# React：我们的用法习惯可能是错误的(不优雅)

在我们React的日常开发中一些常用的写法，看似运行的很好，实际可能并不优雅。学习React并不是如何如何使用它，而是如何写出优雅，干净的代码。下面举一些例子，总结了一些React开发中不好的写法及相应更好的写法。(**仅代表个人观点**)

## 过多的声明state

### 问题

一个组件中声明了过多的state，过多的setState方法。例如下面的这样：

ini复制代码

```
import { useState } from "react";

export default function MoreState() {
  const [username, setUsername] = useState("");
  const [age, setAge] = useState("");
  const [gender, setGender] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [address, setAddress] = useState("");
  const [city, setCity] = useState("");

  const onSubmit = () => {
    // ...
  };

  return (
    <form onSubmit={onSubmit}>
      <input
        type="text"
        name="username"
        placeholder="username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <br />
      <input
        type="text"
        name="age"
        placeholder="age"
        value={age}
        onChange={(e) => setAge(e.target.value)}
      />
      <br />
      <input
        type="text"
```

```
          name="gender"
          placeholder="gender"
          value={gender}
          onChange={(e) => setGender(e.target.value)}
        />
        <br />
        <input
          type="text"
          name="email"
          placeholder="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        <br />
        <input
          type="text"
          name="password"
          placeholder="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <br />
        <input
          type="text"
          name="address"
          placeholder="address"
          value={address}
          onChange={(e) => setAddress(e.target.value)}
        />
        <br />
        <input
          type="text"
          name="city"
          placeholder="city"
          value={city}
          onChange={(e) => setCity(e.target.value)}
        />
        <br />
        <button type="submit">提交</button>
      </form>
    );
  }
```

实际上这样并不好维护，接手项目的人都疯了😁。还有这样的：

```
export default function ShipmentPage() {
  const router = useRouter()
  const { id } = router.query
  const { t } = useTranslation("common")
  const [activeStep, setActiveStep] = useState<string>(t("Select"))
  const [width, setWidth] = useState("0%")
  const [order, setOrder] = useState<Order | null>(null)
  const [shipment, setShipment] = useState<Shipment | null>(null)
  const [isModalOpen, setIsModalOpen] = useState(false)
  const [isModalOpen2, setIsModalOpen2] = useState(false)
  const [isModalOpen3, setIsModalOpen3] = useState(false)
  const [isModalOpen4, setIsModalOpen4] = useState(false)
  const [isModalOpen5, setIsModalOpen5] = useState(false)
  const [isModalOpen6, setIsModalOpen6] = useState(false)
  const [isModalOpen7, setIsModalOpen7] = useState(false)
  const [isModalOpen8, setIsModalOpen8] = useState(false)
  const [isModalOpen9, setIsModalOpen9] = useState(false)
  const [isModalOpen10, setIsModalOpen10] = useState(false)
  const [isModalOpen11, setIsModalOpen11] = useState(false)
  const [isModalOpen12, setIsModalOpen12] = useState(false)
  const [isModalOpen13, setIsModalOpen13] = useState(false)
  const [isModalOpen14, setIsModalOpen14] = useState(false)
  const [isModalOpen15, setIsModalOpen15] = useState(false)
  const [isModalOpen16, setIsModalOpen16] = useState(false)
  const [isModalOpen17, setIsModalOpen17] = useState(false)
  const [isModalOpen18, setIsModalOpen18] = useState(false)
  const [isModalOpen19, setIsModalOpen19] = useState(false)
  const [isModalOpen20, setIsModalOpen20] = useState(false)
  const [isModalOpen21, setIsModalOpen21] = useState(false)
  const [isModalOpen22, setIsModalOpen22] = useState(false)
  const [isModalOpen23, setIsModalOpen23] = useState(false)
  const [isModalOpen24, setIsModalOpen24] = useState(false)
}
```



## 解决方法

把能合并的state，合并成一个对象表示。当然也可以使用useReducer。当属性中出现嵌套结构时，例如属性中有对象和数组时，使用useReducer更好一些。

ini复制代码

```
import { useState } from "react";

export default function MoreState() {
  const [userInfo, setUserInfo] = useState({
    username: "",
    age: "",
    gender: "",
    email: "",
    password: "",
    address: "",
    city: ""
  });

  const onChange = (e) => {
    setUserInfo((pre) => ({ ...pre, [e.target.name]: e.target.value }));
  };

  const onSubmit = (e) => {
    e.preventDefault();
    console.log(111, userInfo);
  };

  return (
    <form onSubmit={onSubmit}>
      <input
        type="text"
        name="username"
        placeholder="username"
        onChange={onChange}
      />
      <br />
      <input type="text" name="age" placeholder="age" onChange={onChange} />
      <br />
      <input
        type="text"
        name="gender"
        placeholder="gender"
        onChange={onChange}
      />
      <br />
      <input type="text" name="email" placeholder="email" onChange={onChange
      <br />
      <input
        type="text"
        name="password"
        placeholder="password"
        onChange={onChange}
      />
      <br />
      <input
        type="text"
        name="address"
        placeholder="address"
        onChange={onChange}
      />
      <br />
      <input type="text" name="city" placeholder="city" onChange={onChange}
```

```
        <br />
        <button type="submit">提交</button>
      </form>
    );
  }
```

## 不必要的state

### 问题

我们在开发React表单时，通常会使用state来记录表单的值，例如：

javascript复制代码

```
import { useState } from "react";

export default function NoState() {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  const onSubmit = (e) => {
    e.preventDefault();
    console.log("需要提交的数据", username, password);
  };

  console.log("组件重新渲染了");

  return (
    <form onSubmit={onSubmit}>
      <label htmlFor="name">名字</label>
      <input
        type="text"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <br />
      <label htmlFor="name">密码</label>
      <input
        type="text"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <br />
      <button type="submit">提交</button>
    </form>
  );
}
```

上面的代码看似并没有什么问题，但是我们只是在提交的时候用到了state，并没有在其他地方使用过这些state。这个例子中我们并不关心这些state值的变化，我们只关心我们提交的数据是否正确。而且我们每次输入的时候组件都是重新渲染。这并不友好，这个时候我们需要非受控组件。

### 解决方法

当表单元素不多时，使用ref来处理，并且每次输入都不会引起组件的重新渲染，因为这个时候我们只关心提交的数据，没有在其他地方使用过这些state。

javascript复制代码

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

export default function NoState() {
  const usernameRef = useRef();
  const passwordRef = useRef();

  const onSubmit = (e) => {
    e.preventDefault();
    console.log(
      "需要提交的数据",
      usernameRef.current.value,
      passwordRef.current.value
    );
  };

  console.log("组件重新渲染了");

  return (
    <form onSubmit={onSubmit}>
      <label htmlFor="name">名字</label>
      <input type="text" ref={usernameRef} />
      <br />
      <label htmlFor="name">密码</label>
      <input type="text" ref={passwordRef} />
      <br />
      <button type="submit">提交</button>
    </form>
  );
}
```

## 过多的useEffect

### 问题

有时当页面第一次挂载时，我们需要进行网络请求，我们经常会这样写：

scss复制代码

```scss
import { useEffect, useState } from "react";

export default function MoreUseEffect() {
  const [data, setData] = useState();

  useEffect(() => {
    fetch("/ss/ss").then((res) => {
      setData(res.data);
    });
```

```
  }, []);

  useEffect(() => {
    // 进行其他逻辑处理...
    console.log(data);
  }, [data]);

  return <>页面第一次加载时请求</>;
}
```

引入了过多的useEfffect，实际上我们只是需要使用请求到的数据来进行其他逻辑的处理，并不需要数据变化时做一些事情。

### 解决方法

把数据的处理逻辑放入第一个useEffect中直接处理。

javascript复制代码

```
import { useEffect } from "react";

export default function MoreUseEffect() {

  useEffect(() => {
    fetch("/ss/ss").then((res) => {
      // setData(res.data);
      // 在这里直接进行数据处理...
      console.log('')
    });
  }, []);

  return <>页面第一次加载时请求</>;
}
```

## 请求竞争问题

### 问题

下面是对fetch请求进行了封装，这种写法有一个问题：当同时有多个请求时，由于请求返回的时间不一样，会出现竞争关系，不会按照请求的顺序返回结果，这样就造成返回的结果不知道是哪次的。

scss复制代码

```
import { useEffect, useState } from "react";

export default function useFetch(url) {
  const [loading, setLoading] = useState(true);
  const [data, setData] = useState();
  const [error, setError] = useState();

  useEffect(() => {
```

```
    setLoading(true);
    fetch(url)
      .then((res) => {
        setData(res.data);
      })
      .catch((e) => {
        setError(e);
      })
      .finally(() => setLoading(false));
  }, [url]);

  return {
    loading,
    data,
    error
  };
}
```

## 解决方法

需要在请求URL变化之后取消前一次的请求。

scss复制代码

```
import { useEffect, useState } from "react";

export default function useFetch(url) {
  const [loading, setLoading] = useState(true);
  const [data, setData] = useState();
  const [error, setError] = useState();

  useEffect(() => {
    const controller = new AbortController();

    setLoading(true);
    fetch(url, { signal: controller.signal })
      .then((res) => {
        setData(res.data);
      })
      .catch((e) => {
        setError(e);
      })
      .finally(() => setLoading(false));
    return () => {
      controller.abort();
    };
  }, [url]);

  return {
    loading,
    data,
    error
  };
}
```

## 使用三元表达式代替&&

### 使用 && 常见的错误

1.当状态值不是Boolean，而是数字0时，数字0会在UI中显示。

javascript复制代码

```
import { useState } from "react";

export default function MoreUseEffect() {
       const [arr] = useState([])

  return <>
    {
      arr.length && <div>11111</div>
    }
    </>;
}
```




### 解决方法

1. 转成Boolean
2. 使用三元表达式代替 && (**推荐**)

## 传递特殊属性ref

### 问题

ref属性是React的特殊属性，不能直接传递使用。

javascript复制代码

```
import {useRef} from 'react'

function InputCom({ref}) {
  return (
    <input type='text' ref={ref} />
  )
}

function App() {
  const inpRef = useRef(null)

  const focus = () => {
```

```
    inpRef.current?.focus()
  }

  return (
    <>
      <InputCom ref={inpRef} />
    </>
  )
}
```

```
Warning: InputCom: `ref` is not a prop. Trying to access it will result in `undefined` being returned. If you need to access the same value within the child component, you should pass it as a different prop. (https://reactjs.org/link/special-props)
```



如果想传递ref需要借助forwardRef函数。

## 解决方法

借助forwardRef转发ref属性

javascript复制代码

```javascript
import { forwardRef, useRef } from "react";

// function InputCom({ ref }) {
//   return <input type="text" ref={ref} />;
// }

const InputCom = forwardRef((props, ref) => {
  return <input type="text" ref={ref} />;
});

export default function ProRef() {
  const inpRef = useRef(null);

  const focus = () => {
    inpRef.current?.focus();
  };

  return (
    <>
      <InputCom ref={inpRef} />
      <br />
      <button onClick={focus}>focus</button>
    </>
  );
}
```