**react的不足？**

react中只是MVC模式的View部分，要依赖引入很多其他模块开发。

当父组件进行重新渲染操作时，即使子组件的props或state没有做出任何改变，也会同样进行重新渲染。

--------

**react 生命周期函数？**

初始化阶段：

getDefaultProps:获取实例的默认属性

getInitialState:获取每个实例的初始化状态

componentWillMount：组件即将被装载、渲染到页面上，只调用一次（可以在调用echars的时候，在antd中改变一个值将所有的颜色都改变）

render:组件在这里生成虚拟的 DOM 节点

componentDidMount:组件真正在被装载之后（在这步请求ajax，因为在这一步渲染出了真实dom节点，不会去避免一些不必要的更新不到的值，在这一步我们能完全加载出来）

运行中状态：

componentWillReceiveProps:组件将要接收到属性的时候调用

shouldComponentUpdate:组件接受到新属性或者新状态的时候（可以用purecomponent来进行优化，就是在shouldUpdate的那一步再一次进行了diff算法封装，加了浅层的对比，）（可以返回 false，接收数据后不更新，阻止 render 调用，后面的函数不会被继续执行了）

（打个比方，我们现在用redux,我这个组件connect连接到仓库，另外一个组件的值连接到仓库，另外一个仓库的值发生改变，而我这个组件也连接了仓库，仓库发生改变，仓库里state发生改变，这个view，state都没改变的组件也是要重新渲染的这就极大的浪费了性能，所以我们这部需要在进行一层优化在进行千层的对比，判断我们的state和store有没有发生改变）

componentWillUpdate:组件即将更新不能修改属性和状态

render:组件重新描绘

componentDidUpdate:组件已经更新

销毁阶段：

componentWillUnmount:组件即将销毁

-------------------

**redux?**

store:保存数据的地方，你可以把它看成一个容器，整个应用只能有一个Store

State:包含所有数据，表示某个时点的数据集合

Action: State的变化，会导致View的变化，就是View发出的通知，表示State应该就要发生变化了。

Action Creator:View要发送多少种消息，就会有多少种Action,我们定义Action Creator函数来生成Action

Reducer: Store收到Action以后，必须给出一个新的State,这样View才会发生改变，这种State的计算过程就叫Reducer

它是一个函数，它接收Action和当前State作为参数，返回一个新的State

dispatch: 是View发出Action的唯一方法

redux 是一个应用数据流框架，主要是解决了组件间状态共享的问题，原理是集中式管理，主要有三个核心方法，action，store，reducer，

工作流程是 view 调用 store 的 dispatch 接收 action 传入 store，reducer 进行 state 操作，view 通过 store 提供的 getState 获取最新的数据，

redux 中只能定义一个可更新状态的 store，redux 把 store 和 Dispatcher 合并,结构更加简单清晰

----**redux原理**

用户点击了一个ui按钮，通过dispatch，发出Action，然后store自动调用reducer,并且传入了两个参数，当前State和收到的Action.Reducer会返回新的State.每当state更新之后，view会根据state触发重新渲染

---

**react-redux实现原理？？**

react-redux是一个轻量级的封装库，它主要通过两个核心方法实现：

Provider：从最外部封装了整个应用，并向connect模块传递store。

Connect：

1、包装原组件，将state和action通过props的方式传入到原组件内部。

2、监听store tree变化，使其包装的原组件可以响应state变化

-----

**redux的性能优化？**

1render里要尽量减少新建变量和bind的使用，传递参数时尽量减少传递参数的数量，bind一般放在constructor中

2shouldComponentUpdate是决定react组件什么时候能够不重新渲染的函数。这个函数默认的实现方式就是简单的返回一个true。

默认每次更新的时候都会调用所用的生命周期函数，包括render函数，重新渲染。

3 immutable对象管理状态，让状态不能被更改。

4 key值唯一 利用diff算法中的key值

5将组件component更换为pureComponent也可以进行优化

6可以用到reselect（数据获取时优化的）都是优化渲染来提高性能的 原理是 只要相关的状态没发生改变，那么就直接使用上一次的缓存结果

-----

react-router?

**react-router的实现原理？**

原理：实现URL与UI界面的同步。其中在react-router中，URL对应Location对象，

而UI是由react components来决定的，这样就转变成location与components之间的同步问题。

withRouter 高阶组件，自带三个对象history,location,match

通过高阶组件包裹可以获得这三个对象实现路由跳转 传值

动态路由？就是通过路由传值，一般通过router里的path 路径后面要添加

传递的值可在location里的search获取我们传递的内容，也可以通过link标签的

to的地址后面

**hash和history区别**

hash就是指url尾巴后的#号以及后面的字符。这里的#和css里的#是一个意思。hash也称作锚点，本身是用来做页面定位的，他可以使对应的id元素显示在可视区域内。由于hash值变化不会导致浏览器向服务器发出请求，而且hash改变会触发hashchange事件，浏览器的进后退也能对其进行控制，所以人们在html5的history出现前，基本都是使用hash来实现前端路由的。他的特点在于：hash虽然出现url中，但不会被包含在HTTP请求中，对后端完全没有影响，因此改变hash不会重新加载页面。hash 本来是拿来做页面定位的，如果拿来做路由的话，原来的锚点功能就不能用了。其次，hash的而传参是基于url的，如果要传递复杂的数据，会有体积的限制。

history模式不仅可以在url里放参数，还可以将数据存放在一个特定的对象中。  
history———利用了HTML5 History Interface 中新增的pushState（）和replaceState（）方法。

-----

**react-router的优缺点：**

1.风格: 与React融为一体,专为react量身打造，编码风格与react保持一致，例如路由的配置可以通过component来实现

2.简单: 不需要手工维护路由state，使代码变得简单

3.强大: 强大的路由管理机制，体现在如下方面

4.路由配置: 可以通过组件、配置对象来进行路由的配置

5.路由切换: 可以通过<Link> Redirect进行路由的切换

6.路由加载: 可以同步记载，也可以异步加载，这样就可以实现按需加载

7.使用方式: 不仅可以在浏览器端的使用，而且可以在服务器端的使用

-----

**React 中 keys 的作用是什么？**

Keys 是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。

在开发过程中，我们需要保证某个元素的 key 在其同级元素中具有唯一性。在 React Diff 算法中 React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重渲染

=-------

**setState的原理及用法？**

当调用setState时，它并不会立即改变，而是会把要修改的状态放入一个任务队列，等到事件循环结束时在合并更新。

常见用法就是传入一个对象，还可以接收一个参数，因为setState是异步的，所以它还可以接收第二个参数，一个回调函数

----------

**setState是同步？？还是异步？？**

答：setState在只有特殊的只有setTimeout跟触发函数的时候，是异步的，其他时候都是同步的。

**为什么会触发异步哪**，

因为提供了一个优化，在每次setState的时候，就会重新渲染一次dom,为了那些不必要的渲染，

我用异步的方法，把这些想要渲染的值全部压到最后，压到最后等着一起渲染出来，这些值更新完全部一起加载出来，然后

再去更新state的值。这样就引发了一次dom渲染，如果是同步的话更新一个，立马就会去刷dom，太浪费性能。

**调用 setState 之后发生了什么？**

在代码中调用 setState 函数之后，React 会将传入的参数对象与组件当前的状态合并，然后触发所谓的调和过程（Reconciliation）。

经过调和过程，React 会以相对高效的方式根据新的状态构建 React 元素树

并且着手重新渲染整个 UI 界面。在 React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，

然后根据差异对界面进行最小化重渲染。在差异计算算法中，React 能够相对精确地知道哪些位置发生

了改变以及应该如何改变，这就保证了按需更新，而不是全部重新渲染。

------

**shouldComponentUpdate 是做什么的，（react 性能优化是哪个周期函数？）**

shouldComponentUpdate 这个方法用来判断是否需要调用 render 方法重新描绘 dom。

因为 dom 的描绘非常消耗性能，如果我们能在 shouldComponentUpdate 方法中能够写出更优化的 dom diff 算法，可以极大的提高性能。

=======

**真实dom和虚拟dom的区别？？**

真实dom:更新慢，可以直接更新html,如果元素更新，则创建新dom,dom操作代价高，消耗内存较多。

虚拟dom:更新快，无法直接更新Html 如果元素更新，则更新jsx DOM操作非常简单 很好的内存消耗

---------

**为什么虚拟 dom 会提高性能?(必考)**

1. 用 js对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中。

2. 当状态变更的时候，重新构造一棵新的对象树。然后对比新旧虚拟DOM树，记录两棵树差异。

3. 把 2 所记录的差异应用到步骤 1 所构建的真正的 DOM 树上，视图就更新了。

原因：虚拟 dom 相当于在 js 和真实 dom 中间加了一个缓存，利用 dom diff 算法减少了对真实DOM的操作次数，从而提高性能。

--------

**react怎么从虚拟dom中拿出真实的dom ?**

Refs是react提供给我们的安全访问DOM元素或者某个组件实例的，可以为元素添加ref属性然后在回调函数中接收该元素，该值

会作为回调函数的第一个参数返回，或者ref可以传字符串

--------

**react diff 原理（常考，大厂必考）**

把树形结构按照层级分解，只比较同级元素。

给列表结构的每个单元添加唯一的 key 属性，方便比较。

React 只会匹配相同 class 的 component（这里面的 class 指的是组件的名字）

合并操作，调用 component 的 setState 方法的时候, React 将其标记为 dirty.到每一个事件循环结束, React 检查所有标记 dirty 的 component 重新绘制.

选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

----------

**React 中 refs 的作用是什么？**

Refs 是 React 提供给我们的安全访问 DOM 元素或者某个组件实例的句柄。我们可以为元素添加 ref 属性然后在回调函数中接受该元素在 DOM 树中的句柄

该值会作为回调函数的第一个参数返回：

---------

**redux中间件？**

作用？负责拦截action,去实现一些操作，通过自己的next转发action

next方法的作用？如果只有一个中间件，就会直接转发给reducer,

run 方法的所用自动调用next方法并且执行。

logger:负责打印我们之前的state和之后的state

thunk 用来执行异步操作的。

优点：学习成本低。

缺点：(1).一个异步请求的action代码过于复杂，且异步操作太分散，相对比saga只要调用一个call方法就显得简单多了。

(2).action形式不统一，如果不一样的异步操作，就要写多个了。

------

promise 简化了thunk的写法，也可以用来执行异步操作

redux-saga?根据generator函数来执行异步操作，用同步的写法企业写

异步代码，有个run 方法，会遍历yiels语句

会返回一个遍历器对象，

------

优点：（1）集中处理了所有的异步操作，异步接口部分一目了然(有提供自己的方法)

（2）action是普通对象，这跟redux同步的action一模一样({type:XXX})

（3）通过Effect，方便异步接口的测试

（4）通过worker和watcher可以实现非阻塞异步调用，并且同时可以实现非阻塞调用下的事件监听

（5） 异步操作的流程是可以控制的，可以随时取消相应的异步操作。

----

//reduxsaga的副作用？

takeEvery:两个参数，第一个接收的是加参数，第二个是要执行的generaor函数

takeLastest:防止重复提交，当点击一次 只会执行最后一次

put作用：转发action

call作用？阻塞的作用，异步不执行完毕，不会执行后面的yield语句

fork的作用？非阻塞，不会等待结果，直接执行后面的yield语句

all的作用？合并saga

-------------

**什么是高阶组件？？**

高阶组件是参数为组件，返回值为新组件的函数。

把不是通过路由切换过来的组件中，将react-router 的 history、location、match 三个对象传入props对象上

-----

**受控组件和非受控组件？**

针对表单而言的

表单元素依赖于状态，表单元素需要默认值实时映射到状态的时候，就是受控组件，这个和双向绑定相似.

受控组件，表单元素的修改会实时映射到状态值上，此时就可以对输入的内容进行校验.

受控组件只有继承React.Component才会有状态.

受控组件必须要在表单上使用onChange事件来绑定对应的事件.

非受控组件即不受状态的控制，获取数据就是相当于操作DOM。

非受控组件的好处是很容易和第三方组件结合。

-----

**react与vue的对比**

相同点：1. 都用虚拟DOM实现快速渲染

2. 我觉得父子，兄弟通信这些都挺像的，也都有自己的状态管理器：react=>redux, vue=>vuex

3. 都是轻量级框架

4. 现在vue也在渐渐吸收react中的一些语法，比如JSX语法，类式声明写法等

不同点：

1. React属于单向数据流——MVC模式，vue则属于双向——MVVM模式。

2. react兼容性比vue好，vue不兼容IE8.

3. react采用JSX语法，vue采用的则是html模板语法。

4. vue的css可以有组件的私有作用域，react则没有。

5. react比vue好的另一点是，它是团队维护，而vue属于个人，一般来说，大型项目更倾向于react，小型则用vue，当然这也不是绝对。

----------

**比较redux和vuex?**

相同点：1.数据驱动视图，提供响应式的视图组件

2.都有virtual DOM， 组件化开发，通过props参数进行父子组件数据的传递，都实现webComponents规范

3.都支持服务端渲染

4.都有native解决方案，reactnative（facebook团队） vs weex（阿里团队）

不同点：1.vuex是一个针对VUE优化的状态管理系统，而redux仅是一个常规的状态管理系统（Redux）与React框架的结合版本。

2.开发模式：React本身，是严格的view层，MVC模式；Vue则是MVVM模式的一种方式实现

3.数据绑定：Vue借鉴了angular，采取双向数据绑定的方式；React，则采取单向数据流的方式

4.数据更新：Vue采取依赖追踪，默认是优化状态：按需更新；

React在则有两种选择：

1）手动添加shouldComponentUpdate，来避免冗余的vdom，re-render的情况

2）Components 尽可能都用 pureRenderMixin，然后采用 redux 结构 + Immutable.js

5.社区：react相比来讲还是要大于vue，毕竟背后支撑团队不同。

facebook vs 个人！当然目前vue的增长速度是高于react的增速，不知道未来的发展趋势是如何。

总之：期待构建一个大型应用程序——选择React，期待应用尽可能的小和快——选择Vue

--------------------------

**为什么浏览器无法读取jsx？**

浏览器只能处理 JavaScript 对象，而不能读取常规 JavaScript 对象中的 JSX。所以为了使浏览器能够读取 JSX，首先，需要用像 Babel 这样的 JSX 转换器将 JSX 文件转换为 JavaScript 对象，然后再将其传给浏览器。

----

**如何理解一切皆组件这句话？？**

组件是 React 应用 UI 的构建块。这些组件将整个 UI 分成小的独立并可重用的部分。每个组件彼此独立，而不会影响 UI 的其余部分。

----

**有状态组件和无状态组件的区别？？**

有状态组件：

1. 在内存中存储有关组件状态变化的信息 1. 计算组件的内部的状态

2. 有权改变状态 2. 无权改变状态

3. 包含过去、现在和未来可能的状态变化情况 3. 不包含过去，现在和未来可能发生的状态变化情况

4. 接受无状态组件状态变化要求的通知，然后将 props 发送给他们。 4.从有状态组件接收 props 并将其视为回调函数。

**什么是HOOKS?**

**Hooks**是 React 16.8 中的新添加内容。它们允许在不编写类的情况下使用state和其他 React 特性。使用 Hooks，可以从组件中提取有状态逻辑，这样就可以独立地测试和重用它。Hooks 允许咱们在不改变组件层次结构的情况下重用有状态逻辑，这样在许多组件之间或与社区共享 Hooks 变得很容易。

**浏览器兼容问题：**

1. ****不同浏览器的标签默认的外补丁和内补丁不同****
2. ****块属性标签float后，又有横行的margin情况下，在**[IE6](http://xiazai.zol.com.cn/detail/11/102978.shtml" \t "https://blog.csdn.net/Sweet__Cat/article/details/_blank)**显示margin比设置的大****
3. ****设置较小高度标签（一般小于10px），在IE6，**[IE7](http://xiazai.zol.com.cn/detail/14/130213.shtml" \t "https://blog.csdn.net/Sweet__Cat/article/details/_blank)**，遨游中高度超出自己设置高度****
4. ****行内属性标签，设置display:block后采用float布局，又有横行的margin的情况，IE6间距bug****
5. ****图片默认有间距****
6. ****标签最低高度设置min-height不兼容****
7. ****透明度的兼容CSS设置****

****dva的原理：数据的改变发生通常是通过用户交互行为或者浏览器行为触发的，当行为会改变数据的时候可以通过dispatch发起一个action，如果是同步会直接通过ruducers改变state，如果是异步行为先会触发Effects然后流向Reducers最终改变state。****