2-4 React 中的状态管理与状态管理库

状态 state

React UI = fn(state)

fn(x) = x+1

state 是变量，一般情况下，state 改变之后，组件会更新。

当你有一个值，这个值变化之后，页面需要随之更新，那么这个变量就可以定义成 **state，比如 useState、useReducer、第三方状态管理库**。官网链接：state：组件的记忆

状态管理

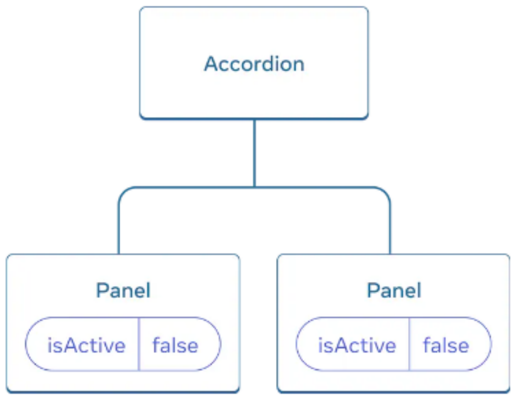
1. 组件内部状态

在 React 函数组件中，我们可以使用 **useState** 和 **useReducer** 定义组件内部 state，这个 state 其实是存储在 VDOM 上的，就是 fiber 节点上。每一个组件都有一个与之对应的 js 对象，即 fiber 节点。

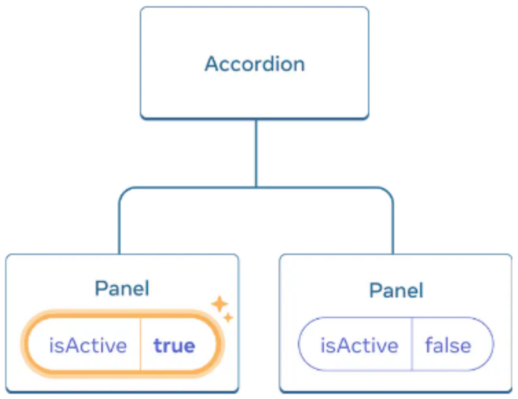
2. 在组件间共享状态

状态提升

有时候，我们希望多个组件的 state 始终同步更改。要实现这一点，我们可以把相关 state 从这些个组件上移除，然后 把 state 放到它们的公共父级，再通过 props 将 state 传递给这些个组件。这被称为“**状态提升**”。



由于一开始所有 `Panel` 内部的 `isActive` 都是 `false`，因此它们的内容都不会显示。



此时，点击任意一个 `Panel` 组件中的按钮都只会更新当前 `Panel` 组件内的 `isActive` 值

在 React 组件外部管理状态

就是 React 状态管理库。

React 状态管理库生态比较繁荣，比如 Redux、MobX、Zustand、Recoil、xState、Valtio 等。AntD4/5 Form 组件也是在外部定义的 state。

以 Redux 为例：

```
export default function createStore(reducer, enhancer) {
  if (enhancer) {
    return enhancer(createStore)(reducer);
  }
  let currentState;
  let listenerIdCounter = 0;
  let currentListeners = new Map(); //[];

  function getState() {
    return currentState;
  }

  function dispatch(action) {
    currentState = reducer(currentState, action);

    currentListeners.forEach((listener) => listener());

    return action;
  }

  function subscribe(listener) {
    const listenerId = listenerIdCounter++;
    currentListeners.set(listenerId, listener);

    return () => {
      currentListeners.delete(listenerId);
    };
  }

  dispatch({type: `@@redux/INIT${randomString()}`});

  return {
    getState,
    dispatch,
    subscribe,
  };
}

const randomString = () =>
```

```
  Math.random().toString(36).substring(7).split("").join(".");
```

再来看一眼 AntD4/AntD5 Form 的简写版源码:

```JavaScript
import { useRef } from "react";

class FormStore {
  constructor() {
    this.store = {}; // 状态值： name: value
    this.fieldEntities = [];

    this.callbacks = {};
  }

  setCallbacks = (callbacks) => {
    this.callbacks = { ...this.callbacks, ...callbacks };
  };

  // 注册实例(forceUpdate)
  // 注册与取消注册
  // 订阅与取消订阅
  registerFieldEntities = (entity) => {
    this.fieldEntities.push(entity);

    return () => {
      this.fieldEntities = this.fieldEntities.filter((item) => item !=
      delete this.store[entity.props.name];
    };
  };

  // get
  getFieldsValue = () => {
    return { ...this.store };
  };

  getFieldValue = (name) => {
    return this.store[name];
  };
  // set
  // name, password
  // name value
  setFieldsValue = (newStore) => {
    // 1. update store
    this.store = {
      ...this.store,
      ...newStore,
    };
    // 2. update Field
    this.fieldEntities.forEach((entity) => {
      Object.keys(newStore).forEach((k) => {
        if (k === entity.props.name) {
          entity.onStoreChange();
        }
      });
    });
  };

  validate = () => {
    let err = [];
    // 简版校验

    this.fieldEntities.forEach((entity) => {
      const { name, rules } = entity.props;

      const value = this.getFieldValue(name);
      let rule = rules[0];

      if (rule && rule.required && (value === undefined || value === "
        err.push({ [name]: rule.message, value });
      }
    });

    return err;
  };

  submit = () => {
    console.log("submit");

    let err = this.validate();
    // 提交
    const { onFinish, onFinishFailed } = this.callbacks;

    if (err.length === 0) {
      // 校验通过,
      onFinish(this.getFieldsValue());
    } else {
      // 校验不通过,
      onFinishFailed(err, this.getFieldsValue());
    }
  };

  getForm = () => {
    return {
      getFieldsValue: this.getFieldsValue,
      getFieldValue: this.getFieldValue,
      setFieldsValue: this.setFieldsValue,
      registerFieldEntities: this.registerFieldEntities,
      submit: this.submit,
      setCallbacks: this.setCallbacks,
    };
  };
}

export default function useForm(form) {
  // 存值，在组件卸载之前指向的都是同一个值
  const formRef = useRef();

  if (!formRef.current) {
    if (form) {
      formRef.current = form;
    } else {
      const formStore = new FormStore();
      formRef.current = formStore.getForm();
    }
  }
  return [formRef.current];
}
```