react优化建议——文章整理

- *原文链接*
- 编译阶段
  - 优化babel配置，webpack 配置项
- 路由阶段
  - 路由懒加载，路由监听器
- 渲染阶段
  - **受控组件颗粒化，独立请求服务选择单元**
    - 颗粒化控制可控性组件

      可控性组件和非可控性的区别就是dom元素值是否与受到react数据状态state控制。一旦由react的state控制数据状态，比如input输入框的值，就会造成这样一个场景，为了使input值实时变化，会不断setState，就会不断触发render函数，如果父组件内容简单还好，如果父组件比较复杂，会造成牵一发动全身，如果其他的子组件中componentWillReceiveProps这种带有副作用的钩子，那么引发的蝴蝶效应不敢想象。
    - 建立独立的请求渲染单元

      建立独立的请求渲染单元，直接理解就是，如果我们把页面，分为请求数据展示部分(通过调用后端接口，获取数据)，和基础部分(不需要请求数据，已经直接写好的)，对于一些逻辑交互不是很复杂的数据展示部分，我推荐用一种独立组件，独立请求数据，独立控制渲染的模式。
      - 1 可以避免父组件的冗余渲染 ，react的数据驱动，依赖于state 和 props 的改变，改变state 必然会对组件 render 函数调用，如果父组件中的子组件过于复杂，一个自组件的state 改变，就会牵一发动全身，必然影响性能，所以如果把很多依赖请求的组件抽离出来，可以直接减少渲染次数。
      - 2 可以优化组件自身性能，无论从class声明的有状态组件还是fun声明的无状态，都有一套自身优化机制，无论是用shouldupdate 还是用 hooks中 useMemo useCallback ，

都可以根据自身情况，定制符合场景的渲条 件，使得依赖数据请求组件形成自己一个小的，适合自身的渲染环境。

- 3 能够和redux ,以及redux衍生出来 redux-action , dva,更加契合的工作，用 connect 包裹的组件，就能通过制定好的契约，根据所需求的数据更新，而更新自身，而把这种模式用在这种小的，需要数据驱动的组件上，就会起到物尽其用的效果。

- **`shouldComponentUpdate` , `PureComponent` 和 `React.memo` , `immetable.js` 助力性能调优**
  - `PureComponent` 和 `React.memo` React API
    - `React.PureComponent` 通过props和state的浅对比来实现 `shouldComponentUpate()`

      如果对象包含复杂的数据结构(比如对象和数组)，他会浅比较，如果深层次的改变，是无法作出判断的，`React.PureComponent` 认为没有变化，而没有渲染试图。

      无论组件是否是 `PureComponent` ，如果定义了 `shouldComponentUpdate()` ，那么会调用它并以它的执行结果来判断是否 update。在组件未定义 `shouldComponentUpdate()` 的情况下，会判断该组件是否是 `PureComponent` ，如果是的话，会对新旧 props、state 进行 `shallowEqual` 比较，一旦新旧不一致，会触发渲染更新。
    - `react.memo` 和 `PureComponent` 功能类似 , `react.memo` 作为第一个高阶组件，第二个参数 可以对 props 进行比较 ， 和 `shouldComponentUpdate` 不同的，当第二个参数返回 true 的时候，证明 `props` 没有改变，不渲染组件，反之渲染组件。
  - `shouldComponentUpdate`
    - 使用 `shouldComponentUpdate()` 以让 `React` 知道当 `state` 或 `props` 的改变是否影响组件的重新 `render` ，默认返回 `ture` ， 返回 `false` 时不会重新渲染更新，而

且该方法并不会在初始化渲染或当使用 `forceUpdate()` 时被调用

- `immetable.js`
  - `immetable.js` 是Facebook 开发的一个js库，可以提高对象的比较性能，像之前所说的pureComponent 只能对对象进行浅比较，,对于对象的数据类型,却束手无策,所以我们可以用 `immetable.js` 配合 `shouldComponentUpdate` 或者 `react.memo` 来使用

- 细节优化
  - 绑定事件尽量不要使用箭头函数
    - 众所周知，react更新来大部分情况来自于props的改变(被动渲染)，和state改变(主动渲染)。当我们给未加任何更新限定条件子组件绑定事件的时候，或者是PureComponent 纯组件， 如果我们箭头函数使用的话,每次渲染时都会创建一个新的事件处理器，这会导致 ChildComponent 每次都会被渲染。
      - 如果我们需要传递参数。我们可以这么写。

```
function index(){
    const handerClick1 = useMemo(()=>(event)=>{
        const mes = event.currentTarget.dataset.mes
        console.log(mes) /* hello,world */
    },[])
    return <div>
        <div  data-mes={ 'hello,world' } onClick={ handerClick1 }  >hello,world</div>
    </div>
}
```

  - 循环正确使用key
  - 无状态组件 `hooks-useMemo` 避免重复声明。
  - 懒加载 `Suspense` 和 `lazy`

- 状态管理

- 学会使用的批量更新
- 合并 `state`
- `useMemo` `React.memo` 隔离单元
- ‘取缔’ `state` ，学会使用缓存。
- `useCallback` 回调
  - `useCallback` 的真正目的还是在于缓存了每次渲染时 inline callback 的实例，这样方便配合上子组件的 `shouldComponentUpdate` 或者 `React.memo` 起到减少不必要的渲染的作用。对子组件的渲染限定来源与，对子组件 `props` 比较，但是如果对父组件的callback做比较，无状态组件每次渲染执行，都会形成新的 `callback` ,是无法比较，所以需要对 `callback` 做一个 `memoize` 记忆功能，我们可以理解为 `useCallback` 就是 `callback` 加了一个 `memoize` 。
- **海量数据源、长列表渲染**

以上内容整理于 幕布文档