1. REACT的运行原理、工作原理？

REACT快速的原因之一是：REACT很少直接操作DOM、浏览器事件，太多的浏览器事件，会占用很大内存。

REACT基本原理是：所有在JSX声明的事件都会被委托在顶层document节点上，并根据事件名和组件名存储回调函数(listenerBank)。每次当某个组件触发事件时，在document节点上绑定的监听函数(dispatchEvent)就会找到这个组件和它的所有父组件(ancestors)，对每个组件创建对应React合成事件(SyntheticEvent)并批量处理，从而根据事件名和组件名调用回调函数。

在web开发中，我们总需要将变化的数据实时反应到UI上，这是就需要对DOM进行操作。而复杂或者频繁的DOM操作通常是性能瓶颈产生的原因（如何进行高性能的复杂DOM操作通常是衡量一个前端开发人员技能的重要指标）。REACT为此引入了虚拟DOM（Virtual DOM）的机制：在浏览器端用Javascript实现了一套DOM API。基于React进行开发时所有的DOM构造都是通过虚拟DOM进行，每当数据变化时，React都会重新构造整个DOM树，然后React将当前整个DOM树和上一次的DOM树进行对比，得到DOM结构的区别，然后仅仅将需要变化的部分进行实际的浏览器DOM更新。而且React能够批量处理虚拟DOM的刷新，在一个事件循环（Event Loop）内的2次数据变化会被合并，例如你连续地先将节点内容从A变成B，然后又从B变成A，React会认为UI不发生任何变化，而如果通过手动控制，这种逻辑通常是极其复杂的。尽管每一次都需要构造完整的虚拟DOM树，但是因为虚拟DOM是内存数据，性能是极高的，而对实际DOM进行操作的仅仅是Diff部分，因为能达到提高性能的目的。这样，在保证性能的同时，开发者将不再需要关注某个数据的变化如何更新到一个或多个具体的DOM元素，而只需要关心在任意一个数据状态下，整个界面是如何Render的。

1. REACT组件应该具备如下特征：
2. 可组合：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有它创建的子组件，通过这个特性，一个复杂的UI可以拆分成多个简单的UI组件。
3. 可重用：每个组件都是具有独立功能的，它可以被使用在多个UI场景。
4. 可维护：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护。
5. REACT：组件的生命周期

在组件的整个生命周期中，随着该组件的props或者state的变化，其DOM表现也会有相应的变化。一个组件就是一个状态机，对于特定的输入，它总返回一致的输出。

一个REACT组件的生命周期分为3个部分：实例化、存在期、销毁时。（装载、更新、卸载）

1. 实例化

当组件在客户端被实例化，第一次被创建时，以下方法依次被调用：

1. getDefaultProps
2. getInitialState
3. componentWillMount
4. render
5. componentDidMount

当组件在服务端被实例化，第一次被创建时，以下方法依次被调用：

1. getDefaultProps
2. getInitialState
3. componentWillMount
4. render

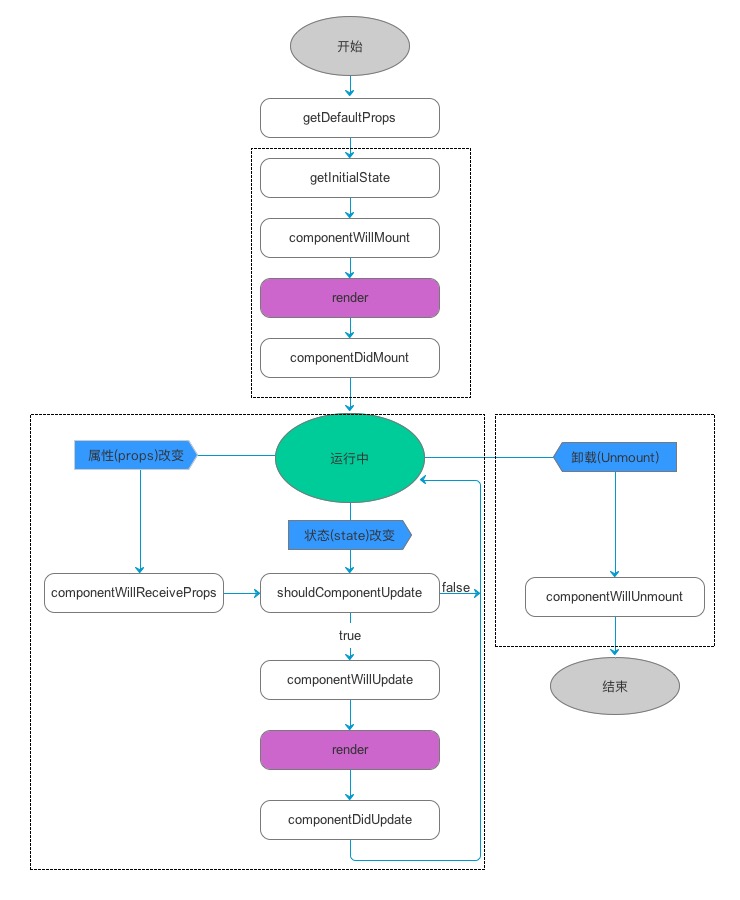
componentDidMount不会在服务端被渲染的过程中调用。

1. 存在期

此时组件已经渲染好并且用户可以与它进行交互，比如鼠标点击、手指点按，或者其它的一些事件，导致应用状态的改变，你将会看到下面的方法依次被调用：

1. componentWillReceiveProps
2. shouldComponentUpdate
3. componentWillUpdate
4. render
5. componentDidUpdate
6. 销毁时

componentWillUnmount



* **初始化**

1. getDefaultProps()

设置默认的props，也可以用defaultProps设置组件的默认属性

1. getInitialState()

在使用ES6的class语法时是没有这个钩子函数的，可以直接在constructor中定义this.state。此时可以访问this.props。

1. componentWillMount()

组件初始化时调用，以后组件更新不调用，整个生命周期只调用1次。

1. render()

react最重要的步骤，创建虚拟dom，进行diff算法，更新dom树都在此进行。此时就不能更改state了。

1. componentDidMount()

组件渲染之后调用，只调用1次。

* **更新**

1. componentWillReceiveProps(nextProps)

组件初始化时不调用，组件接受新的props时调用

1. shouldComponentUpdate(nextProps,nextState)

react性能优化非常重要的一环。组件接受新的state或者props时调用，我们可以设置在此对比前后2个props和state是否相同，如果相同则返回false阻止更新，因为相同的属性状态一定会生成相同的dom树，这样就不需要创造新的dom树和旧的dom树进行diff算法对比，节省大量性能，尤其是在dom结构复杂的时候。

1. componentWillUpdate(nextProps,nextState)

组件初始化时不调用，只有在组件将要更新时才调用，此时可以修改state

1. render()

组件渲染

1. componentDidUpdate()

组件初始化时不调用，组件更新完成后调用，此时可以获取dom节点。

* 卸载

1. componentWillUnmount

组件将要卸载时调用，一些事件监听和定时器需要在此时清除。

constructor()---构造方法

在React中constructor表示父类的构造方法，用来新建父类的this对象，这是ES6对类的默认方法，该方法是类中必须有的，如果没有显示定义，则会默认添加空的constructor()方法。

super()---继承

在class方法中，继承是使用extends关键字来实现的。子类必须在constructor()调用super()方法，否则新建实例时会报错。

报错的原因：子类没有自己的this对象，它只能继承父类的this对象，然后对其进行加工，而super()就是将父类中的this对象继承给子类的。没有super，子类就得不到this对象。

如果用了constructor就必须写super()，是用来初始化this的，可以绑定事件到this上；

如果在constructor中要使用this.props，就必须给super加参数：super(props)；

无论有没有constructor，在render中this.props都是可以使用的，这是REACT自动附带的；

如果没用到constructor，是可以不写的；REACT会默认添加一个空的constructor。

cookie和localStorage、sessionStorage的最大区别：

共同点：都保存在浏览器，且是同源的

不同点：cookie在浏览器和服务器间来回传递，sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。

**cookie**：在浏览器和服务器间来回传递；cookie数据不能超过4K；因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。**数据有效期**：只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。**作用域**：在所有同源窗口中都是共享的。

**sessionStorage**：存储数据可以达到5M或者更大；**数据有效期**：仅在当前浏览器窗口关闭前有效。**作用域**：不在不同的浏览器窗口中共享，即使是同一个页面。

**localStorage**：存储数据可以达到5M或者更大；**数据有效期**：始终有效，窗口或者浏览器关闭也一直保存。**作用域**：在所有同源窗口中共享。

一个div在另一个div内水平垂直居中的多种方法：

1. position

父级元素设置：position:relative;

子级元素设置：position:absolute;margin:auto;top:0;right:0;bottom:0;left:0;

1. position transform元素未知宽度

父级元素设置：position:relative;

子级元素设置：position:absolute;left:50%;top:50%;transform:translate(-50%,-50%);

1. flex布局

父级元素设置：display:flex;justify-content:center;（使子项目水平居中）align-items:center; （使子项目垂直居中）

1. grid布局

父级元素设置：display:grid;justify-content:center;（使子项目水平居中）align-items:center;（使子项目垂直居中）

1. table-cell布局

父级元素设置：display:table-cell;vertical-align:middle;

子级元素设置：margin:0 auto;

1. css3的box方法

父级元素设置：

display:box; box-orient:horizontal; box-pack:center; box-align:center;

display:-webkit-box; -webkit-box-orient:horizontal; -webkit-box-pack:center; -webkit-box-align:center;

display:-moz-box; -moz-box-orient:horizontal; -moz-box-pack:center; -moz-box-align:center;

display:-o-box; -o-box-orient:horizontal; -o-box-pack:center; -o-box-align:center;

display:-ms-box; -ms-box-orient:horizontal; -ms-box-pack:center; -ms-box-align:center;

JS截取与分割字符串常用技巧总结：

1. substring()

str.substring(start,end)表示从start到end之间的字符串，包括start位置的字符但不包括end位置的字符。

1. substr()

substr(start,length)表示从start位置开始，截取length长度的字符串

1. split()

使用一个指定的分隔符把一个字符串分割存储到数组

var str=”jpg|bmp|gif|ico|png”;

var arr=str.split(“|”)

//arr是一个包含字符值"jpg"、"bmp"、"gif"、"ico"和"png"的数组

1. join()

使用您选择的分隔符将一个数组合并为一个字符串

1. indexOf()

返回字符串中匹配子串的第一个字符的下标

1. lastIndexOf()

返回从右向左出现某个字符或者字符串的首个字符索引值（与indexOf相反）

常见REACT面试题、REACT基础知识点：

1. React中key的作用是什么？

render(){

return(

<ul>

{

this.state.todoItems.map((item,key)=>{

return <li key={key}>{item}</li>

})

}

</ul>

)

}

key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，因而我们绝对不可忽视转换函数中key的重要性。

1. 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和老树的节点差异，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。

1. 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
* 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
* 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）
* 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

1. 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方法中写出更优化的dom diff算法，可以极大地提高性能。

1. 为什么虚拟dom会提高性能？（必考）

虚拟dom相当于在JS和真实DOM中间加了一个缓存，利用dom diff算法避免了没有必要的dom操作，从而提高性能。

1.用JavaScript对象结构标识DOM树的结构；2.然后用这个树构建一个真正的DOM树，插到文档当中，当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异，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。

1. 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。

1. 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：

class CustomForm extends Component{

handleSubmit=()=>{

console.log(“Input Value:”,this.input.value);

}

render(){

return(

<form onSubmit={this.handleSubmit}>

<input type=”text” ref={(input)=>this.input=input} />

<button type=”submit”>Submit</button>

</form>

)

}

}

上述代码中的input域包含了一个ref属性，该属性声明的回调函数会接收input对应的DOM元素，我们将其绑定到this指针以便在其它的函数中使用。另外值得一提的是，refs并不是类组件的专属，函数式组件同样能够利用闭包暂存其值

句柄，是系统程序对数据定义的标识，通常是一个整数。系统程序和用户程序需要进行数据交换，为了避免传递大量数据，尤其是不希望用户直接操作数据，系统使用一个数字来表示这组数据。用户程序如果需要对数据进行修改，都只能通过系统程序提供的函数以及数据的句柄和相关参数来完成。系统程序很可能使用数据的地址作为句柄，但这仅仅是可能，完全取决于定义句柄的程序。

1. 展示组件(Presentational component)和容器组件(container component)之间有何不同

* 展示组件关心组件看起来是什么。展示专门通过props接受数据和回调，并且几乎不会有自身的状态，但当展示组件拥有自身的状态时，通常也只关心UI状态而不是数据的状态。
* 容器组件则更关心组件是如何运作的。容器组件会为展示组件或者其它容器组件提供数据和行为，它们会调用Flux actions，并将其作为回调提供给展示组件。容器组件经常是有状态的，因为它们是（其它组件的）数据源。

1. 类组件(Class component)和函数式组件(Functional component)之间有何不同

* 类组件不仅允许你使用更多额外的功能，如组件自身的状态和生命周期钩子，也能使组件直接访问store并维持状态
* 当组件仅是接收props，并将组件自身渲染到页面时，该组件就是一个“无状态组件(stateless component)”，可以使用一个纯函数来创建这样的组件。这种组件也被称为哑组件(dumb components)或展示组件

1. （组件的）状态(state)和属性(props)之间有何不同

* state是一种数据结构，用于组件挂载时所需数据的默认值。state可能会随着时间的推移而发生突变，但多数时候是作为用户事件行为的结果。
* props(properties的简写)则是组件的配置。props由父组件传递给子组件，并且就子组件而言，props是不可变的(immutable)。组件不能改变自身的props，但是可以把其子组件的props放在一起（统一管理）。Props也不仅仅是数据，回调函数也可以通过props传递。

1. 何为受控组件(controlled component)

在HTML中，类似<input> <textarea> <select>这样的表单元素会维护自身的状态，并基于用户的输入来更新。当用户提交表单时，前面提到的元素的值将随表单一起被发送。但在React中会有些不同，包含表单元素的组件将会在state中追踪输入的值，并且每次调用回调函数时，如onChange会更新state，重新渲染组件。一个输入表单元素，它的值通过React的这种方式来控制，这样的元素就被称为“受控元素”。

1. 何为高阶组件(higher order component)

高阶组件是一个以组件为参数并返回一个新组件的函数。HOC运行你重用代码、逻辑和引导抽象。最常见的可能是Redux的connect函数。除了简单分享工具库和简单的组件，HOC最好的方式是共享React组件之间的行为。如果你发现你在不同的地方写了大量代码来做同一件事情时，就应该考虑将代码重构为可重用的HOC。

1. 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。

1. 除了在构造函数中绑定this，还有其它方式吗？

你可以使用属性初始值设定项(property initializers)来正确绑定回调，create-react-app也是默认支持的。在回调中你可以使用箭头函数，但问题是每次组件渲染时都会创建一个新的回调。

1. 在构造函数中调用super(props)的目的是什么

在super()被调用之前，子类是不能使用this的，在ES2015中，子类必须在constructor中调用super()。传递props给super()的原因则是便于（在子类中）能在constructor访问this.props。

1. 应该在React组件的何处发起Ajax请求

在React组件中，应该在componentDidMount中发起网络请求。这个方法会在组件第一次“挂载”（被添加到DOM）时执行，在组件的生命周期中仅会执行一次。更重要的是，你不能保证在组件挂载之前Ajax请求已经完成，如果是这样，也就意味着你将尝试在一个未挂载的组件上调用setState，这将不起作用。在componentDidMount中发起网络请求将保证这有一个组件可以更新了。

1. 描述事件在React中的处理方式

为了解决跨浏览器兼容性问题，您的React中的事件处理程序将传递SyntheticEvent的实例，它是React的浏览器本机事件的跨浏览器包装器。这些SyntheticEvent与您习惯的原生事件具有相同的接口，除了它们在所有浏览器中都兼容。有趣的是，React实际上并没有将事件附加到子节点本身。React将使用单个事件监听器监听顶层的所有事件。这对于性能都是有好处的，这也意味着在更新DOM时，React不需要担心跟踪事件监听器。

1. createElement和cloneElement有什么区别

React.createElement()：JSX语法就是用React.createElement()来构建React元素的。它接受三个参数，第一个参数可以使一个标签名，如div、span，或者React组件。第二个参数为传入的属性。第三个以及之后的参数，皆作为组件的子组件。

React.createElement(

type,

[props],

[…children]

)

React.cloneElement()与React.createElement()相似，不同的是它传入的一个参数是一个React元素，而不是标签名或组件。新添加的属性会并入原有的属性，传入到返回的新元素中，而旧的子元素将被替换。

React.cloneElement(

element,

[props],

[…children]

)

1. React中有3种构建组件的方式

React.createClass()、ES6 class 和无状态函数

1. React组件的划分业务组件技术组件

* 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提供的connect方法联系起来

1. 简述flux思想
2. 用户访问View
3. View发出用户的Action
4. Dispatcher收到Action，要求Store进行相应的更新
5. Store更新后，发出一个“change”事件
6. View收到“change”事件后，更新页面
7. React项目中用过什么脚手架（本题是开放性题目）

create-react-app Yeoman等

1. 了解redux么，说一下redux吧

* 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getS tate获取最新的数据，flux也是用来进行数据操作的，有4个组成部分action、dispatch、view、store，工作流程是view发出一个action，派发器接收action，让store进行数据更新，更新完成以后store发出change，view接受change更新视图。Redux和Flux很像。主要区别在于Flux有多个可以改变应用状态的store，在Flux中dispatcher被用来传递数据到注册的回调事件，但是在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
* 新增 state,对状态的管理更加明确，通过 redux，流程更加规范了，减少手动编码量，提高了编码效率，同时缺点时当数据更新时有时候组件不需要，但是也要重新绘制，有些影响效率。一般情况下，我们在构建多交互，多数据流的复杂项目应用时才会使用它们

1. redux有什么缺点
2. 一个组件所需要的数据，必须由父组件传过来，而不能像flux中直接从store取。
3. 当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新render，可能会影响效率，或者需要写复杂的shouldComponentUpdate进行判断

react组件的生命周期方法都可以被分割成4个阶段：初始化、挂载阶段（Mounting）、更新阶段（Updating）、卸载阶段（Unmounting）。

* Mounting—下面这些方法都将会在component实例被创建和插入到DOM后调用：

constructor()

componentWillMount()

render()

componentDidMount()

* Updating—props或者state的变化都会导致更新。下面这些方法会在component重新渲染时调用：

componentWillReceiveProps()

shouldComponentUpdate()

componentWillUpdate()

render()

componentDidUpdate()

* Unmounting—该方法将会在component从DOM中移除时调用

componentWillUnmount()

1. componentWillMount

componentWillMount()是在组件挂载（Mount）之前被调用

componentWillMount()是唯一一个在服务器端渲染（ssr）调用的生命周期钩子

关于setState在componentWillMount使用：可以使用。因为它是render方法之前被调用，因此setState也不会导致重绘（re-render）

1. componentDidMount

componentDidMount()在组件挂载之后立即执行

在这个钩子里适合：

* ajax请求
* 初始化DOM节点的操作
* 设置计时器setTimeout或者setInterval（温馨提示，别忘了在componentWillUnmount关闭这些计时器）

关于setState在componentDidMount使用：可以使用。但是经常导致性能问题。当然非要在render前拿到DOM节点的大小和位置，是可以用的。

1. componentWillReceiveProps(nextProps)

componentWillReceiveProps将会在已挂载组件（mounted component）接收到新的props之前调用。所以初始化props的mount是不会触发这个函数。直接setState也不会触发这个函数。

在这个钩子里适合：

* 更新state的值（比如重置）
* 比较this.props和nextProps

特别要注意的是，当父组件导致该组件re-render时，即便props没有发生任何变化，react也可能执行该钩子函数。所以如果想要真正处理props的变化，要比较当前props和nextProps

setState在componentWillReceiveProps中可以使用。

1. shouldComponentUpdate(nextProps,nextState)

当改变state或者props并且在render之前会调用shouldComponentUpdate，该钩子函数用于告诉React组件是否需要重新渲染。

shouldComponentUpdate默认return true，如果return false，componentWillUpdate、render、componentDidUpdate都将不会被调用。当return false时，当它们的state发生改变时，并不会阻止子组件（child component）进行重新渲染。

shouldComponentUpdate在2种情况下不会被调用：

* 组件初始化
* 使用forceUpdate的情况

shouldComponentUpdate和react组件性能优化相关的：this.state和nextState、this.props和nextProps做比较来决定出return false并告诉react可以不更新该组件。

如果做浅层的数据比较完全可以用PureComponent来代替（深层的嵌套数据PureComponent也无能为力）

react不建议在shouldComponentUpdate做深层的对比或者用JSON.stringfy()，因为这样反而损害到性能。

1. componentWillUpdate(nextProps,nextState)

state或props更新后，re-render之前调用

注意：不要在componentWillUpdate中使用this.setState，或者redux分发一个action(dispatch a Redux action)，因为在componentWillUpdate之前会触发组件的更新。如果非要做以上操作，可以在componentWillReceiveProps中。

1. componentDidUpdate(prevProps,prevState)

在组件更新之后马上调用componentDidUpdate

在这个钩子函数中可以：

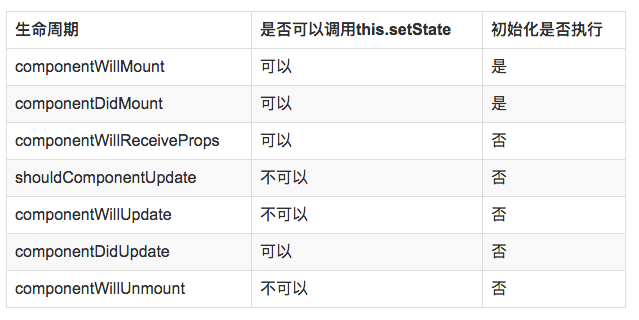
* 操作DOM
* 发起网络请求

1. componentWillUnmount

在组件卸载(unmounted)和销毁(destroyed)前调用

在componentWillUnmount你可以执行任何需要清除的方法。比如：

* 清除计时器
* 断开网络请求
* 解绑DOM事件
* 等等



注意：

* componentWillMount和componentWillReceiveProps调用setState不会重复渲染(re-render)
* componentDidUpdate，不能直接this.setState，不然会溢出栈。需要对prevProps与this.props和prevState和this.state做一个判断再执行this.setState。类似while循环不能陷入死循环。

1. props和state的区别

* props是别人的，props实现组件间的状态传递，props从父组件到子组件的数据传递；state是自己的，state只能定义在组件内部，定义组件的自己的状态。
* props是不可变的；state可以通过this.setState改变



1. jsx是什么？

jsx是一个看起来很像XML的JavaScript语法扩展

jsx优点：

* 执行更快，因为它在编译为JavaScript代码后进行了优化
* 它是类型安全的，在编译过程中就能发现错误
* 编写模板更加简单快速
* 更加直观，可读性高

jsx最终是JavaScript对象，react通过这个对象来创建或者更新虚拟元素最终来管理virtual DOM（虚拟DOM）。

jsx对象元素可以理解为和真实元素一一对应的，它的创建、更新、删除都是在内存中完成的。并不会直接渲染到真实DOM中去，整个react应用程序唯一操作到DOM就是：ReactDOM.render(<HappyReact />,document.getElementById(‘id’));

1. jsx语法

* render函数只能return一个根节点，只允许被一个标签包裹
* Component命名首字大写，HTML标签用小写
* 如果不存在子节点，可以使用自闭合<div />
* jsx的注释为：{/\*\*/}
* JavaScript属性表达式，属性值用{}
* 三元表达式
* 数组递归（渲染列表）map
* 2个特殊属性class、for。因为class、for在JavaScript中这2个单词都是关键词，因此需要做一手转换。其它属性可以像html一样添加上去。
* jsx书写样式
* 事件处理，使用inline方式的驼峰式写法，例如onClick、onChange
* HTML转义 --> dangerouslySetInnerHTML={{\_\_html: '<div>hhh</div>'}}
* 利用es6中…展开运算符
* 如果属性值是true，直接写属性名，如：<Button disabled={true} />可以写成：<Button disabled />
* false、null、undefined、true是有效的子内容，但是不会被渲染出来。以下表达式渲染结果是一样的：

<div />

<div></div>

<div>{false}</div>

<div>{null}</div>

<div>{undefined}</div>

<div>{true}</div>

1. refs是什么？（refs功能，如何实现？）

react提供了一种特殊属性，允许您直接访问DOM元素或组件实例。

ref可以返回一个字符串（string）或者一个回调函数（cb），这个回调函数会在组件实例化或者销毁之后立即执行。

字符串refs在未来的版本中可能被移除，所以推荐回调的方式来代替。

class TextInput extends Component{

componentDidMount(){

this.textInput.focus()

}

render(){

return (

<input ref={(input)=>this.textInput=input} />

)

}

}

官方推荐几种很好的方式使用refs：

* 管理焦点，文本选择或者媒体播放
* 触发重要的动画
* 整合第三方DOM库

当然能不用refs就尽量不要用refs，不要过度依赖refs来解决问题

1. 什么是受控组件？什么是非受控组件？

在react表单组件可被分为两类：受控组件 和 非受控组件。

1. 关于setState

官方是这么定义setState：setState()排队更改组件的state，并通过更新state来告诉React，该组件极其子组件需要重新渲染。这是用于**响应事件处理程序**和**服务器响应**更新用户界面的主要方法。

setState是同步还是异步？

1. 调用setState()实际上发生了什么？

简单地说，就是更改state、更新UI

复杂地说，就是怎么合并新的state，怎么根据新state来更新UI

1. setState()第二个参数是什么？它是什么作用？

setState的第二个参数是一个可选的回调参数。这个回调函数将在setState完成后执行，并且重新渲染组件。在这个回调函数中你可以拿到刚更新的state的值。但是这样的逻辑，官方推荐使用componentDidUpdate。

1. 如何在setState后直接获取修改后的值
2. setState第2个参数，回调函数中获取
3. 使用setTimeout
4. setState第一个参数有两种传递方式1.一个对象 2.一个函数 这两种写法有什么区别呢？

使用setState要注意：

1. setState可能会引发不必要的渲染(shouldComponentUpdate/PureComponent)
2. setState无法完全掌控应用中所有组件的状态(Redux/Mbox)
3. 什么是高阶组件？它是如何使用的？

高阶组件，它是一个函数，并不是一个组件。

通俗地讲，它接收一个React组件作为输入，输出一个新的增强版的React组件。

1. 什么是PureComponent？介绍一下PureComponent和shouldComponentUpdate有什么区别？

PureComponent和Component是相同的，只要把继承类从Component换成PureComponent即可。PureComponent改变了shouldComponentUpdate，它会自动检查组件是否重新渲染。也就是说，只有当PureComponent检查到props或者state变化时，才会调用render函数，因此不用写额外的价差。还可以减少Virtual DOM的生成和比对过程，达到提升性能的目的。

**注意**：PureComponent的shouldComponent只是进行了浅比较（state、props对象结构简单，可以理解为对象只有一层），对于复杂且嵌套更深层数据的比较会出现偏差。对于深比较，你可以选择在shouldComponentUpdate进行深比较检查来确定组件是否渲染，但是你要知道 深比较 是非常昂贵的。当然，你可能知道 使用Immutable来帮助嵌套数据的快速比较。

1. shouldComponentUpdate的作用以及它的重要性

shouldComponentUpdate允许我们手动地判断是否要进行组件更新，根据组件的应用场景设置函数的合理返回值能够帮我们避免不必要的更新。

1. 为什么我们利用循环产生的组件中要用上key这个特殊的prop？

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。之所以需要key，因为react是非常高效的，它会借助元素的key值来判断元素是新创建的，或者移动（交换位置）而来的，从而减少不必要的元素重渲染。

**JavaScript实现动态添加、移除元素或属性的方法分析：**

appendChild(newnode)：向节点的子节点列表的末尾添加新的子节点；

insertBefore(newnode,existingnode)：在已有子节点之前插入新的子节点；

removeChild(node)：删除元素的某个指定的子节点，并以Node对象返回被删除的节点，如果节点不存在则返回null；

innerHTML：属性设置或返回表格行的开始和结束标签之间的HTML；

setAttribute(attributename,attributevalue)：添加指定的属性，并为其赋指定的值。将属性设置为undefined等同于删除；

removeAttribute(attributename)：删除指定的属性；

getAttributeNode(attributename)：以Attr对象返回指定属性名的属性值；

removeAttributeNode(attributenode)：删除Attr形式指定的属性，同时返回被删除的Attr形式的属性。

**CSS3 transform属性**

div{

transform:rotate(7deg);

-ms-transform:rotate(7deg); /\* IE9 \*/

-moz-transform:rotate(7deg); /\* Firefox \*/

-webkit-transform:rotate(7deg); /\* Safari和Chrome \*/

-o-transform:rotate(7deg); /\* Opera \*/

}

transform属性向元素应用2D或3D转换。该属性允许我们对元素进行旋转、缩放、移动和倾斜。

transform:none | transform-functions;

none 定义不进行转换

matrix(n,n,n,n,n,n) 定义2D转换，使用六个值的矩阵

matrix3d(n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n) 定义3D转换，使用16个值的4x4矩阵

translate(x,y) 定义2D转换

translate3d(x,y,z) 定义3D转换

translateX(x) 定义转换，只是用X轴的值

translateY(y) 定义转换，只是用Y轴的值

translateZ(z) 定义3D转换，只是用Z轴的值

scale(x,y) 定义2D缩放转换

scale3d(x,y,z) 定义3D缩放转换

scaleX(x) 通过设置X轴的值来定义缩放转换

scaleY(y) 通过设置Y轴的值来定义缩放转换

scaleZ(z) 通过设置Z轴的值来定义3D缩放

rotate(angle) 定义2D旋转，在参数中规定角度

rotate3d(x,y,z,angle) 定义3D旋转

rotateX(angle) 定义沿着X轴的3D旋转

rotateY(angle) 定义沿着Y轴的3D旋转

rotateZ(angle) 定义沿着Z轴的3D旋转

skew(x-angle,y-angle) 定义沿着X和Y轴的2D倾斜转换

skewX(angle) 定义沿着X轴的2D倾斜转换

skewY(angle) 定义沿着Y轴的2D倾斜转换

perspective(n) 为3D转换元素定义透视视图

**CSS3 transition 属性**

transtion属性是一个简写属性，用于设置4个过渡属性：

* transition-property 规定设置过渡效果的CSS属性的名称
* transition-duration 规定完成过渡效果需要多少秒或毫秒
* transition-timing-function 规定过渡效果的速度曲线
* transition-delay 定义过渡效果何时开始

**注意**：请始终设置transition-duration属性，否则时长为0，就不会产生过渡效果。

默认值：all 0 ease 0

transition:property duration timing-function delay;

**CSS :before 选择器(:after 选择器)**

p:before{

content:”台词：”

}

:before选择器在被选元素的内容前面插入内容，请使用content属性来指定要插入的内容

在每个<p>元素前面插入内容，并设置所插入内容的样式：

p:before{

content:”台词：-”

background-color:yellow;

color:red;

font-weight:bold;

}

input:checked input:not(checked)

input:enabled input:disabled

JS中的循环---最全的循环总结：https://www.cnblogs.com/baiyunke/p/7821299.html

1. while循环

var num=1; //1.声明循环变量

while(num<=10){ //2.判断循环条件

document.write(num+”<br />”); 3.执行循环体操作

num++; //4.更新循环变量

}

while循环中的表达式，运算结果可以是各种类型，但是最终都会转为真假，转换规则如下：

1. Boolean：true为真，false为假
2. String：空字符串为假，所有非空字符串为真
3. Number：0为假，一切非0数字为真
4. null/undefined/NaN：全为假
5. Object：全为真
6. do-while循环

while循环特点：先判断后执行

do-while循环特点：先执行再判断，即使初始条件不成立，do-while循环至少执行一次

1. for循环
2. for有3个表达式：①声明循环变量；②判断循环条件；③更新循环变量。3个表达式之间，用;分割，for循环3个表达式都可以省略，但是两个“;”缺一不可。
3. for循环的执行特点：先判断再执行，与while相同
4. for循环3个表达式都可以有多部分组成，第二部分多个判断条件用&& ||连接，第一三部分用逗号分割
5. for-in循环

主要用于遍历对象

for()中的格式：for(key in value){}

key表示obj对象的每一个键值对的键！所有循环中，需要使用obj[keys]来取到每一个值。

for-in循环，遍历时不仅能读取对象自身上面的成员属性，也能延续原型链遍历出对象的原型属性。

所以，可以使用hasOwnProperty判断一个属性是不是对象自身上的属性。

obj.hasOwnProperty(keys)==true 表示这个属性是对象的成员属性，不是原先属性。

1. for-of循环

ES6借鉴C++、Java、C#和Python语言，引入了for…of循环，作为遍历所有数据结构的统一方法。

一个数据结构只要部署了Symbol.iterator属性，就被视为具有iterator接口，就可以用for…of循环遍历它的成员。也就是，for…of循环内部调用的是数据结构的Symbol.iterator方法。

for…of循环可以使用的范围包括数组、Set和Map结构、某些类似数组的对象（比如arguments对象、DOM NodeList对象）、后文的Generator对象，以及字符串。

1. 数组：JavaScript原有的for…in循环，只能获得对象的键名，不能直接获取键值。ES6提供for…of循环，允许遍历获取键值。
2. Set和Map结构
3. 类似数组的对象

与其它遍历语法比较：

for…in循环有几个缺点：①数组的键名是数字，但是for…in循环是以字符串作为键名“0”、“1”、“2”等等；②for…in循环不仅遍历数字键名，还会遍历手动添加的其它键，甚至包括原型链上的键；③某些情况下，for…in循环会以任意顺序遍历键名。

for…in循环主要为遍历对象而设计的，不适用于遍历数组。

for…of循环：有着同for…in一样的简洁语法，但是没有for…in那些缺点。不同于forEach方法，它可以与break、continue和return配合使用。提供了遍历所有数据结构的统一操作接口。

1. 循环控制语句
2. break：跳出本层循环，继续执行循环后面的语句。如果循环有多层，则break只能跳出一层。
3. continue：跳过本次循环剩余的代码，继续执行下一次循环。①对于for循环，continue之后执行的语句，是循环变量更新语句i++；②对于while、do-while循环，continue之后执行的语句是循环条件判断。因此，使用这2个循环时，必须将continue放到i++之后使用，否则continue将跳过i++进入死循环。

**ES6新增类型set**详解：https://www.jianshu.com/p/e09b5e5e5e91

set是ES6中新增的类型，和数组类似，唯一不同在于该类型不会有重复的数据，一般常用来对数据进行去重操作。

**new Map()**

**JavaScript toString() 方法**

toString()方法可把一个逻辑值转换为字符串，并返回结果。booleanObject.toString()

**mouseover和mouseenter的区别：**

mouseover：当鼠标指针位于元素上方时，会发生mouseover事件。对应：mouseout。

mouseenter：当鼠标指针穿过元素时，会发生mouseenter事件。对应：mouseleave。

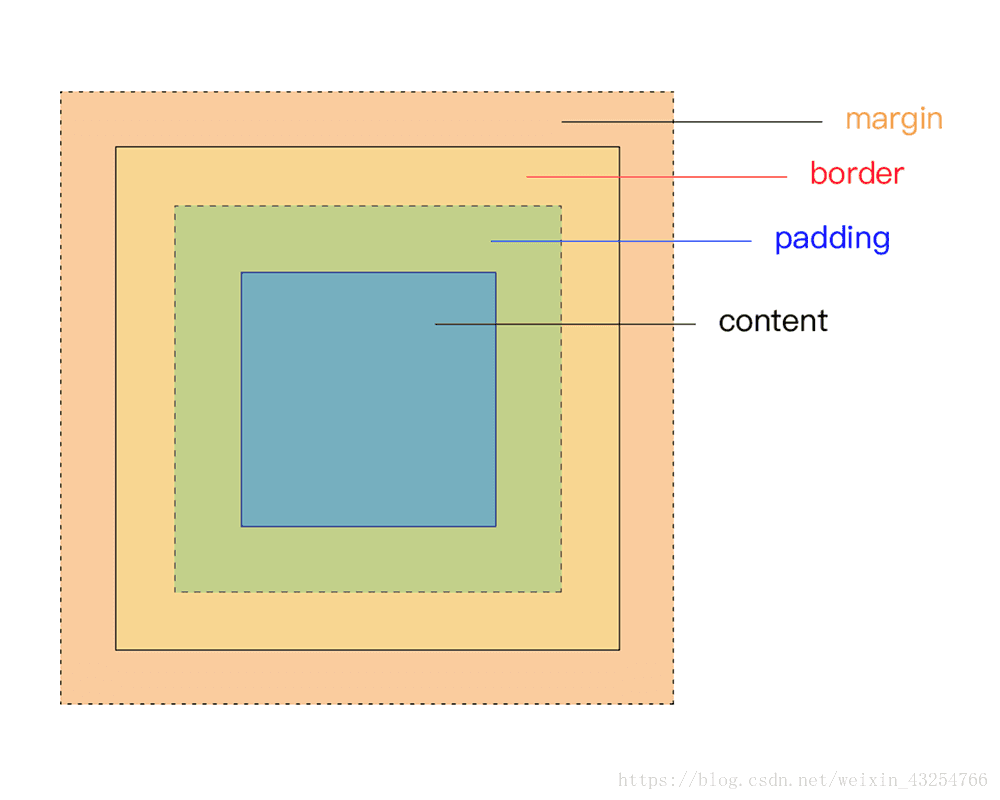
mouseenter是穿过，所以只能触发一次事件；mouseover是位于上方，如果给div设定了一个mouseover事件，其子孙后代都可以响应事件，所以一旦鼠标从父级进入自己也会触发这个效果，当子级回到父级也会触发这种效果。

所以可以这么理解：mouseenter事件只作用于目标元素，而mouserover作用于目标元素及其后代元素。

**什么是盒子模型？**

在我们平时看到的网页中，内部的每一个标签元素它都是由4个部分构成的：内容(content)、外边距(margin)、内边距(padding)、边框(border)。

其中一个重点是：**IE盒模型和标准盒模型的区别**。IE盒模型的content包括border、padding。



语法：box-sizing:border-box || content-box || inherit

* 当使用content-box时，页面将采用标准模式来解析计算，content-box也是默认模式
* 当使用border-box时，页面将采用怪异模式解析计算，怪异模式也称为IE模式
* 当使用inherit时，页面将从父元素继承box-sizing的值

页面导入样式时有几种方法？它们之间有什么区别？

* link标签引入，也是当下用的最多的一种方式，它属于XHTML标签，除了能加载CSS外，还能定义rel、type、media等属性
* @import引入，@import是CSS提供的，只能用于加载CSS
* style嵌入方式引入，减少页面请求（优点），但只会对当前页面有效，无法复用、会导致代码冗余，不利于项目维护（缺点）。此方式一般只会在项目主站首页使用（腾讯、淘宝、网易、搜狐）等大型网站主页，之前有看到过都是这种方式，但后来有些也舍弃了。

小结：link页面被加载时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载，且link是XHTML标签，无兼容问题；link支持动态JS去控制DOM节点去改变样式，而@import不支持。

**简单讲述一下块元素、内联元素、空元素有哪些？它们之间的区别？**

* 行内元素有：a、b、span、img、input、select、textarea、em、img、strong
* 块级元素有：ul、ol、li、dl、dt、dd、h1、h2、h3、h4、h5、h6、p、section、div、form 等
* 空元素：<input type=”hidden” />、<br>、<hr>、<link>、<meta>

小结：块元素总是独占一行，margin对内联元素上下不起作用

说说cookies、sessionStorage、localStorage，你对它们的理解

* cookie是网站为了标示用户身份而存储在用户本地终端上的数据（通常经过加密），cookie数据始终在同源的http请求中携带，即：会在浏览器和服务器间来回传递。
* sessionStorage和loacalStorage不会自动把数据发给服务器，仅在本地保存。
* 大小：cookie数据大小不能超过4K，sessionStorage和localStorage虽然也有存储大小的限制，但是比cookie大得多，可以达到5M或更大。
* 时效：localStorage存储持久数据，浏览器关闭后数据不会丢失，除非用户主动删除数据或清除浏览器/应用缓存。sessionStorage数据在当前浏览器窗口关闭后自动删除。cookie设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。

1. 如何让cookie浏览器关闭就失效？----不对cookie设置任何正、负或0时间的即可。
2. sessionStorage在浏览器多窗口之间（同域）数据是否互通共享？----不会，都是独立的，localStorage会共享
3. 能让localStorage也跟cookie一样设置过期时间吗？答案是可以的，在存储数据时，也存储一个时间戳，get数据之前，先拿当前时间跟你之前存储的时间戳做比较。

**简述一下你对HTML语义化的理解**

语义化是指根据内容的类型，选择合适的标签（代码语义化），即用正确的标签做正确的事情；html语义化让页面的内容结构化，结构更清晰，有助于浏览器、搜索引擎解析对内容的抓取；语义化的HTML在没有CSS的情况下也能呈现较好的内容结构与代码结构；搜索引擎的爬虫也依赖HTML标记来确定上下文和各个关键字的权重，利于SEO。

**position的static、relative、absolute、fixed的区别**

* absolute：绝对定位，元素会相对于值不为static的第一个父元素进行定位（会一直往父级节点查找），它是脱离正常文档流、不占位的；
* fixed：同样是绝对定位，但元素会相对于浏览器窗口进行定位，而不是父节点的position（IE9以下不支持）；
* relative：相对定位，元素相对于自身正常位置进行定位，属于正常文档流；
* static：position的默认值，也就是没有定位，当元素设置该属性后（top、bottom、left、right、z-index）等属性将失效；
* inherit：从父元素继承position属性的值。

**用纯CSS样式写三角形ICON**

**css display:box 新属性** 在元素上设置该属性，可使其子代排列在同一水平上

**什么是外边距合并：**当2个垂直元素都设置有marign外边距相遇时，它们将形成一个外边距。合并后的外边距的高度等于2个发生合并的外边距的值中的较大的那个。

:before和:after两伪元素，平时都是使用双冒号还是单冒号？有什么区别？以及它们的作用

* 单冒号（:）用于CSS3伪类，双冒号（::）用于CSS3伪元素。（伪元素由双冒号和伪元素名称组成）
* 双冒号是在当前规范中引入的，用于区分伪类和伪元素。不过浏览器需要同时支持旧的已经存在的伪元素写法，比如： :first-line :first-letter :before :after等。而新的在CSS3中引入的伪元素则不允许再支持旧的单冒号的写法。
* 想让插入的内容出现在其它内容前，使用::before，之后使用::after，在代码顺序上，::after生成的内容也比::before生成的内容靠后。如果按堆栈视角，::after生成的内容会在::before生成的内容之上。

Chorme、Safari等浏览器，当表单提交用户选择记住密码后，下次自动填充表单的背景变成黄色，影响了视觉体验是否可以修改？

方法一：

input:-webkit-autofill,textarea:-webkit-autofill,select:-webkit-autofill{

background-color:#fff; //设置成元素原本的颜色

background-image:none;

color:rgb(0,0,0);

}

方法二：

input:-webkit-autofill{

-webkit-box-shadow:0px 0 3px 100px #ccc inset; //背景色

}

**浏览器的最小字体为12px，如果还想再小，怎么办**：css3的样式transform:scale(0.7)，scale有缩放功能

**display:none与visibility:hidden两者的区别：**

* display:none;在页面中不占位置，visibility:hidden保留原来的位置
* display:none显示/隐藏 页面会产生回流和重绘的问题，visibility则不会。（JS的重绘/回流）。

**CSS样式优先级排序如何计算的？**

* 同权重：内联样式表（标签内部）>嵌入样式表（当前文件中）>外部样式表（外部文件中）
* !important > id > class >tag
* important比内联优先级高

**li与li之间有看不见的空白间隔是什么原因引起的？怎么解决？**

* 行框的排列会受到中间空白（回车/空格）等的影响，因为空格也属于字符，这些空白也会被应用样式、占据空间，所以会有间隔，把字符大小设为0，就没有空格了。

**js有哪些基本数据类型**

ECMAScript标准定义有7种数据类型：

* Boolean
* Null
* Undefined
* Number
* String
* Object（Array、Function、Object）
* Symbol（ECMAScript6新定义，Symbol生成一个全局唯一、表示独一无二的值）

**ES6入门之Symbol**

ES5对象属性名都是字符串，容易造成属性名的冲突。

eg：var a={name:’Lucy’}; a.name=’Lily’; 这样就会重写属性

ES6引入了一种新的原始数据类型Symbol，表示独一无二的值。

注意：Symbol函数前不能使用new命令，否则会报错。这是因为生成的Symbol是一个原始类型的值，不是对象。

Symbol函数可以接受一个字符串作为参数，表示对Symbol实例的描述，主要是为了在控制台显示，或者转为字符串时，比较容易区分。

// 没有参数的情况

var s1=Symbol();

var s2=Symbol();

s1===s2 //false

//有参数的情况

var s1=Symbol(“foo”);

var s2=Symbol(“foo”);

s1===s2’ //false

Symbol值不能与其它类型的值进行运算

作为属性名的Symbol

var mySymbol=Symbol();

// 第一种写法

var a={};

a[mySymbol]=”Hello!”;

// 第二种写法

var a={ [mySymbol]:”Hello!” }

// 第三种写法

var a={};

Object.defineProperty(a,mySymbo,{value: ”Hello!”});

//以上写法都得到同样结果

a[mySymbol] //”Hello!”

注意：Symbol值作为对象属性名时，不能用点运算符

var a={};

var name=Symbol();

a.name=”Lily”;

a[name]=”Lucy”;

console.log(a.name,a[name]); //Lily,Lucy

Symbol值作为属性名时，该属性是公开属性，不是私有属性

Symbol在类外部可以访问，只是不会出现在for…in、for…of循环中，也不会被Object.keys()、Objectt.getOwnPropertyNames()返回。但有一个Object.getOwnPropertySymbol方法，可以获取指定对象的所有Symbol属性名。

Symbol.for() Symbol.keyFor()

Symbol.for机制有点类似于单例模式，首先在全局中搜索有没有以该参数作为名称的Symbol值，如果有就返回这个Symbol值，否则就新建并返回一个以该字符串为名称的Symbol值。

var s1=Symbol.for(‘foo’);

var s2=Symbol.for(‘foo’);

console.log(s1===s2); //true

Symbol.keyFor方法返回一个已登记的Symbol类型值的key。实质就是检测该Symbol是否已创建

var s1=Symbol.for(‘foo’);

Symbol.keyFor(s1); // ’foo’

var s2=Symbol(‘foo’);

Symbol.keyFor(s2); // undefined

**用JS将386485473.88转换为386,485,473.88（千位分隔符）：（看demo）**

**JS toLocaleString()方法**

toLocaleString()方法可根据本地时间把Date对象转换为字符串，并返回结果。

语法：dateObject.toLocaleString()

返回值：dateObject的字符串表示，以本地时间区表示，并根据本地规则格式化

**JS toString()、toLocaleString()、valueOf()3个方法的区别**

Array、Boolean、Date、Number等对象都具有toString()、toLocaleString()、valueOf()3个方法，那这3个方法有什么区别？

1. JS Array

* valueOf()：返回数组本身
* toString()：把数组转换为字符串，并返回结果，每一项以逗号分隔
* toLocalString()：把数组转换为本地数组，并返回结果

1. JS Boolean

* valueOf()：返回Boolean对象的原始值
* toString()：根据原始布尔值或者booleanObject对象的值返回字符串“true”或“false”。默认为“false”。
* toLocalString()：Boolean对象没有toLocalString()方法。但是在Boolean对象上使用这个方法也不会报错。

1. JS Date

* valueOf()：返回Date对象的原始值，以毫秒表示
* toString()：把Date对象转换为字符串，并返回结果。使用本地时间表示
* toLocalString()：可根据本地时间把Date对象转换为字符串，并返回结果，返回的字符串根据本地规则格式化

1. JS Math

console.log(Math.PI.valueOf()); //结果：3.141592653589793

* valueOf：返回Math对象的原始值

1. JS Number

var num=new Number(1337);

console.log(num.valueOf());

console.log(num.toString());

console.log(num.toLocaleString());

* valueOf：返回一个 Number 对象的基本数字值
* toString()：把数字转换为字符串，使用指定的基数
* toLocalString()：把数字转换为字符串，使用本地数字格式顺序

1. JS String

var string=new String(“abc”);

console.log(string.valueOf());

console.log(string.toString());

* valueOf：返回某个字符串对象的原始值
* toString()：返回字符串

toString()方法与toLocalString()方法区别：

* toLocalString()是调用每个数组元素的toLocaleString()方法，然后使用地区特定的分隔符把生成的字符串连接起来，形成一个字符串
* toString()方法获取的是String（传统字符串），而toLocaleString()方法获取的是LocaleString（本地环境字符串）
* 如果你开发的脚本在世界范围都有人使用，那么将对象转换成字符串时，请使用toString()方法来完成
* LocaleString()会根据你机器的本地环境来返回字符串，它和toString()返回的值在不同的本地环境下使用的符号会有微妙的变化
* 所以使用toString()是保险的，返回唯一值得方法，它不会因为本地环境的改变而发生变化。如果是为了返回时间类型的数据，推荐使用LocaleString()。若是在后台处理字符串，请务必使用toString()。

**JavaScript toUpperCase()方法**

定义和用法：toUpperCase()方法用于把字符串转换为大写

语法：string.toUpperCase()

相对应：toLowerCase，把字符串转换为小写

**项目上线前，你们做过哪些性能优化：**

* 图片预加载，CSS样式表放在顶部且link链式引入，Javascript放在底部body结束标签前
* 减少http请求次数，如防抖和节流
* API接口数据设置缓存，CSS Sprites/SVG Sprites，JS、CSS源码压缩、图片大小控制合适，使用iconfont或SVG它们比图片更小更清晰，网页Gzip压缩
* 减少DOM操作次数，优化javascript性能
* 减少DOM元素数量，合理利用:after :before等伪类
* 避免重定向，图片懒加载，前后端分离开发，资源按需加载，最好能做到首屏直出（即：服务端渲染）
* 避免使用CSS Expression（CSS表达式），又称Dynamic properties（动态属性）；
* 多域名分发划分内容到不同域名，解决浏览器域名请求并发数问题，同时也解决了请求默认携带的cookie问题
* 尽量减少iframe使用，它会阻塞主页面的渲染
* 对所有资源压缩：JS、CSS、字体、图片等，甚至html

**你对重绘、重排的理解**

* 首先网页首次渲染生成时，这个可称为重排；
* 修改DOM、样式表、用户事件或行为（鼠标悬停、页面滚动、输入框键入文字、改变窗口大小等等），这些都会导致页面重新渲染，那么重新渲染，就需要重新生成布局和重新绘制节点，前者叫做“重排”，后者“重绘”
* 减少或集中对页面的操作，即多次操作集中在一起执行
* 总之可以简单总结为：重绘不一定重排，但重排必然会重绘

**Map和Set是ES6标准新增的数据类型**

new Set() 用来去重数组 Set类似于数组，区别在于它所有的成员都是唯一的，不能有重复的值

Set实例的方法分为两大类：操作方法（用于操作数据）和遍历方法（用于遍历成员）。

**四个操作方法：**

1. add(value)：添加某个值，返回Set结构本身
2. delete(value)：删除某个值，返回一个布尔值，表示删除是否成功
3. has(value)：返回一个布尔值，表示该值是否为Set的成员
4. clear()：清除所有成员，没有返回值

**四个遍历方法：**

1. keys()：返回键名的遍历器
2. values()：返回键值得遍历器
3. entries()：返回键值对的遍历器
4. forEach()：使用回调函数遍历每个成员
5. keys() values() entries()

由于Set结构没有键名，只有键值（或者说键名和键值是同一个值），所以keys方法和values方法的行为完全一致

Set结构的实例默认可遍历，它的默认遍历器生成函数就是它的values方法。这意味着，可以省略values方法，直接用for…of循环遍历Set

1. forEach()：Set结构的实例的forEach方法，用于对每个成员执行某种操作，没有返回值
2. 遍历的应用：

扩展运算符（…）内部使用for…of循环，所以也可用于Set结构

扩展运算符和Set结构相结合，就可以去除数组的重复成员

数组的map和filter方法也可以用于Set

Set类似于数组，而Map就类似于键值对(key,value)

ES6提供了Map数据结构。它类似于对象，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（包括对象）都可以当作键。

如果Map的键是一个简单类型的值（数字、字符串、布尔值），则只要2个值严格相等，Map将其视为一个键，包括0和-0。布尔值true和字符串true则是2个不同的键。undefined和null也是2个不同的键。虽然NaN不严格相等于自身，但Map将其视为同一个键。

Map有Size()属性，查看Map对象大小，set(key,value)，get(key)，delete(key)，has(key)，clear()方法

遍历方法：Map结构原生提供3个遍历器生成函数和1个遍历方法：

keys() 返回键名的遍历器

values() 返回键值的遍历器

entries() 返回所有成员的遍历器

forEach() 遍历Map的所有成员

需要特别注意：Map的遍历顺序就是插入顺序

Map结构转为数组结构，比较快速的方法是使用扩展运算符…

结合数组的map方法、filter方法，可以实现Map的遍历和过滤（Map本身没有map和filter方法）

Map可以跟数组、对象、JSON相互转换

**Array对象&数组方法大全：**

Array对象属性：

1. constructor 返回对创建此对象的数组函数的引用
2. length 设置或返回数组中元素的数目 如：new Array(size)，返回的数组，length字段将被设为size的值
3. prototype 使您有能力向对象添加属性和方法

Array对象方法：

1. concat() 连接2个或更多的数组，并返回结果
2. join() 把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
3. pop() 删除并返回数组的最后一个元素
4. push() 向数组的末尾添加一个或更多元素，并返回新的长度
5. reverse() 颠倒数组中元素的顺序
6. shift() 删除并返回数组的第一个元素
7. slice() 从某个已有的数组返回选定的元素
8. sort() 对数组的元素进行排序
9. splice() 删除元素，并向数组添加新元素
10. toSource() 返回该对象的源代码
11. toString() 把数组转换为字符串，并返回结果
12. toLocaleString() 把数组转换为本地字符串，并返回结果
13. unshift() 向数组的开头添加一个或更多元素，并返回新的长度
14. valueOf() 返回数组对象的原始值
15. indexOf()、lastIndexOf()
16. forEach() 对数组进行遍历循环，对数组中的每一项运行给定函数。这个方法没有返回值。参数都是function类型，默认有传参，参数分别为：遍历的数组内容、对应的数组索引、数组本身
17. map() 指：“映射”，对数组中的每一项运行给定函数，返回每次函数调用的结果组成的数组
18. filter() “过滤”功能，数组中的每一项运行给定函数，返回满足过滤条件组成的数组
19. every() 判断数组中每一项是否都满足条件，只有所有项都满足条件，才会返回true
20. some() 判断数组中是否存在满足条件的项，只要有一项满足条件，就会返回true
21. reduce()、reduceRight() 这2个方法都会实现迭代数组的所有项，然后构建一个最终返回的值。reduce()方法从数组的第一项开始，逐个遍历到最后。而reduceRight()则从数组的最后一项开始，向前遍历到第一项。

这2个方法都接收两个参数：一个在每一项上调用的函数、作为归并基础的初始值（可选的）。

传给reduce()和reduceRight()的函数接收4个参数：前一个值、当前值、项的索引和数组对象。

这个函数返回的任何值都会作为第一个参数自动传给下一项。第一次迭代发生在数组的第二项上，因此第一个参数是数组的第一项，第二个参数是数组的第二项。

box-shadow: offset-x offset-y blur spread color inset;

box-shadow: x轴偏移量 y轴偏移量 [阴影模糊半径] [阴影扩展] [阴影颜色] [投影方式];

blur：模糊 spread:伸展 inset:内凹

offset-x：必需，取值正负皆可。offset-x水瓶阴影的位置。

offset-y：必需，取值正负皆可。offset-y垂直阴影的位置。

blur：可选，只能取正值。blur-radius阴影模糊半径，0即无模糊效果，值越大阴影边缘约模糊。

spread：可选，取值正负都可。spread代表阴影的周长向四周扩展的尺寸，正值，阴影扩大，负值阴影缩小。

color：可选。阴影的颜色。如果不设置，浏览器会取默认颜色，通常是黑色，但各浏览默认颜色有差异，建议不要省略。可以是rgb(250,0,0)，也可以是有透明值的rgba(250,0,0,0.5)。

inset：可选。关键字，将外部阴影（默认outset）改为内部阴影。Inset阴影在背景之上，内容之下。

Note：inset可以写在参数的第一个或最后一个，其它位置是无效的。

css zoom属性

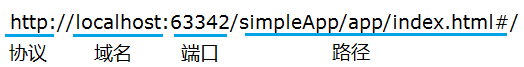
zoom:normal | <number> | <percentage>

默认值：normal

跨域的原理和方法、浏览器的同源策略、JSONP

域名、协议、端口不同，导致跨域

URL由协议、域名、端口、路径组成。如果2个URL的协议、域名、端口都相同，则表示它们同源。相反，只要协议、域名、端口有任何一个不同，就被当做跨域。



同源策略：浏览器采用同源策略，禁止页面加载或执行与自身来源不同的域的任何脚本。换句话说，浏览器禁止的是来自不同源的“document”或脚本，对当前“document”读取或设置某些属性。

情景：比如一个恶意网站的页面通过iframe嵌入了银行的登录页面（二者不同源），如果没有同源限制，恶意网页上的javascript脚本就可以在用户登录银行的时候获取用户名和密码。

浏览器中有哪些不受同源限制：<script>、<img>、<iframe>、<link>这些包含src属性的标签可以加载跨域资源。但浏览器限制了JavaScript的权限使其不能读、写加载的内容。

跨域：指一个域的网页去请求另一个域的资源。比如从http://www.baidu.com/页面去请求http://www.google.com的资源。

跨域技术-JSONP：

JSONP是什么？

上面提到的包含src属性的<script>标签可以加载跨域资源。JSONP就是利用<script>标签的跨域能力实现跨域数据的访问。

解决跨域问题常用的方案有2个：

1. JSONP：利用script标签可跨域的特点，在跨域脚本中可以直接回调当前脚本的函数
2. CORS：服务器设置HTTP响应头中Access-Control-Allow-Origin值，解除跨域限制

jquery实现回车键事件

键盘事件有3：keydown、keypress、keyup，分别是按下、按着没上抬、上抬键盘

$(document).keyup(function(event){

if(event.keyCode===13){

$(“#submit”).trigger(“click”);

}

});

js返回上一页

history.go(-1);

history.back(-1);

window.history.go(-1);

点击未来元素

$(document).on(“click”, “#testDiv”,function(){

//此处的$(this)指“#testDiv”，而不是$(document)

});

JS阻止冒泡和取消默认事件（默认行为）

**防止冒泡和捕获**：w3c的方法是e.stopPropagation，IE则是e.cancelBubble=true

阻止冒泡：window.event?window.event.cancelBubble=true:e.stopPropagation();

**取消默认事件**：w3c的方法是e.preventDefault()，IE则是e.returnValue=false;

当需要停止冒泡行为时，可以使用：

function stopBubble(e){

if(e&&e.stopPropagation){

e.stopPropagation();

}else{

window.event.cancelBubble=true;

}

}

当需要阻止默认行为时，可以使用：

function stopDefault(e){

if(e&&e.preventDefault){

e.preventDefault();

}else{

window.event.returnValue=false;

}

return false;

}