react

react-hook

什么叫做副作用

react-hook中的副作用指的就是谁的变化会引起render

自定义hook与function有什么区别

Hooks只能在React函数组件内调用。无法在普通函数内调用。普通函数中无法调用useState，useEffect等。

如何在普通函数中调用react的hook呢，将普通函数定义成自定义Hook。所有的函数必须以use开头use[functionName],这样就可以在函数中使用hooks了

举例：

```
import { useCallback, useEffect, useState } from "react";
import { Button } from "antd";

const testFunc = () => {
  const [count, setCount] = useState<number>(0);
  const add = useCallback(() => {
    setCount(prevState => prevState + 1);
    console.log(count);
  }, [setCount]);
  return {add,count}
}
export default function CallbackDemo() {
  const {add,count} = testFunc();
  return (
    <>
      <Button onClick={() => {
        add();
      }}>{count}</Button>
    </>
  )
}
```

这里会报错： Line 5:29: React Hook "useState" is called in function "testFunc" that is neither a React function component nor a custom React Hook function. React component names must start with an uppercase letter. React Hook names must start with the word "use" react-hooks/rules-of-hooks Line 6:15: React Hook "useCallback" is called in function "testFunc" that is neither a React function component nor a custom React Hook function. React component names must start with an uppercase letter. React Hook names must start with the word "use" react-hooks/rules-of-hooks

需要改成:

```
import { useCallback, useEffect, useState } from "react";
import { Button } from "antd";

const useTestFunc = () => {
  const [count, setCount] = useState<number>(0);
  const add = useCallback(() => {
    setCount(prevState => prevState + 1);
    console.log(count);
  }, [setCount]);
  return {add,count}
}
export default function CallbackDemo() {
  const {add,count} = useTestFunc();
  return (
    <>
      <Button onClick={() => {
        add();
      }}>{count}</Button>
    </>
  )
}
```

什么变化可以被监听到

1. deps中可以被监听到的变化类似一个指向一块内存的指针，引用内部的变化是不会被察觉的。

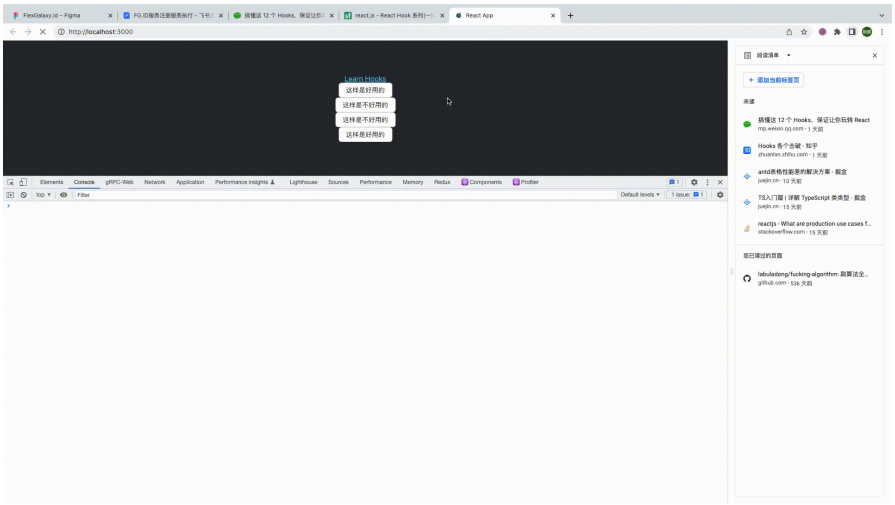
举例：

```
import { useCallback, useEffect, useState } from "react";
import { Button } from "antd";


export default function CallbackDemo() {
  const [testObj, setTestObj] = useState({ a: 1 });

  const testObjCallback = useCallback(() => {
    console.log("testObjCallback",testObj);
  }, [testObj]);
  const testObjPropsCallback = useCallback(() => {
    console.log("testObjPropsCallback",testObj.a);
  }, [testObj.a]);
  useEffect(() => {
    testObjCallback();
    testObjPropsCallback();
  }, [testObjCallback]);

  return (
    <>
      <Button onClick={() => {
        setTestObj({ a: 2 });
      }}></Button>
      <Button onClick={() => {
        testObj.a += 1;
        setTestObj(testObj);
        console.log(testObj)
      }}></Button>
      <Button onClick={() => {
        setTestObj((prevState) => {
          prevState.a += 1;
          return prevState;
        });
        console.log(testObj)
      }}></Button>
      <Button onClick={() => {
        setTestObj((prevState) => {
          const newState = { ...prevState }
          newState.a += 1;
          return newState;
        });
        console.log(testObj)
      }}></Button>
    </>
  )
}
```



各类hook详解

useMemo

useMemo:

1. 第一个参数为一个函数，会返回该函数的执行结果

用法举例：

1. 需求：

useRef

useRef:

1. 在页面re-render的时候不render useRef中的值；
2. 获取dom属性

用法举例：

1. 需求：

```
export default function ApiVersion() {
  const pageManager = useRef(new PageManager(0, 10, 1));
  return  (
    <Table<ApiVersion>
      pagination={{
        ...pagination,
      }}
      onChange={(pagination) => {
        pageManager.current.current = pagination.current;
      }}
    />
  )
}
```

useCallback

useCallback:

1. 第一个参数为一个函数，会返回该函数
    a. 第一个参数在声明时不会被执行
    b. useMemo在声明的时候就会执行第一个参数

用法举例:

1. 需求：

常用的自定义hook

使用hook模拟生命周期

useLasted

Hooks的实现原理

为什么不能在循环/判断中使用hook