplitTimes` 这三个组件，目前，我们并不知道它们是否应该有自己的 state，但是从简单开始，首先假设它们没有自己的 state，定义为函数形式的无状态组件。

按照数据边界来分割组件

现在，我们来检视一下，这样的组件划分，是否符合“按照数据边界划分”的原则。

渲染 `MajorClock`，需要的是当前展示的时间，在点击“启动”按钮之后，这个时间是不断增长的。

渲染 `ControlButtons`，两个按钮显示什么内容，完全由当前是否是“启动”的激活状态决定。此外，Buttons 是秒表中唯一有用户输入的组件，对于按钮的按键会改变秒表的状态。

最后，计次时间 `SplitTimes`，需要渲染多个时间，可以想象，需要有一个数组来记录所有计次时间。

总结一下所有需要的数据和对应标识符，以及影响的组件：

数据	标识符	影响的组件
当前时间	timeElapsed	MajorClock
是否启动	activated	MajorClock, ControlButtons
计次时间	splits	SplitTimes

从上面的表格可以看出，每个数据影响的组件都不多，唯一影响两个组件的数据是 activated，这个 activated 基本上就是一个布尔值，数据量很小，影响两个组件问题也不大。

这样的组件划分是符合以数据为边界原则的，很好。

state 的位置

接下来，我们要确定 state 的存储位置。

当秒表处于启动状态，MajorClock 会不断更新时间，似乎让 MajorClock 来存储时间相关的 state 很合理，但是仔细考虑一下，就会发现这样并不合适。

设想一下，MajorClock 包含一个 state 记录时间，因为 state 是组件的内部状态，只能通过组件自己来更新，所以要 MajorClock 用一个 `setTimeout` 或者 `setInterval` 来持续更新这个 state，可是，另一个组件 ControlButtons 将会决定什么时候暂停 MajorClock 的 state 更新，而且，当用户按下“计次“按钮的时候，MajorClock 还需要一个方法把当前的时间通知给 SplitTimes 组件。

这样一个数据传递过程，想一想都觉得很麻烦，明显不合适。

这时候就需要考虑这样的原则，**尽量把数据状态往上层组件提取**。在秒表这个应用中，上层组件就是 StopWatch，如果我们让 StopWatch 来存储时间状态，那一切就会简单很多。

StopWatch 中利用 setTimeout 或者 setInterval 来更新 state，每一次更新会引发一次重新渲染，在重新渲染的时候，直接把当前时间值传递给 MajorClock 就完事了。

至于 ControlButtons 对状态的控制，让 StopWatch 传递函数类型 props 给 ControlButtons，当特定按钮时间点击的时候回调这些函数，StopWatch 就知道何时停止更新或者启动 setTimeout 或者 setInterval，因为这一切逻辑都封装在 StopWatch 中，非常直观自然。

对了，还有 SplitTimes，它需要一个数组记录所有计次时间，这些数据也很自然应该放在 StopWatch 中维护，然后通过 props 传递给 SplitTimes，这样 SplitTimes 只单纯做渲染就足够。

组件 props 的设计

当我们确定了组件结构和 state 之后，剩下来要做的就是 props 的设计了。

先来看 MajorClock，因为它依赖的数据只有当前时间，所以只需要一个 props。

```
const MajorClock = ({milliseconds}) => {
  //TODO: 返回数字时钟的JSX
};

MajorClock.propTypes = {
  milliseconds: PropTypes.number.isRequired
};
```

和函数参数的命名一样，props的命名一定力求简洁而且清晰。对于MajorClock，如果把这个props命名为 `time`，很容易引起歧义，这个 time 的单位是什么？是毫秒？还是秒？还是一个 Date 对象？

所以，我们明确传入的 props 是一个代表毫秒的数字，所以命名为 `milliseconds`，如果你的开发团队可以接受，也可以简写为 `ms`。

然后是 ControlButtons，这个组件需要根据当前是否是“启动”状态显示不同的按钮，所以需要一个 props 来表示是否“启动”，我们把它命名为 `activated`。

此外，StopWatch 还需要传递回调函数给 ControlButtons，所以还需要支持函数类型的 props，分别代表 ControlButtons 可以做的几个动作：

- 启动（start）
- 停止（pause）
- 计次（split）
- 复位（reset）

一般来说，为了让开发者能够一眼认出回调函数类型的 props，这类 props 最好有一个统一的前缀，比如 `on` 或者 `handle`，我个人倾向于用 on，所以，ControlButtons 的接口就是下面这样：

```
const ControlButtons = (props) => {
  //TODO: 返回两个按钮的JSX
};

ControlButtons.propTypes = {
  activated: PropTypes.bool,
  onStart: PropTypes.func.isRquired,
  onPause: PropTypes.func.isRquired,
  onSplit: PropTypes.func.isRquired,
  onReset: PropTypes.func.isRquired,
};
```

最后是 SplitTimes，很简单，它需要接受一个数组类型的 props。

你知道吗？PropTypes 也可以支持数组类型的定义哦。

```
const SplitTimes = (props) => {
  //TODO: 返回所有计次时间的JSX
}

SplitTimes.propTypes = {
  splits: PropTypes.arrayOf(PropTypes.number)
};
```

至此，完成了秒表的组件接口设计，我们还完全没有涉及组件内部的具体代码编写，这在后面的小节中会详细讲解，不过，一个好的设计就是要在写代码之前就应用被证明最佳的原则，这样写代码的过程就会少走弯路。

小结

在这一节中，我们通过设计秒表，展示了组件接口设计的三个原则：

1. 保持接口小，props 数量要少
2. 根据数据边界来划分组件，利用组合（composition）
3. 把 state 尽量往上层组件提取

同时，我们也接触了这样一些最佳实践：

1. 避免 renderXXXX 函数
2. 给回调函数类型的 props 加统一前缀
3. 使用 propTypes 来定义组件的 props

留言

评论将在后台进行审核，审核通过后对所有人可见

王巍达 前端 @ JD
有一个问题想讨论一下，例子本身是一个很简单的组件，所以如此划分是没有什么异议的，但是真实的后台项目中一个页面往往比较庞大，在项目中尝试保持页面中的子组件纯时，由于要把各个子组件的state与逻辑提到最外层的页面组件，会导致最外层的组件代码膨胀的很快，子组件可能纯了，只是render一个jsx代码量比较少，但是外层组件很可能很快就达到几百行了，不知道这点有没有什么好的建议
▲ 0 收起评论 17天前

> **程墨** Hulu
> 你的具体情况我不了解，但我猜想并不是说所有state要放在“一个组件”里，根据逻辑依然可以一组组件的state放在一个组件，其他组的组件放在另一个组件里。
> 14天前
>
> **Hahet**
> 哈哈 我也有有遇到这样的问题， 就是如果让ui组件尽量写成无状态组件（把控制权扔给使用者）的话， 那使用的时候就会麻烦一点（多写点代码）。 如果在ui组件里面给一些内部状态可能会简化使用， 但是内部代码会变复杂， 不好维护。如何取舍
> 9天前
>
> 评论审核通过后显示 评论

于先森 前端工程师 @ 小年糕
请问大佬平常写组件是只通过ui分组件 还是只通过数据分 还是先看ui再结合数据 感谢大佬回答。
▲ 0 收起评论 27天前

> **于先森** 前端工程师 @ 小年糕
> 不好意思语言组织有误 我的意思是先根据功能数据之后看UI分 还是先看UI再结合数据分
> 27天前
>
> **程墨** Hulu
> 回复 于先森：虽然做事不应该太绝对，但是我一般是按照数据来分组件，不过，UI上一个组件本来未必对应一个代码层次的组件。
> 26天前
>
> 评论审核通过后显示 评论

yadong 前端开发 @ 知乎
请教个问题，是把单个按钮（启动对应一个）抽成一个组件好呢，还是像作者一样将整个控制部分抽成一个组件好？
▲ 0 收起评论 29天前

> **于先森** 前端工程师 @ 小年糕
> 我的想法是在整个控制部分中 再添加一个单独的按钮组件
> 27天前 回复
>
> 评论审核通过后显示 评论

起风了8866 前端工程师
我试着回答一下类组件和函数式组件的区别和用法。函数式组件不能拥有自己的状态(state)，生命周期只有render。类组件可以有自己的状态(也可以不声明)。使用时优先考虑函数式组件
▲ 1 评论 1月前

jjunxi
第一步叫组件的划分，第二步叫根据数据边界分割组件。
第一步很好理解，基本是按照功能或者“块”来划分组件。
第二步的作用是把需要的数据关联到对应的组件上么？
“根据数据边界来划分组件”，这句话很抽象。通俗的说法是啥样的？
▲ 0 收起评论 1月前

> **程墨** Hulu
> 下面有一个评论也回答了这个问题，通俗说来，就是把相近的数据封装在同一个组件里。
> 1月前
>
> **于先森** 前端工程师 @ 小年糕
> 我觉得通俗一点来讲就是设计组件时最大化减少组件与组件之间数据的依赖性
> 27天前 回复
>
> 评论审核通过后显示 评论

明易 前端开发工程师
根据数据边界来划分组件这句话不太懂 能具体讲讲么
▲ 5 收起评论 1月前

> **程墨** Hulu
> 比如说，你的应用可以有两种划分数据方法A+B和C+D，A和B之间的纠结很多，C和D之间的纠结不多，那你应该选择为A、B各自构建一个组件？还是为C、D构建各自构建一个组件？当然是后者，因为这样两个组件之间的交互纠结最少，复杂性也最低。
> 1月前
>
> 评论审核通过后显示 评论

hellochen
无状态组件和类组件 有什么功能上的差异吗
▲ 0 收起评论 1月前

> **程墨** Hulu
> 无状态组件和类组件并不是对立的概念，一个类组件如果没有自己的state，一样是无状态组件，类组件和函数形式组件才是对立的概念。
> 1月前
>
> **起风了8866** 前端工程师
> 我试着回答一下类组件和函数式组件的区别和用法。函数式组件不能拥有自己的状态(state)，生命周期只有render。类组件可以有自己的状态(也可以不声明)。使用时优先考虑函数式组件
> 1月前
>
> 评论审核通过后显示 评论

仪式感
划分组件那块，既然数据来源都是一致，访问的props也一样，抽出无状态组件和写里面有啥区别？
▲ 0 收起评论 1月前

> **程墨** Hulu
> 我……有点无言以对，我可以请你再看一遍这部分内容吗？如果你看完依然有这个问题，我乐意再来回答这个问题。
> 1月前
>
> **Jexxie**
> "任何一个复杂组件都是从简单组件开始的，一开始我们在 render 函数里写的代码不多，但是随着逻辑的复杂，JSX 代码越来越多，于是，就需要拆分函数中的内容。"
> 1月前 回复
>
> **仪式感**
> 回复 程墨：可能问的不太清楚，原文说到“因为这些 renderXXXX 函数访问的是同样的 props 和 state，这样代码依然耦合在了一起”，我想问的是抽出来之后就低耦合了嘛？
> 1月前
>
> **zhangyanling77** 前端开发 @ 成都
> 回复 仪式感：作者的意思是你抽出来的render函数还是得放在当前这个组件内一起维护对不对，并且他们接受相同的state和props，那么抽离出去成为组件是不是更独立，也降低了耦合度。你没理解到作者表达的是render臃肿了不好，即便是抽象成函数。但是组件就不一样了
> 1月前
>
> **十二不死灰** 前端 @ 上海汉得研发
> 回复 仪式感：实际开发中，如果不抽成组件，界面简单还行，一复杂，一个文件里多个renderXXX函数 逻辑再多一些 代码上千行简简单单。不利于维护和优化...特别改他人代码的时候。抽出的无状态组件另写一个文件，用的时候导入（大佬写例子 懒得拆出去）。
> 1月前
>
> 评论审核通过后显示 评论

清秋 SDE @ CMBC
划分组件这块，看例子的思路还是先通过 UI 不同部分来划分组件，然后再用“根据数据边界划分组件”这个原则来确认组件划分是否合理。
▲ 0 评论 1月前

袁某某 喊666的咸鱼
当处秒表于启动状态 -> 当秒表处于启动状态
▲ 0 收起评论 1月前

> **程墨** Hulu
> 改过来了，谢谢指正。
> 1月前
>
> **胡子不想说话**
> 五笔输入法的锅，哈哈
> 1月前
>
> 评论审核通过后显示 评论

合拾 浏览器鼓励师 @ BBD
先来看 MajorClock，因为它依赖的数据只有当前时间，所以只需要一个 prop。（只需要一个props）
▲ 0 评论 2月前