**React + Vue面试题集锦**

# 1.react面试题

## 1.1 react和vue的区别

### 1.1.1 相同点：

1）都支持服务端渲染

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） 数据驱动视图

4）都有支持native的方案，React的React native，Vue的weex

### 1.1.2 不同点：

1） React严格上只针对MVC的view层，Vue则是MVVM模式

2） virtual DOM 不一样 vue会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。而对于React而言，每当应用的状态被改变时，全部子组件都会重新渲染。当然，这可以通过shouldComponentUpdate这个生命周期方法来进行控制，

3） 组件写法不一样 React 推荐的做法是 JSX + inline style，也就是把 HTML 和 CSS 全都写进 JavaScript 了，即”all in js” Vue 推荐的是使用 webpack + vue-loader 的单文件组件格式，即html,css,js写在同一个文件；

4） 数据绑定：Vue有实现了双向数据绑定，React数据流动是单向的

5） state对象在react应用中是不可变的，需要使用setState方法更新状态；在Vue中，state对象并不是必须的，数据由data属性在Vue对象中进行管理。

## 1.2 react的优缺点

### 1.2.1 优点：

1. 可以通过函数式方法描述视图组件（好处：相同的输入会得到同样的渲染结果，不会有副 作 用；组件不会被实例化，整体渲染性能得到提升）

2. 集成虚拟DOM（性能好）

3. 单向数据流（好处是更容易追踪数据变化排查问题

4. 一切都是component：代码更加模块化，重用代码更容易，可维护性高

5. 大量拥抱 es6 新特性

6: jsx

### 1.2.2 缺点：

1. jsx的一个问题是，渲染函数常常包含大量逻辑，最终看着更像是程序片段，而不是视觉呈现。后期如果发生需求更改，维护起来工作量将是巨大的

2. 大而全，上手有难度

## 1.3 jsx的优缺点

允许使用熟悉的语法来定义HTML元素树 JSX 让小组件更加简单、明了、直观。 更加语义化且易懂的标签 JSX 本质是对JavaScript语法的一个扩展，看起来像是某种模板语言，但其实不是。但正因为形似HTML，描述UI就更直观了，也极大地方便了开发； 在React中babel会将JSX转换为React.createElement函数调用，然后将JSX转换为正确的JSON对象（VDOM 也是一个“树”形的结构） React/JSX乍看之下，觉得非常啰嗦，但使用JavaScript而不是模板语法来开发（模板语法比较有局限性），赋予了开发者许多编程能力。

## 1.4 dom diff算法和虚拟DOM

1） React中的render方法，返回一个DOM描述，结果仅仅是轻量级的js对象。Reactjs只在调用setState的时候会更新dom，而且还是先更新Virtual Dom，然后和实际DOM比较，最后再更新实际DOM。

1. React.js 厉害的地方并不是说它比 DOM 快（这句话本来就是错的），而是说不管你数据怎么变化，我都可以以最小的代价来更新 DOM。方法就是我在内存里面用新的数据刷新一个虚拟的 DOM 树，然后新旧 DOM 树进行比较，找出差异，再更新到真正的 DOM 树上。

1. 当我们修改了DOM树上一些节点对应绑定的state，React会立即将它标记为“脏状态”。在事件循环的最后才重新渲染所有的脏节点。在实际的代码中，会对新旧两棵树进行一个深度优先的遍历，这样每个节点都会有一个唯一的标记，每遍历到一个节点就把该节点和新的的树进行对比。如果有差异的话就记录到一个对象里面，最后把差异应用到真正的DOM树上。 算法实现 1 步骤一：用JS对象模拟DOM树 2 步骤二：比较两棵虚拟DOM树的差异 3 步骤三：把差异应用到真正的DOM树上 这就是所谓的 diff 算法
2. dom diff采用的是增量更新的方式，类似于打补丁。React 需要为数据添加 key 来保证虚拟 DOM diff 算法的效率。key属性可以帮助React定位到正确的节点进行比较，从而大幅减少DOM操作次数，提高了性能。

1. virtual dom，也就是虚拟节点。它通过JS的Object对象模拟DOM中的节点，然后再通过特定的render方法将其渲染成真实的DOM节点。 <http://react-china.org/t/dom/638>

## 1.5 为什么js对象模拟DOM会比js操作DOM来得快

为了解决频繁操作DOM导致Web应用效率下降的问题，React提出了“虚拟DOM”（virtual DOM）的概念。Virtual DOM是使用JavaScript对象模拟DOM的一种对象结构。DOM树中所有的信息都可以用JavaScript表述出来，例如：

<ul>

<li>Item 1</li>

<li>Item 2</li>

<li>Item 3</li>

</ul>

可以用以下JavaScript对象来表示：

{

tag: 'ul',

children: [{

tag: 'li', children: ['Item 1'],

tag: 'li', children: ['Item 2'],

tag: 'li', children: ['Item 3']

}]

}

这样可以避免直接频繁地操作DOM，只需要在js对象模拟的虚拟DOM进行比对，再将更改的部分应用到真实的DOM树上

## 1.6 react组件性能优化(通信方式)

1.兄弟组件不能直接相互传送数据，此时可以将数据挂载在父组件中，由两个组件共享

2.子组件向父组件通讯，可以通过父组件定义事件（回调函数），子组件调用该函数，通过实参的形式来改变父组件的数据来通信

//子组件

this.props.onCommentSubmit({author, content, date:new Date().getTime()});

//父组件

render(){

return(

<div className="m-index">

<div>

<h1>评论</h1>

</div>

<CommentList data={this.state.data} />

<CommentForm onCommentSubmit={this.handleCommentSubmit.bind(this)} />

</div>

)

}

3.非父子组件间的通信：可以使用全局事件来实现组件间的沟通，React中可以引入eventProxy模块，利用eventProxy.trigger()方法发布消息，eventProxy.on()方法监听并接收消息。

1. 组件间层级太深，可以使用上下文方式，让子组件直接访问祖先的数据或函数，通过this.context.xx

## 1.7 无状态组件

无状态组件其实本质上就是一个函数，传入props即可，没有state，也没有生命周期方法。组件本身对应的就是render方法。例子如下：

function Title({color = 'red', text = '标题'}) {

let style = {

'color': color

}

return (

<div style = {style}>{text}</div>

)

}

无状态组件不会创建对象，故比较省内存。没有复杂的生命周期方法调用，故流程比较简单。没有state，也不会重复渲染。它本质上就是一个函数而已。

对于没有状态变化的组件，React建议我们使用无状态组件。总之，能用无状态组件的地方，就用无状态组件。

## 1.8 高阶组件

高阶组件（HOC）是函数接受一个组件，返回一个新组件。其前身其实是用ES5创建组件时可用的mixin方法，但是在react版本升级过程中，使用ES6语法创建组件时，认为mixin是反模式，影响了react架构组件的封装稳定性，增加了不可控的复杂度，逐渐被HOC所替代。 实现高阶组件的方式有：

属性代理

import React, { Component } from 'React';

//高阶组件定义

const HOC = (WrappedComponent) =>

class WrapperComponent extends Component {

render() {

return <WrappedComponent {...this.props} />;

}

}

//普通的组件

class WrappedComponent extends Component{

render(){

//....

}

}

//高阶组件使用

export default HOC(WrappedComponent)

反向继承

反向继承是指返回的组件去继承之前的组件(这里都用WrappedComponent代指)

const HOC = (WrappedComponent) =>

class extends WrappedComponent {

render() {

return super.render();

}

}

我们可以看见返回的组件确实都继承自WrappedComponent,那么所有的调用将是反向调用的(例如:super.render())，这也就是为什么叫做反向继承。

## 1.9 react事件和传统事件有什么区别吗

### 1.9.1 react事件

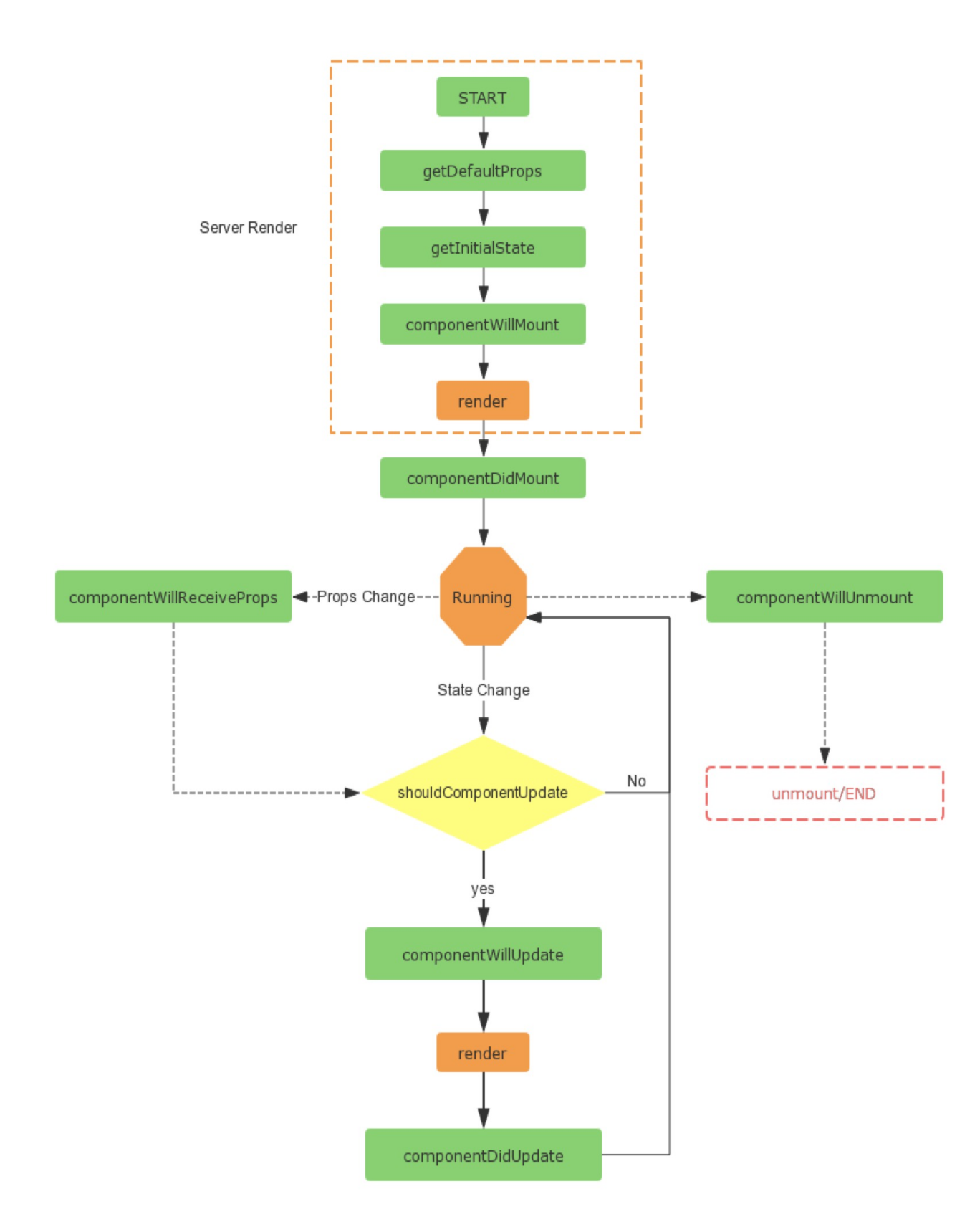
React 实现了一个“合成事件”层（synthetic event system），这个事件模型保证了和 W3C 标准保持一致，所以不用担心有什么诡异的用法，并且这个事件层消除了 IE 与 W3C 标准实现之间的兼容问题。

### 1.9.2 合成事件”还提供了额外的好处：事件委托

“合成事件”会以事件委托（event delegation）的方式绑定到组件最上层， 并且在组件卸载（unmount）的时候自动销毁绑定的事件

## 1.10 react的生命周期图

1.组件更新过程



react组件更新过程：

props/state change：

1.componentWillReceiveProps(nextProps)

只要是父组件的render被调用，在render中被渲染的子组件就会经历更新的过程。不管父组件传给子组件的props有没有改变，都会触发子组件的此函数被调用。注意：通过setState方法触发的更新不会调用此函数

1. shouldComponentUpdate(nextProps,nextState) 3.componentWillUpdate 4.render 5.componentDidUpdate

## 1.11 React中调用setState之后发生了什么事情?

1) React会将当前传入的参数对象与组件当前的状态合并,然后触发调和过程,在调和的过程中,React会以相对高效的方式根据新的状态构建React元素树并且重新渲染整个UI界面  
2) React得到的元素树之后,React会自动计算出新的树与老的树的节点的差异,然后根据差异对界面进行最小化的渲染,在React的差异算法中,React能够精确的知道在哪些位置发生看改变以及应该如何去改变,这样就保证了UI是按需更新的而不是重新渲染整个界面.

## 1.12 React中Element(虚拟DOM对象)与Component的区别?

1. 组件可以生成虚拟DOM对象  
2. 组件它最终返回的是虚拟DOM对象的集合。  
3. ReactElement是描述屏幕上所见的内容的数据结构,是对于UI的对象的表述  
4. ReactComponent则是可以接收参数输入并且返回某个ReactElement的函数或者类.

## 1.13 在什么情况下会优先选择使用ClassComponent而不是FunctionalComponent?

1) 定义复杂组件的时候  
2) 定义组件内部需要设置状态的时候

## 1.14 受控组件(Controlled Component) 与 非受控组件(Uncontrolled Component) 之间的区别是什么？

1) 受控组件，受控组件定义上更多的是针对表单项  
2) 受控组件： 将表单项内的数据交由组件内部管理，通常是放在组件的state状态中，通过修改状态去控制表单项的显示  
3) 非受控组件内部的内容通常直接放在DOM元素中，通常通过refs去管理  
4) 看上去非受控组件比受控组件更为简单， 但还是提倡使用受控组件，因为 **页面与数据分离**

## 1.15 React中keys的作用是什么?

1) key是react中列表元素对应的唯一值。  
2) Keys 是React在操作列表中元素被修改,添加,或者删除的辅助标识.  
3) 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ReactDiff算法中React会借助元素的Key值来判断该元素是新创建的还是被移动而来的元素,React会保存这个辅助状态,从而减少不必要的元素渲染.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,因此我们在开发中不可忽视Key值的使用.  
4) <font color=red>平时设定key的唯一值得时候使用的遍历数组结构的index，注意当新添加的元素中有表单项且变化之前表单项有内容的时候，表单项会根据key的值记录变化之前内部输入的值，此时应该自己去设定key的唯一值</font>

## 1.16 redux管理数据的机制

1. redux是一个独立专门用于做状态管理的JS库(不是react插件库)
2. 它可以用在react, angular, vue等项目中, 但基本与react配合使用
3. 作用: 管理react应用中多个组件共享的状态
4. Redux核心思想：store对象负责调度管理状态并和组件进行对接
5. Redux核心管理流程：

Redux初始化的时候调用reducer获取初始化状态，当组件的状态需要改变的时候可以将需要改变的状态方式和内容通知redux，redux接收到信号后调用dispatch去分发一个对象，该对象为action对象，其中包含的了改变状态的方式和参与改变状态的数据(可能没有数据)， 当dispatch分发对象之后会触发reducer函数的调用，reducer函数是专门用来服务于stroe对象的，reducer函数处理完状态数据，将数据交给store对象，store对象再将状态数据交给对应的组件，从而进行对应的页面渲染。

# 2. vue

## 2.1 vue 双向绑定底层实现原理

vue.js 采用数据劫持的方式，结合发布者-订阅者模式，通过Object.defineProperty()来劫持各个属性的setter，getter以监听属性的变动，在数据变动时发布消息给订阅者，触发相应的监听回调：

https://github.com/hawx1993/tech-blog/issues/11

## 2.2 vue 虚拟DOM和react 虚拟DOM的区别

在渲染过程中，会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。而对于React而言，每当应用的状态被改变时，全部子组件都会重新渲染。 在 React 应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。 如要避免不必要的子组件的重新渲染，你需要在所有可能的地方使用 PureComponent，或是手动实现shouldComponentUpdate 方法

在React中，数据流是自上而下单向的从父节点传递到子节点，所以组件是简单且容易把握的，子组件只需要从父节点提供的props中获取数据并渲染即可。如果顶层组件的某个prop改变了，React会递归地向下遍历整棵组件树，重新渲染所有使用这个属性的组件。

## 2.3 v-show和v-if区别

与v-if不同的是，无论v-show的值为true或false，元素都会存在于HTML代码中；而只有当v-if的值为true，元素才会存在于HTML代码中

## 2.4 vue组件通信

非父子组件间通信，Vue 有提供 Vuex，以状态共享方式来实现同信，对于这一点，应该注意考虑平衡，从整体设计角度去考量，确保引入她的必要。

父传子: this.$refs.xxx 子传父: this.$parent.xxx

还可以通过$emit方法出发一个消息，然后$on接收这个消息

## 2.5 你如何评价vue

框架能够让我们跑的更快，但只有了解原生的JS才能让我们走的更远。

vue专注于MVVM中的viewModel层，通过双向数据绑定，把view层和Model层连接了起来。核心是用数据来驱动DOM。这种把directive和component混在一起的设计有一个非常大的问题，它导致了很多开发者滥用Directive（指令），出现了到处都是指令的情况。

优点： 1.不需要setState，直接修改数据就能刷新页面，而且不需要react的shouldComponentUpdate就能实现最高效的渲染路径。 2.渐进式的开发模式，模版方式->组件方式->路由整合->数据流整合->服务器渲染。上手的曲线更加平滑简单，而且不像react一上来就是组件全家桶 3.v-model给开发后台管理系统带来极大的便利，反观用react开发后台就是个杯具 4.html，css与js比react更优雅地结合在一个文件上。

缺点：指令太多，自带模板扩展不方便； 组件的属性传递没有react的直观和明显

## 2.6 说说你对MVVM的理解

Model层代表数据模型，可以在Model中定义数据修改和操作业务逻辑； view 代表UI组件。负责将数据模型转换成UI展现出来 ViewModel 是一个同步View和Model的对象

用户操作view层，view数据变化会同步到Model，Model数据变化会立即反应到view中。viewModel通过双向数据绑定把view层和Model层连接了起来

## 2.7 为什么选择vue

reactjs 的全家桶方式，实在太过强势，而自己定义的 JSX 规范，揉和在 JS 的组件框架里，导致如果后期发生页面改版工作，工作量将会巨大。

vue的核心：数据绑定 和 视图组件。

Vue的数据驱动：数据改变驱动了视图的自动更新，传统的做法你得手动改变DOM来改变视图，vuejs只需要改变数据，就会自动改变视图，一个字：爽。再也不用你去操心DOM的更新了，这就是MVVM思想的实现。

视图组件化：把整一个网页的拆分成一个个区块，每个区块我们可以看作成一个组件。网页由多个组件拼接或者嵌套组成

## 2.8 vue中mixin与extend区别

全局注册混合对象，会影响到所有之后创建的vue实例，而Vue.extend是对单个实例进行扩展。

mixin 混合对象（组件复用）

同名钩子函数（bind，inserted，update，componentUpdate，unbind）将混合为一个数组，因此都将被调用，混合对象的钩子将在组件自身钩子之前调用

methods，components，directives将被混为同一个对象。两个对象的键名（方法名，属性名）冲突时，取组件（而非mixin）对象的键值对

## 2.9 双向绑定和单向数据绑定的优缺点

只有 UI控件 才存在双向，非 UI控件 只有单向。 单向绑定的优点是可以带来单向数据流，这样的好处是流动方向可以跟踪，流动单一，没有状态, 这使得单向绑定能够避免状态管理在复杂度上升时产生的各种问题, 程序的调试会变得相对容易。单向数据流更利于状态的维护及优化，更利于组件之间的通信，更利于组件的复用

双向数据流的优点：

无需进行和单向数据绑定的那些CRUD（Create，Retrieve，Update，Delete）操作； 双向绑定在一些需要实时反应用户输入的场合会非常方便 用户在视图上的修改会自动同步到数据模型中去，数据模型中值的变化也会立刻同步到视图中去；

缺点：

双向数据流是自动管理状态的, 但是在实际应用中会有很多不得不手动处理状态变化的逻辑, 使得程序复杂度上升 无法追踪局部状态的变化 双向数据流，值和UI绑定，但由于各种数据相互依赖相互绑定，导致数据问题的源头难以被跟踪到

Vue 虽然通过 v-model 支持双向绑定，但是如果引入了类似redux的vuex，就无法同时使用 v-model。

双绑跟单向绑定之间的差异只在于，双向绑定把数据变更的操作隐藏在框架内部，调用者并不会直接感知。

<input v-model="something">

<!-- 等价于以下内容 -->

<input :value="something" @input="something = $event.target.value">

也就是说，你只需要在组件中声明一个name为value的props，并且通过触发input事件传入一个值，就能修改这个value。

## 2.10 VueJS 开发常见问题集锦

https://segmentfault.com/a/1190000010230843

## 2.11谈谈你对组件的理解

一个组件应该有以下特征：

1. 可组合（Composeable）：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（own）它创建的子组件，通过这个特性，一个复杂的 2. UI 可以拆分成多个简单的 UI 组件；

3. 可重用（Reusable）：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（Maintainable）：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. 可测试（Testable）：因为每个组件都是独立的，那么对于各个组件分别测试显然要比对于整个 UI 进行测试容易的多。

### 18.简述前端优化的方式 旧的雅虎34条|h5新添加的方式

1、尽量减少HTTP请求次数  
2、减少DNS查找次数  
3、避免跳转  
4、可缓存的AJAX  
5、推迟加载内容  
6、预加载  
7、减少DOM元素数量  
8、根据域名划分页面内容  
9、使iframe的数量最小  
10、不要出现404错误  
11、使用内容分发网络  
12、为文件头指定Expires或Cache-Control 13、Gzip压缩文件内容  
14、配置ETag  
15、尽早刷新输出缓冲  
16、使用GET来完成AJAX请求  
17、把样式表置于顶部  
18、避免使用CSS表达式（Expression）  
19、使用外部JavaScript和CSS  
20、削减JavaScript和CSS  
21、用<link>代替@import  
22、避免使用滤镜  
23、把脚本置于页面底部  
24、剔除重复脚本