react全家桶

（1）运用jsx语法

（2）React生命周期

（3）前后端交互用到的fetch/axios

（4）React-router路由

（5）数据流解决方案redux

（6）React-redux

（7）中间件

（8）Dva框架

（9）Umi框架

**react的优缺点以及特点**

优势：

1. 引入了虚拟DOM和diff算法，提高了Web性能。

2. 组件化，模块化开发。便于我们后期维护性。

3. 单向数据流，比较有序，便于数据管理。

不足：

1. react中只是MVC模式的View部分，所以在react开发中要依赖很多其他模块。

2. 当父组件进行重新渲染时，即使子组件的props或state没有任何改变，也会同样进行重新渲染。（react如何避免重复渲染）

特点：

1. 声明式设计（React采用声明式，可以轻松描述应用。）

2. 高性能（React通过对DOM的模拟，最大限度地减少与DOM的交互。）

3. 灵活（React可以与已知的库或框架很好地配合。）

在React如何做性能优化？

1. 给DOM的遍历上加上唯一的key。提高diff算法效率。

尽量不要用index,如果说在DOM中删了某一个节点，index也就会发生变化，这时候就会重新渲染，所以key值最好使用id。

1. 能用const声明的就用const。

在render里面尽量减少新变量的创建以及函数的指向改变问题。

3. 减少对真实DOM的操作。

4. 如果是用webpack搭建环境的话，当一个包过大加载过慢时，可分成多个包来优化。

5. 使用react-loadable，按需加载路由

6. 定时器，超时器使用过后在unmount中清除。

7. immutable对象管理状态 让状态不能被更改

8. 把component更换成pureComponent 也可以进行优化

你对pureComponent的理解

1.当更新时，如果组件的props或者state都没有改变，render函数就不会触发。

2.省去虚拟DOM的生成和对比过程，达到提升性能的目的。

3.具体原因是因为react自动帮我们做了一层浅比较

简述react项目的开发过程？

答：

安装node环境   
 使用creat-react-app创建项目  
 通过npm start运行项目  
 查看生产环境的项目目录及结构  
 开始写自己的组件 然后拼装组件  
 然后在查看项目 没问题了  
 在通过webpack打包到生产环境中   
 然后提交到公司git服务器上   
 然后链条测试  
 上线

react生命周期及相关用法

react生命周期分为mount阶段、update阶段、unmount阶段。

(1) mount阶段：

componentWillMount：实例挂载之前。进行初始化数据的交互。

Render：渲染组件

componentDidMount：实例挂载完成。在这个函数中，实现前后端数据交互。

(2) update阶段：

componentWillReceiveProps：父组件改变时调用。

sholudComponentUpdate：是否重新渲染组件，一般在这个函数中做组件的性能优化。

componentWillUpdate：组件数据更新前调用

componentDidUpdate：组件数据更新完成时调用

(3) unmount阶段：

componentWillUnmount：一般用来销毁不用的变量或者是解除无用定时器以及解绑无用事件。防止内存泄漏问题。

react如何避免重复渲染

React 官方提供了 PureRenderMixin 插件，它的功能就是在不需要重新渲染的情况下让shouldComponentUpdate 返回 false, 使用这个插件就能够减少不必要的重新渲染，性能得到也得到一些提升。

但是在 React 的新版本里面，提供了 React.PureComponent，而不需要使用这个插件。 所以说一个较大的组件决定重渲染的时候，我们可以在每一个子组件中绑定（新的）shouldComponentUpdate方法，这样可以减少子组件重新渲染的次数。

无状态组件和有状态组件有什么区别

**无状态组件**

无状态组件的创建形式使代码的可读性更好，而且减少了大量冗余的代码，以至于只有一个render方法。

组件不会被实例化

组件不能访问this对象，也不能访问生命周期方法

无状态组件只能访问输入的props ，同样的props渲染同样的结果，而且没有副作用。

**有状态组件**

React.Component创建的组件，其成员函数不会自动绑定this，如果我们没有手动绑定this，就不能获取当前组件实例对象。

React.Component创建的组件，其状态state是在constructor中像初始化组件属性一样去声明。

补充：React.Component有三种手动绑定方法

1》constructor(props) *{*

*super(props);*

*this.handleClick = this.handleClick.bind(this); //构造函数中绑定*

*}*

2》<div onClick={this.handleClick.bind(this)}></div> *//使用bind来绑定*

3》<div onClick={()=>this.handleClick()}></div> *//使用arrow function来绑定*

如何选择哪种方式创建组件

只要有可能，尽量使用无状态组件。

如果需要state、生命周期方法等，使用class的创建组件

react在哪个生命周期做优化?

可以在shouldComponentUpdate这个方法进行优化，用来判断是否调用 render 方法重绘 dom。  
因为 dom 的重绘非常消耗性能，所以可以在这个方法中去做dom的diff算法的优化，这样就可以极大的提高性能。

但对于我们中级工程师来说，一般我不会去修改shouldComponentUpdate这个方法。而是直接去使用。 True/false

react组件间传值。（组件间通信方式）

（1）props传值，父组件通过props向子组件传值，子组件通过回调函数向父组件传值

（2）redux跨级通信。

（3）通过prop-types的context实现跨级通信

跨级组件间双向通信：使用 context 对象，根组件和其他所有子孙通信，

不太适合组件间通信（可以实现，不好维护）

（4）使用事件订阅实现非嵌套组件间通信，也可以实现跨级组件间通信

porps和state的区别？

props：一般用于父组件向子组件通信，在组件之间通信使用。

state： 一般用于组件内部的状态维护，更新组件内部的数据状态，更新子组件的props等。

在react如何阻止冒泡事件?

阻止冒泡事件分两种情况

1. 阻止合成事件间的冒泡，用e.stopPropagation();
2. 阻止js原生事件的冒泡，用e.nativeEvent.stopPropagation();

**补充：如何阻止事件冒泡、阻止默认动作？**

**阻止事件冒泡:**

e.stopPropagation(IE9+及其他浏览器),

window.e.cancelBubble = true(IE678)

**阻止默认事件:**

e.preventDefault()-->IE9+及其他浏览器

window.e.returnValue = false   IE678

**jQuery:**

     阻止事件冒泡:e.stopPropagation()

     阻止默认事件:epreventDefault()

     同时阻止事件冒泡和默认事件: return false

对react中事件机制的理解。

Reac事件是合成事件，不是原生事件。

合成事件和原生事件的区别：

1. 写法不同：合成事件是camal命名法，原生事件是全部小写
2. 绑定位置不同：合成事件全部委派到document上，原生事件绑定到真实dom上。所以一般是在componentDidMount中或者ref回调函数中绑定，在componentWillUnmount阶段进行解绑，防止内存泄漏。
3. 执行顺序不同：先执行原生事件，事件冒泡至document上，再执行合成事件。

react怎么从虚拟dom中拿出真实dom？

在react中如何获取真实dom？

通过refs获得 DOM 元素或者某个组件实例。

我们可以为元素添加 ref 属性，在回调函数中接收该DOM元素，该DOM元素会作为回调函数的第一个参数返回。

或者可以给ref传字符串。通过this.refs获取dom。

例如：<input ref=((input)=>{return this.name=input}) />， this.name.value取值  
或者 <input ref="name" />，this.refs.name取值。

为什么虚拟DOM可以提高性能？

当数据变化的时候，无需整体的重新渲染，而是局部刷新。虚拟DOM它是以js的形式存在，计算性能会比较好。而且由于减少对真实DOM的操作次数。性能会有很大提升。

diff算法原理 / 什么是diff算法（diff算法的理解）

* 对新旧两棵DOM树进行同层比较。
* 给每层元素添加唯一的 key 值。提高diff算法效率。
* React 只会匹配相同组件名字
* 调用组件setState 方法的时候, React 将它标记为 dirty.到每一个事件循环结束, React 检查所有标记 dirty 的组件 重新渲染.这样就实现了局部刷新的效果。

React 中 keys 的作用是什么？

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。此外，React 还需要借助 Key 值来判断元素与本地状态的关联关系，因此我们绝不可忽视转换函数中 Key 的重要性。

为什么建议传递给 setState 的参数是一个 callback 而不是一个对象

因为 this.props 和 this.state 的更新可能是异步的，不能依赖它们的值去计算下一个 state。

react的setState的原理及用法。说一下你对setState理解。

原理：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，再合并并更新。  
因此，setState有 异步，合并更新两个特性。

使用：1.最常见的用法就是传入一个对象。

2.因为setState是异步的，所以它还可以接收第二个参数，第二个参数它是一个回调函数

setState（）的第二个参数是什么？什么时候会被调用？有什么作用？

会在setState函数调用完成并且组件开始重渲染的时候被调用，我们可以用该函数来监听渲染是否完成。

router3到4有什么改变?

1. V4不再使用V3里的{props.children}

2. V4还同时支持渲染多个路由

3.多了一个限制未登录的用户访问某些路由功能，可以在应用程序顶端中设置一个主入口，区别登录和未登录UI展示界面。

react-router的实现原理以及他们的优缺点。

**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。

**通过地址栏地址的切换，来切换不同组件或同一组件的不同状态。**

优点：

1. 简单: 不需要手工维护路由state，使代码变得简单

2. 强大: 强大的路由管理机制。

体现在如下方面

路由配置: 可以通过组件、配置对象来进行路由的配置

路由切换: 可以通过<Link> history.push等进行路由的切换

路由加载: 可以同步加载，也可以异步加载，这样就可以实现按需加载

3.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

缺点：API不太稳定，在升级版本的时候需要进行代码改动。

HashRouter和BrowserRouter的区别。

HashRouter是基于location.hash来实现的，地址栏带有#

BrowserRouter是基于h5的api里面的history对象来实现的，地址栏不带有#。

路由跳转方法有哪些？或者说你是如何实现路由跳转的？

方法一：history.push('路由路径')

方法二：使用高阶组件

方法三：<Link/>标签

方法四：使用dva/router里面的routerRedux

方法五：PropTypes中的context上下文对象

方法六：使用库query-string

<Route>组件，有三种渲染方式:

(1)<Route component>：在地址匹配的时候React的组件才会被渲染，route props也会随着一起被渲染；

(2)<Route render>：这种方式对于内联渲染和包装组件却不引起意料之外的重新挂载特别方便；

(3)<Route children>：与render属性的工作方式基本一样，除了它是不管地址匹配与否都会被调用；

路由跳转传值：

1>react-router库，它提供了一个hashHistory对象

hashHistory.push({

pathname: 'message/detailMessage',

query: {

**title:value.title,**

**time:value.time,**

**text:value.text**

**},**

})

2><Route path="/news/:title/:content" component={News}></Route>

News组件接收：

this.props.match.params.title

this.props.match.params.content

3><Link to="/news?title=react全家桶&content=react-router4">新闻</Link>

News组件接收:

this.props.location.search.title (URLSearchParams对象处理search数据)

this.props.location.search. content

给你一个场景，问你如何去实现，例如：如何配置动态路由?

1.在react中，我们一般在路由表中，直接在路由的后面添加/：id 这样去实现动态路由

2.在umi框架中，我们一般直接文件名称是$开头，这样也可以自动生成动态路由

Redux实现原理以及特点（优缺点）

创建全局store之后，通过dispatch转发action，给相应的reducer函数，reducer函数做处理之后得到新的state，connect做链接后，这个组件的props发生变化，重新render。这是redux的原理。

特点：redux是数据流的解决方案，数据中心化管理，单向数据流，函数式编程。

但是当数据流庞大的时候，会需要我们维护大量的action function ,reducer,saga,常量等，使得redux数据中心化管理变得复杂，难以维护。

react-redux的实现原理（什么是react-redux?或者说你对react-redux的理解） React-redux和redux的区别

1.Redux作为一个通用模块，主要还是用来处理store中state的变化，通过react-redux的connect方法做连接，可以在React和Redux的项目中将两者结合的更好。

2.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：从最外部用context方式封装了整个应用，并向connect模块传递store。

Connect：

1、将state和action通过props的方式传入到使用connect做链接的组件内部。

2、监听store的变化，使得connect做链接的这个组件可以响应state变化

对redux中间件的理解，用过哪些中间件以及他们各自的作用？

理解：位于action和reducer之间,负责转发action函数,可以通过中间件实现异步操作。

**redux-logger** 打印前后store中state以及action对象 主要作用负责监控state

redux-promise-middleware 简化异步，不用维护大量action function

**redux-thunk** 它帮助我们统一了异步和同步action 的调用方式，把异步过程放在 action 里面去解决。  
缺点：

(1).异步请求的代码过于复杂，而且异步操作太分散，而redux-saga只要调用一个call方法就可以。

(2).action形式不统一，如果不一样的异步操作，就需要写多个thunk中间件。

优点：学习成本低

**redux-saga:**  
优点：

（1）通过Effects，集中处理了所有的异步操作。异步操作是可控的，可以随时取消相应的异步操作。最重要的是saga它是基于generator函数去实现的异步。

（2）action是普通对象，这跟redux同步的action是一样的({type:XXX})。

缺点：学习成本高。

说一下axios和fetch的区别。

Axios是基于原生js对ajax进行promise封装实现的。它的用法是axios().then().catch()

Fetch是基于jq对$ajax进行promise封装实现的。它的用法是fetch().then().then().catch()

而且fetch不能拦截mockjs。当然我们在工作中会经常用axios。因为axios的封装比fetch少一层。（可以说说axios特点）

什么是axios ？axios有哪些特性？

Axios是原生js中对ajax进行promise封装实现的。

它本身具有以下特征：

从浏览器中创建 XMLHttpRequest

从 node.js 发出 http 请求

支持 Promise API

拦截请求和响应

转换请求和响应

自动转换JSON数据

取消请求

xss攻击

1.跨站脚本攻击，它指得是恶意攻击者往web页面插入恶意的html代码或js脚本，当用户浏览的时候，嵌入的脚本就会被执行，从而窃取用户的信息.或者入侵到后台的数据库

2.解决方法：设置正则表达式

了解dva吗？(什么是dva？优缺点？)-------（看笔记吧）

dva是react的一个开发框架，脚手架都已经配好了，而且集成了react-router、react-redux、redux、redux-saga、redux-devtools调试工具、prop-types等等，简化数据流，支持热重载，提高开发效率，也容易于维护。

缺点：需要手动合并mock、手动合并model、手动配置路由。

umi框架可以解决dva中的缺点。（看笔记吧）

Roadhog是dva基于webpack的脚手架，配置文件接口有两个：

1. mock配置接口：**.roadhog.mock.js**
2. webpack配置接口：**.webpackrc.js**（js文件）

**.webpackrc** （json文件）

Dva有几个api?（你知道几个dva的api）

7个api

（1）创建dva实例：app.dva()

（2）注册dva插件：app.use()

（3）注册model：app.model()

（4）卸载model：app.unmodel()

（5）替换model：app.replacemodel()

（6）注册router：app.router()

（7）启动dva实例：app.start()

说一下你们公司的业务流程。

1. 后台写好接口推送到开发服务器。
2. 把接口给我们前端。
3. 这时候就可以前后端分离开发。
4. 前端在开发环境，通过proxy代理服务器，请求开发服务器接口。进行开发。
5. 我们开发完毕之后，将代码推送到开发服务器上。
6. 后台写好真实接口数据，也推送达到开发服务器上。
7. 之后，进行前后端联调。

Proxy代理服务器的跨域原理。

浏览器处于安全考虑是禁止跨域的，但是服务器端没有这个说法。

浏览器发送请求给webpack-dev-server，通过proxy代理，将真实接口转化为远程接口，并且发送给远程服务器，远程服务器再将数据发送给proxy代理，后发送给浏览器。

vue和react的区别

相同点：

中心思想相同：一切都是组件，组件实例之间可以嵌套。

都提供了钩子函数，可以让开发者按照自己的需求定制。

都没有内置AJAX，Route等 具有功能的核心包，而是以插件的方式加载。

不同点：

1、react严格上针对的是mvc模式的view层，vue则是mvvm模式。

2、操作dom的方式不同，vue使用的是v指令操作dom，react是通过js进行操作。

3、vue可以通过v-model实现双向数据绑定。

4、react中state是不能直接改变的，需要使用setState改变。

vue中的state不是必须的，数据主要是在data属性中管理。

react中的高阶函数

高阶函数就是一个纯js且没有副作用的函数。  
高阶组件就是一个函数，且该函数接受一个组件作为参数，并返回一个新的组件。

**Hooks（为什么使用HOOKS？为什么会有Hooks?）**

解决了使用无状态组件（Function）还是有状态组件（Class）问题

解决了使用哪个生命周期函数问题

解决了组件中的this指向，晕头转向的问题

注意：hooks函数不要使用在条件语句，这些逻辑语句里面。有些情况可能会改变hooks的有无，这样会报错。因为每一个hook函数是按顺序赋值执行的。

例如：hook函数—useState()函数—将无状态组价可以转化成有状态的组件，在无状态组件里面轻松的去改变state值。

const [第一个参数，第二个参数]=useState(初始值)

第一个参数：数值、对象、数组等等

第二个参数：对第一个参数做处理的功能函

useEffect 相当于componentDidMount和componentDidUpdate

受控组件和不受控组件的区别

受控组件

受setState()的控制

不受控组件

表单数据由DOM本身处理。即不受setState()的控制。

什么是副作用effects

本来吃这个药💊，我只是想治个感冒而已，结果感冒虽然治好了，但是却过敏了。这个是副作用。

本来我只是想渲染DOM而已，可以当DOM渲染完成之后，我还要执行一段别的逻辑。这个就是副作用。

在React中，如果利用得好，副作用可以帮助我们达到更多目的，应对更为复杂的场景。

react 组件的划分业务组件和技术组件？

根据组件的职责通常把组件分为 UI 组件和容器组件。

UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑。

两者通过 React-Redux 提供 connect 方法联系起来。

**React项目负责哪些模块？做过哪些功能？遇到过什么问题？**

最近着手的react项目，它是一个后台管理系统。功能这块是通过使用ant design这个库，实现登录，注册，管理员的添加与删除，还负责自动时间和天气功能这个两个模块，模拟网络延迟，前后台数据交互。这些功能。

**（看单车笔记）**

webpack基础配置项有哪些

1. entry设置入口文件
2. output设置出口文件
3. module主要是用于打包文件。包含各种loader。Css-loader/less-loader/style-loader/babel-loader
4. plugins设置插件 比如html-webpack-plugin、clean-webpack-plugin

CommonsChunkPlugin 公用模块独立打包

1. resolve用来配置webpack-dev-server服务器

什么是webpack? webpack的作用。

webpack可以看做是一个模块打包机，它做的事情是分析项目结构，找到js模块以及其他一些浏览器不能识别的拓展语言，并将其打包为合适的格式供浏览器使用。

webpack如何提取公共代码

提取公共代码，简单来说，就是减少代码冗余，提高加载速度。

这里要用到两个插件：

* commonChunkPlugin
* webpack.optimize.commonsChunkPlugin

Webpack热加载怎么实现（拼凑纸张上面的笔记）

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

webpack打包最基本的实现方式 是将所有的模块代码放到一个数组里 通过数组ID来引用不同的模块、对吗

webpack中每个模块有一个唯一的id，是从0开始递增的。参数则为一个数组。数组的每一项都为一个function。function的内容则为每个模块的内容，并按照require的顺序排列。webpack**有哪些常见的Loader？他们是解决什么问题的？**

* file-loader：把文件输出到一个文件夹中，在代码中通过相对 URL 去引用输出的文件
* url-loader：和 file-loader 类似，但是能在文件很小的情况下以 base64 的方式把文件内容注入到代码中去
* image-loader：加载并且压缩图片文件
* babel-loader：把 ES6 转换成 ES5，jsx转换成js
* css-loader：加载 CSS，支持模块化、压缩、文件导入等特性
* style-loader：把 CSS 代码注入到 JavaScript 中，通过 DOM 操作去加载 CSS。
* eslint-loader：通过 ESLint 检查 JavaScript 代码

webpack有哪些常见的Plugin（插件）？他们是解决什么问题的？

* define-plugin：定义环境变量
* html-webpack-plugin：将src目录的html打包到生产目录下。
* clean-webpack-plugin:在打包前将dist目录下的文件，先清除一下。commonChunkPlugin：提取公共代码。

Loader和Plugin的不同？

**不同的作用**

**Loader**它是"加载器"。Webpack将一切文件视为模块，但是webpack是只能解析js文件，如果想将其他文件也打包的话，就会用到loader。 所以Loader的作用是让webpack拥有了加载和解析*非JavaScript文件*的能力。

**Plugin**它是"插件"。Plugin可以扩展webpack的功能，让webpack具有更多的灵活性。

**不同的用法**

* **Loader**在module.rules中去配置，类型是数组，每一项都是一个Object。
* **Plugin**在plugins中单独配置.类型也是数组，每一项是一个plugin的实例。

如何利用webpack来优化前端性能？（提高性能和体验）

用webpack优化前端性能是指优化webpack的输出结果，让打包的最终结果在浏览器中，运行快速而且高效。

* 压缩代码。可以利用webpack的plugin插件来压缩JS文件， 利用css-loader来压缩css文件
* 利用CDN加速。
* 通过commonChunkPlugin插件提取公共代码。减少冗余代码。

如何提高webpack的构建速度？

将所有第三方库打包成一个library库，然后去引用它，webpack打包业务代码的时候，就不在处理第三方库，从而提高构建速度。

简述一下es6中的箭头函数和普通函数有什么区别？

箭头函数省略function关键字，而且自带bind（this）指向

this指向永远是其上下文this，任何方法都改变不了this指向

function是可以定义构造函数的，而箭头函数是不可以的

箭头函数没有原型属性（没有\_\_proto\_\_和prototype）

箭头函数不能当做Generator函数,不能使用yield关键字

let、var和const的区别？

相同点：三者都是用来创建变量的

不同点：

var的作用域是函数作用域 变量的默认值都是 undefined

let的作用域是块级作用域 变量不能重复声明

const是声明常量 常量一旦被声明则不能改变

call、bind方法是否可以改变箭头函数的this指向？

不可以，箭头函数自带bind(this)

es6中proxy拦截器，如何使用？以及应用场景

new Proxy拦截器，第一个参数是对谁进行一个拦截，第二个参数是对象，包含两个方法，get方法和set方法，get方法是在对象属性显示之前触发，set方法是在对象属性改变之前触发。

应用场景：比如用户名，手机号，银号卡号等等，这些隐私信息，需要我们拦截之后，进行处理。把一些信息转化为\*\*\*

let pro=new Proxy({

name:"tom",

age:15,

sex:"男"

},{

get(target,key){ //对象属性显示之前触发get方法 target当前监控对象

console.log("你触发了get方法");

return "机密"+target[key];

},

set(target,key,value){ //对象属性改变之前触发set方法

console.log("你触发了set方法");

target[key]=value+"\*\*\*";

}

})

Es6中的symbol

特点：值是独一无二的。

不能隐式转换。

不能与其他数据类型做运。

不能使用点运算符进行操作。

let obj={};

obj[Symbol("a")]="tom"; //symbol数据类型，表示独一无二的值，即使值一样也代表两个值

obj[Symbol("a")]="jarry";

obj[Symbol("a")]="susan";

//**因为键名相同无法遍历 提供了Object.getOwnPropertySymbols(对象)专门遍历symbol数据**

for(let value of Object.getOwnPropertySymbols(obj)){

console.log(value);

}

**请描述一下同步和异步的区别？**

同步：脚本会停留并且等待服务器回应再继续操作，在等待服务器处理的过程中客户端浏览器一直处于等待状态，不能进行其他操作。

异步：脚本允许页面进行其他进程并处理可能的回复。

**什么是promise？以及promise怎么使用？（用promise做过封装吗？这个问题看笔记）**

Promise它是es6提供的api，它是一个对象，有三个状态，pedding进行中，resolve已完成，reject已失败。状态一旦改变则不可逆转。

Promise的构造函数会接受一个函数作为参数，这个函数参数有两个参数，分别是resolve和reject，他们的作用是将promise的pedding状态转化为resolved或者rejected，并将成功或者失败的返回值传递出去。成功了调用then方法，失败了调用catch方法。

**什么是generator函数？**

Generator函数他是一个状态机，内部封装了很多状态，每个状态都是一个yield。当调用generator函数的时候，会返回一个遍历器对象，他有一个接口 是next方法。我们通过next方法就可以遍历每一个状态。

Next方法的运行逻辑：遇到yield，就会暂停后面的语句，将yield后面的表达式或者返回值赋值给遍历器对象的value值。当再次调用next方法的时候，接着执行下面的语句，直到遇到yield语句，暂停。如果没有遇到yield，则到函数执行完毕，将函数的返回值赋值给遍历器对象的value值。没有返回值，value就是undefined。

**什么是地狱回调？地狱回调的缺点。以及如何解决回调地狱？**

函数作为参数层层嵌套，一个函数作为参数需要依赖另一个函数的执行调用，我们称它是回调地狱。

不利于我们对项目的维护和阅读。

Es6中的promise和generator函数都可以解决回调

Es7中的await/async

**import和require区别**

import是es6的引入模块方法，require是es5的引入模块语法

import是nodeJS的一种语法， require是AMD规范的框架

import呢它的前面不允许有任何除了import以外的语法，require则是可以的。

**set和map的区别：**

相同点：set和map都是es6提供的新的数据结构。

不同点：set是一个构造函数、用来生成 Set 数据结构。

有add()方法和delete()方法。返回一个布尔值、表示参数是否为Set的成员。用这个方法可以实现数组去重。

clear()方法用来清除所有成员。

Map类似于对象、也是键值对的集合，但是“键”的范围不限于字符串，各种类型的值都可以 当作键（包括Object）。

add()、delete()、has()和clear()

**es6提供了哪些新的api?**

类、继承、链式写法、隐式迭代。

Let和const新声明变量的方法。

新的数据结构set和map

新的数据类型sylmol

Promise对象

Generetor函数

解构

补充：

2019-5-22蚂蚁金服面试(1)

用哪个框架多一些 用了多久 redux是否用过 $是什么意思

（1）redux缺点

答：1.redux单项数据流维护困难

2.需要维护大量的action、constant、reducer

3.组件每次连接都需要加载store里的state。

（2）react项目 阻止组件渲染

主要就是通过父组件传值进行条件判断，是否进行渲染。

这个网址有实例：https://blog.csdn.net/gwdgwd123/article/details/81987054

（3）react性能优化

一、渲染角度

1）渲染角度优化:render 里面尽量减少新建变量和改变函数的指向（通过bind）

2）style={{color:'#f00'}} 应该用style1={color:'#f00'} style={style1},这样可以防止再每一次渲染的时候都重新创建一个对象。

3）key要唯一

二、reselect（数据获取时优化）

react和redux都是通过数据驱动的的方式驱动渲染过程，那么处理优化渲染过程，获取数据的过程也是需要考虑的一个优化点。

数据量大的时候会降低性能。

可以用reselect，它的动作原理：只要相关的状态没有改变，那么就直接使用上一次的缓存结果。

https://blog.csdn.net/weixin\_41902031/article/details/80353134

（4）http请求过程 三次握手

第一次握手：客户端向服务器发送SYN报文，并发送客户端初始序列号Seq=X；等待服务器确认，

　　第二次握手：服务器接收客户端的SYN报文，然后向客户端返回一个包SYN+ACK响应报文，并发送初始序列号Seq=Y

　　第三次握手：客户端接受SYN+ACK报文，并向服务器发送一个ACK确认报文，至此连接建立

　　【建立连接的最重要目是让连接的双方交换初始序号(ISN, Initial Sequence Number)，所以再响应的ACK报文中会包含序列号递增序列】

（5）ES6相比es5新增什么

1.let const 变量命名方式 2. 箭头函数 3.类和继承 4.数组遍历 5.解构 6.拓展运算符 7.模板字符串

8.数组遍历 9.新增数据结构（set，map）10.Generator函数 11.promise异步对象 12.proxy拦截器

13.新增数据类型 symbol

（6）ant design 表单验证 如果有大量数据要跳转到另一页面如何实现

大家看一下：自己再网上找找

antd里的form表单封装思路

1.通过Form.create这个方法取包裹表单就可以实现数据双向绑定 组件中可以通过

getFieldDecorator方法与封装的组件进行数据双向绑定 它的参数是以数组形式获得

2.在封装的组件中用props进行接收其他组件传递过来的内容

3.要在组件中创建一个初始值，每个表单的内容和type值和表单中输入的内容的值

4.在封装的组件中，要注意数据传过来的时候是否有内容，需要进行判断一下,一般传递的都是数组形式，在封装的组件中进行遍历

5.在封装好的组件中进行判断type值是否等于需要的数据，如果是就返回一个form表单

6.单独封装一个方法 去接收传递的value值，让这个方法去遍历，push进去遍历出来的内容，通过在封装的组件中进行调用这个方法 把组件中的全部value通过参数传递给封装的方法

7.组件中的就可以直接使用Form.create包裹后输出的标签就可以通过属性传递的方法传递给封装的组件

8.也可以通过属性传递的方法把当前组件的函数传递给封装好的组件

9.获取输入的值可以通过this.props.form.getFieldValue去获得

10.resetFields重置表单中的内容

ETable封装基本思路：

1.首先我们要通过ajax去获取后台的数据，然后刷新给表格

2.一般我们要找基础表格和高级表格的相似之处，把这些相似的地方进行统一封装

3.通过bordered这个可以让表格带边框与不带边框

4.通过属性把数据源传递给封装好的组件，使用dataSource方法

5.通过属性把判断是否有分页也传递给封装的组件，通过pagination方法，可以定义个false和true传递过去

6.通过属性把是否让每行内容是否可以进行单选或复选传递给封装组件，通过rowSelect方法判断 布尔值

7.在封装的组件中 直接通过...在ETable的属性中解构出来就可以了

8.在封装的组件中，我们要通过this.props去获得rowSelection传递过来的值来判断是否要开启单选或复选

9.如果要使用单选或复选，那么我们要定义一个初始的对象，在这个值中，我们去判断是单选还是复选，判断点击的是每行的内容 selectedRowKeys 这个方法，这个方法要配合onChange方法配合使用

10.然后要通过判断，来确定你调用这个封装的组件的组件里面是否要使用单选或复选，并且获得组件中是否要用到单选或复选

react hooks 16.7 新增语法

yarn add react@next react-dom@next

--统一组件写法

--简化写法，减少代码量，加快开发速度

--合并生命周期方法

const [ count , setCount ] = useState ( 5 )

async await

简述一下async和await？

1.async 表示这是一个 async 函数，而 await 只能在这个函数里面使用。

2.await 表示在这里等待 await 后面的操作执行完毕，再执行下一句代码。

3.await 后面紧跟着的最好是一个耗时的操作或者是一个异步操作(当然非耗时的操作也可以的， 但是就失去意义了)。

4.async代替了generator函数 用同步的方法去写异步操作

5. 其实在使用 async/await 的时候最简单的场景就是当你需要执行一个耗时操作时

或者必须为异步操作时就可以直接使用 async 来执行你的函数，

在这个函数内部 使用 await 关键字来达到异步执行的最终目的：

执行完毕（异步执行）了，可以运行下一行代码了！

ES7/8/9新增api

Array.prototype.includes()方法接受两个参数

----- 要搜索的值 搜素的开始索引

-----includes()方法用来判断一个数组是否包含一个指定的值，根据情况，如果包含则返回true,否则返回false

求幂运算符（ \*\* ）

4\*\*3 64

Math。pow(4,3)

Async/Await,用于处理js异步操作的语法糖，帮助我们摆脱回调地狱，编写更加优雅的代码

React.txt

es6新增

箭头函数、generator函数、解构、模板语法、Map、fillter，set, promise, let , const, class类 extends继承

阻止冒泡

js原生中：

[IE9+]：event.stopPropagation()

[JQ]：return false 会同时阻止默认行为和冒泡事件(在原生js中只会阻止默认事件)

在React中：

合成事件: event.stopPropagation()

阻止js原生事件冒泡：e.nativeEvent.stopPropagation()

state对内， props对外

this.setState

特点： 异步 合并更新

原理：为了合并setState，我们需要一个队列来保存每次setState的数据，然后在一段时间后执行合并操作和更新state，并清空这个队列，然后渲染组件。

执行：当state改变的时候去调用第二个参数，执行回调函数

如何防止react直接渲染

diff算法里面的key值 钩子函数 shouldComponentUpdata

map和forEach的区别

map是有返回值的 forEach是没有返回值的

受控类型组件：

A）非受控组件

定义：该组件内部的状态不受state控制；

B）受控组件

定义：组件的状态变化受到state的控制；

简述react

首先它是mvc框架

他就是一个JS UI库，能够和市面上的包结合在一块使用，有很好的兼容性

数据流是单向的，只能父给子传值

react生命周期

初始化阶段

constructor () 初始化state

componentWillMount

render es6转es5 jsx转js

componentDidMount

更新阶段

componentWillReceiveProps

shouldComponentUpdate

componentWillUpdate

render

componentDidUpdate

卸载阶段2

componentWillUnMount

虚拟DOM

首先它存在内存中，刷新只会在内存中刷新

其次采用DIff算法，从而减少了全局刷新，实现了只刷新更改区域

diff算法

1.把树形结构按照层级分解，只比较同级元素

2.给列表结构的每个单元添加唯一的key值，方便比较。

3.react只会匹配相同的class的Component（class指得是class组件的名字）

4.合并操作，调用component和setState方法的时候，React将其标记为dirty，到每一个事件循环结束，react都会检查所有标记的dirty的component重新重绘

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

redux

单向数据流

函数式编程

state只读

集中式管理

react-redux 和 redux 区别

react-redux相当于连接react和redux的桥梁，他利用高阶组件connect将react和redux连接起来，使得开发方便

redux缺点？

要维护大量的Action Contants Function

当父组件的Props值发生改变的时候，即使子组件的state没有发生改变，依旧会重新渲染

例如设置一个点击事件，当点击的时候发送dispatch（action），然后去找action，

再转发action去找对应的reducer，更改state去覆盖store里面的state，哪个组件

需要就通过connect去连接store，通过mapStateToProps方法传递给props，直接

解构出来就可以了

高阶组件

高阶组件是一个函数，能够接受一个组件并返回一个新的组件。

高阶组件通过呗包裹的React组件来操作props

写一个函数，传入形参是个组件，返回这个组件，并且给他添加新的属性

promise

1.promise是es6新增的API 是一个异步对象 用来解决异步操作

2.promise有三个状态，pending，resolved，rejected

3.进入其中某个状态，就不能再更改

4.有两个方法， .then方法 成功回调 .catch方法 错误回调

5.多条数据回来的时候，可以通过 race方法，他有三个参数，每个参数都是一个函数，谁最先回来，以谁为准

.all方法 有三个参数，每个参数都是函数，谁最后回来 以谁为准

.reace方法 谁快谁输出

回调地狱简述 如何解决？

简述: 下一步的内容需要等到上一步的内容执行完毕

解决方法：使用generator函数进行封装

generator

1.generator是一个状态机，每条yield语句都是一个状态

2.用同步的方法去写异步操纵

3.generator会返回一个遍历器对象，遍历器对象都有一个next方法，可以对yield语句进行

遍历，每次调用一次next方法，都会返回一个对象{value:值，done：false}

直到yield语句执行完毕或碰上 return 会返回{value：undefined，done：true}

react-router

1.hashRouter 是基于location.hash来实现的 url地址会出现#号 对用户不友好

一般开发用这个，上线之前换成browserRouter

2.browserRouter 是基于H5的history来实现的 url地址不会出现#号，对用户友好

3.withRouter 高阶组件 也叫高阶路由 自带三个对象 history location match

一般用于接收不到这三个对象的组件，通过高阶路由包裹可以获得这三个对象

实现路由跳转 传值

4.动态路由 就是通过路由传值 一般通过Route里面的path的路径后面添加要传递

的值，可以在location里面的search获取我们传递的内容，也可以通过Link的标签

的to的地址后面加？id这样传递 也是通过location里面的search获取

如果只想匹配一个路由，可以使用Switch，在 <Switch> 中只有一个 <Route> 会被渲染， 同时 可以再在每个路由添加exact，做到精准匹配

　　 Redirect，浏览器重定向，当多有都不匹配的时候，进行匹配

中间件

1.中间件的作用：负责拦截action，然后去实现一些操作，通过中间件自己

的方法next转发action

2.next方法的作用：如果只有一个中间件，就会直接转发给

reducer，如果还有下一个中间件，就转发给下一个中间件，然后再转发给reducer

3.run方法的作用：自动调用next方法并且执行

常用中间件

（1）redux-logger

--负责拦截action 并且转发action

--负责打印我们之前的state和之后state

（2）redux-thunk

--负责拦截action 并且转发action

--让我们action可以执行异步操作

（3）redux-promise

--负责拦截action 并且转发action

--简化thunk的写法 也是让我们的action可以执行异步操作

（4）redux-saga

--generator函数是一个状态机，内部有许多状态，每条yield语句都是一个状态

--根据generator函数来进行异步操作，用同步的写法去写异步代码

--generator函数有一个run方法，他会递归遍历yield语句 当遍历完成或碰上return遍历结束

--generator函数会返回一个遍历器对象 遍历器对象都有一个next方法，当你调用next方法会遍历一次yield语句，并且返回一个对象，对象里面有个done来判断是否遍历完成，变为true表示遍历完成 或碰上return就会返回true

--当next方法带参数，参数会给上一条yield语句的返回值赋值

（5）redux-saga/effects 也叫saga的附件或副作用

--takeEvery的作用：两个参数，第一个是接收的假参数，第二个是你要执行的generator函数

--takeLatest作用：防止重复提交，当点击多次 只会执行最后一次

--put的作用：转发action

--call的作用：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

react组件间传值

1. 父->子 通过props

子->父 子组件通过调用父组件的里面的函数来进行传值

2.redux进行组件间通信

3.context全局对象的方法进行跨级传值

context全局对象存在于propType中这是一个格式校验，一般想要使用context全局对象就必须要在这个组件中使用格式校验才可以获得context全局对象

4.事件订阅 哪个组件需要进行跨级通信 就让这两个组件进行订阅

dva

1.dva是一个开源的框架，它的脚手架是roadhog脚手架，roadhog是基于af-webpack搭建的脚手架

2.它集成了react-router、redux和react-redux、redux-saga和redux的监控redux-devtools 、

3.在dva中如果想要配置webpack就通过dva中自带的.webpackrc来进行配置

4.在dva中 所有的mock最终都要合并到.roadhogrc.mock.js中

5.dva的特点

--易学易用 等同于reducer 订阅，会触发

--elm概念 通过reducers、effects、subscriptions来组织model

--插件机制 做异步操作用的

--支持hmr热加载 基于babel-plugin-dva-hmr 来实现

6.dva的优点

--简化了数据流

--方便维护

--提高了开发体验和速度

--支持跨度代理

--支持mock.js

--自带mock

对roadhog的了解

是dva基于webpack的脚手架 ,配置文件接口有两个：

1 mock的配置接口: roadhog.mock.js

2 webpack配置文件接口

.webpackrc 配置方式为 json

.webpackrc.js 配置方式为 js

Dva有几个api?（你知道几个dva的api）

7个api

（1）创建dva实例：app.dva()

（2）注册dva插件：app.use()

（3）注册model：app.model()

（4）卸载model：app.unmodel()

（5）替换model：app.replacemodel()

（6）注册router：app.router()

（7）启动dva实例：app.start()

umi框架

1.umi是一个企业级的可插拔的开源框架

2.umi中集成了dva umi中的数据流管理通过dva来进行管理

3.umi的优点

--插件化

--开箱即用

--约定式路由

--可以兼容到IE9

--完善的ts支持

--高性能

简述umi 约定式

我按照你的规则写东西，那么就会自动生成

遍历器iterator

1.iterator接口就是为了所有数据类型(Array Object Map Set)提供了一种统一的访问机制,

使用for..of会自动去寻找这个接口

2.ES6规定，默认的iterator接口部署在数据结构的symbol.iterator属性，

也就是说一个数据结构只有具有（iterator）属性就可以用for...of和扩展运算符...

3.使用场景：for...of、Array.from()、 Map()、 Set()、 WeakMap()、 WeakSet()、 Promise.all()、 Promise.race()

这些方法都会自动调用这个接口

webpack基础配置项有哪些

1》entry设置入口文件

2》output设置出口文件

3》module主要是用于打包文件。包含各种loader。Css-loader/less-loader/style-loader/babel-loader

4》plugins设置插件 比如html-webpack-plugin、clean-webpack-plugin

CommonsChunkPlugin 公用模块独立打包

5》resolve用来配置webpack-dev-server服务器

webpack打包最基本的实现方式 是将所有的模块代码放到一个数组里 通过数组ID来引用不同的模块、对吗

webpack中每个模块有一个唯一的id，是从0开始递增的。参数则为一个数组。数组的每一项都为一个function。function的内容则为每个模块的内容，并按照require的顺序排列

webpack常用插件

1.htmlWebpackPlugin 复制模板页

2.cleanWebpackPlugin 清除打包文件夹 并且从新创建一个文件夹

3.webpack-dev-server 热重载

4.ProvidePlugin 可以在任何地方自动加载模块而不需要import 或 require 方法

5.CommonsChunkPlugin 是一个可以用来提取公共模块的插件

6.ExtractTextWebpackPlugin 分离 CSS

7.UglifyJsPlugin 代码压缩输出

fetch 做前后端交互 可以拦截 mockjs吗？不可以 promise基于JQ里的ajax进行的封装

axios 对原生JS里的AJAX进行封装

说一下axios和fetch的区别。

Axios是基于原生js对ajax进行promise封装实现的。它的用法是axios().then().catch()

Fetch是基于jq对$ajax进行promise封装实现的。它的用法是fetch().then().then().catch()

而且fetch不能拦截mockjs。当然我们在工作中会经常用axios。因为axios的封装比fetch少一层。（可以说说axios特点）

什么是axios ？axios有哪些特性？

Axios是原生js中对ajax进行promise封装实现的。

它本身具有以下特征：

从浏览器中创建 XMLHttpRequest

从 node.js 发出 http 请求

支持 Promise API

拦截请求和响应

转换请求和响应

自动转换JSON数据

取消请求

import和require区别

import是es6的引入模块方法，require是es5的引入模块语法

import是nodeJS的一种语法， require是AMD规范的框架

import呢它的前面不允许有任何除了import以外的语法，require则是可以的。

Es6中的symbol

特点：值是独一无二的。

不能隐式转换。

不能与其他数据类型做运。

不能使用点运算符进行操作。

let obj={};

obj[Symbol("a")]="tom"; //symbol数据类型，表示独一无二的值，即使值一样也代表两个值

obj[Symbol("a")]="jarry";

obj[Symbol("a")]="susan";

//因为键名相同无法遍历 提供了Object.getOwnPropertySymbols(对象)专门遍历symbol数据

for(let value of Object.getOwnPropertySymbols(obj)){

console.log(value);

}

ts

.typescript

TypeScript特点:

预编译 tsx/ts?js

有类型的script

面向对象 枚举 接口 访问修饰符

typescript 预编译语言:javascript语法变成类似于Java或C#面向对象编程企业级语言

react新增.txt

废除的生命周期

componentWillMount

componentWillRecieveProps

componentWillUpdate

新增

static getDerivedStateFromProps( nextProps, prevState ) { }

用于替换 compoentWillReceiveProps 可以用来控制 props 更新 state 的过程；它返回一个对象表示新的 state；如果不需要更新，返回 null 即可

在每次渲染之前都会调用，不管初始挂载还是后面的更新都会调用，这一点和componentWillReceiveProps不同（只有当父组件造成重新渲染时才调用

　　简单的理解就说从props中获取state，这个生命周期的功能实际上就是将传入的props映射到state上面

　　getDerivedStateFromProps是一个静态函数，也就是这个函数不能通过this访问到class的属性，也并不推荐直接访问属性。而是应该通过参数提供的nextProps以及prevState来进行判断，根据新传入的props来映射到state

　　如果props传入的内容不需要影响到你的state，那么就需要返回一个null，这个返回值是必须的，所以尽量将其写到函数的末尾

getSnapshotBeforeUpdate() {}

　在最近的更改被提交到DOM元素前，使得组件可以在更改之前获得当前值，此生命周期返回的任意值都会传给componentDidUpdate()。

　　用于替换 componentWillUpdate，该函数会在update后 DOM 更新前被调用，用于读取最新的 DOM 数据，返回值将作为 componentDidUpdate 的第三个参数

　　在最新的渲染数据提交给DOM前会立即调用，它让你在组件的数据可能要改变之前获取他们

场景题.txt

给你一个场景，问你如何去实现，例如：如何配置动态路由?

1.在react中，我们一般在路由表中，直接在路由的后面添加/：id 这样去实现动态路由

2.在umi框架中，我们一般直接文件名称是$开头，这样也可以自动生成动态路由

当开发一个项目的时候，多个团队合作，在不通过版本控制工具的情况下，如何合并，并且能正常访问？

1.首先设置一个强制开发规范，每个文件应该取什么名字，提前设置好；

2.把每个团队的代码和接口，静态资源拷贝到同一个工程目录下

3.通过路由组件进行加载，这样就可以通过路由进行正常访问

例如我们封装了一个组件<Header/>

那么我们在组件中使用这个组件 像这样 <Header><p>你好啊</p></Header>

如何让你好啊 在其他组件中也显示

vue通过插槽

react一般通过属性进行传递