哎，这是你的React面试小抄吧

基础篇

如果是在组件内部定义，类组件可以定义this.state上，函数组件可以使用useState或者useReducer定义。这种适合不需要多组件共享的state。

如果是在组件外部定义，可以使用常见的状态管理库，如redux、mobx等，也可以自行定义，只要保证组件能够随之更新就行。这种适合多组件共享的state。

拓展：redux、mobx、reocil、form等原理、hooks原理等。

hooks出现之前，函数组件内部无法定义state，主要是因为函数组件每次更新，定义在函数体里的值都要重新初始化，没法保存。

而hooks提供的useState或者useReducer可以让函数组件在组件内定义state，每一个hook都有个对应的hook对象，这个对象上会存储状态值、reducer等值，这个hook对象又以单链表的数据结构存在fiber上，而fiber是React的虚拟DOM，存在于内存中。

拓展：分析下hook对象的具体值。

```
js 复制代码

useEffect(creater, deps);
```

deps数组为空，则相当于componentDidMount。deps有值，则相当于componentDidUpdate。creater的返回值在组件卸载或者更新前执行。

但是与 `componentDidMount`、`componentDidUpdate` 不同的是，传给 `useEffect` 的函数会在浏览器完成布局与绘制**「之后」**，在一个延迟事件中被调用。

拓展：与useLayoutEffect的区别，以及各自的使用场景。

可以的。

middleware只是包装了 store 的 dispatch 方法。技术上讲，任何 middleware 能做的事情，都可能通过手动包装 dispatch 调用来实现，但是放在同一个地方统一管理会使整个项目的扩展变的容易得多。

拓展：redux-thunk、redux-promise、redux-saga等中间件的使用以及原理。

---

redux store 的原版 dispatch 只能接受 plainObject 类型的参数，使用了 thunk 之后，可以处理函数类型的参数，主要用于异步。

```js 复制代码
export default function isPlainObject(obj: any): boolean {
  if (typeof obj !== 'object' || obj === null) return false

  let proto = obj
  while (Object.getPrototypeOf(proto) !== null) {
    proto = Object.getPrototypeOf(proto)
  }

  return Object.getPrototypeOf(obj) === proto
}
```

拓展：redux、redux-thunk 原理。

---

react-router 使用的 history 库，

原理源码篇

1. createStore 创建store 数据状态管理库
2. reducer 初始化、修改状态函数，**「定义修改规则」**
3. getState 获取状态值 （getter）
4. dispatch 提交更新 (setter)
5. subscribe 变更订阅 订阅state改变之后要做的事情，一般是组件更新

下面是一个简版的redux实现：

```js
export default function createStore(reducer) {
   // 状态值
  let currentState;
  // 响应函数数组
  let currentListeners = [];

  // 获取状态
  function getState() {
    return currentState;
  }

  // 修改状态、执行相应函数
  function dispatch(action) {
    currentState = reducer(currentState, action);
    currentListeners.forEach(listener => listener());
    return action;
  }

  // 订阅响应函数
  function subscribe(listener) {
    currentListeners.push(listener);
    // 取消订阅函数
    return () => {
      const index = currentListeners.indexOf(listener);
      currentListeners.splice(index, 1);
    };
  }

  // 初始化状态值
  dispatch({type: "YYYY/0000"});

  return {
    getState,
```

```
    dispatch,
    subscribe
  };
}
```