



前端面试夜话



关于我

慕课网讲师，擅长各种源码

- ◆ <https://www.imooc.com/t/135647>
- ◆ 8年前端老玩家
- ◆ 皇室战争4300杯王者荣耀黄金1半职业玩家

大纲

大厂面试方方面面

- ◆ 前端知识图谱
- ◆ 一面二面三面
- ◆ 软技能

面试

让面试官半小时内，觉得你牛逼

- ◆ 所以面试的奇技淫巧意义不大
- ◆ 重点在平时修炼
- ◆ 掌握必要的软技能，多拿钱



01

• 前端基础

02

• 多端 (Node , 小程序 , App)

03

• 软件工程师

04

• 软技能

01 前端基础

- ◆ html+css基础布局
- ◆ webpack
- ◆ 异步
- ◆ ES6
- ◆ React.js
- ◆ 浏览器
- ◆ Typescript
- ◆ Vuejs
- ◆ 性能优化
- ◆ 安全

HTML+CSS布局

倔强青铜必备

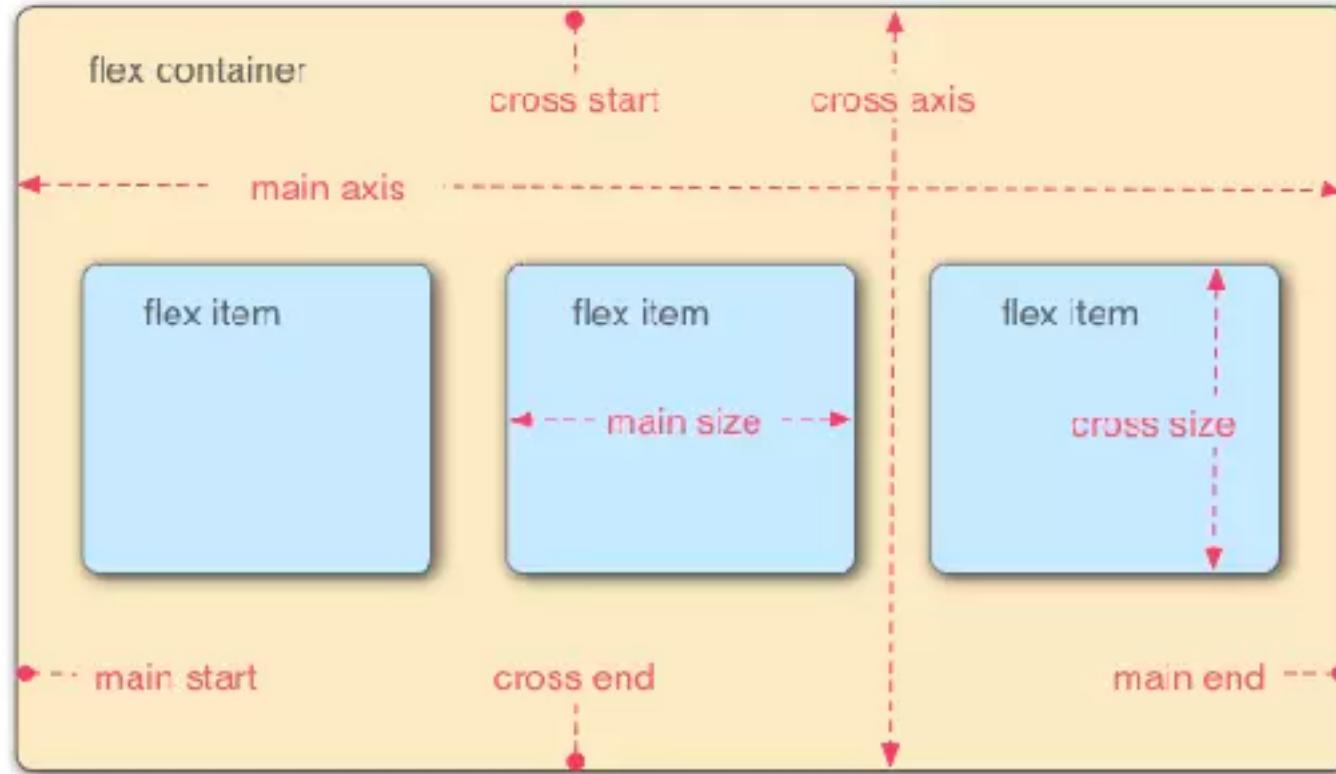
- ◆ 内容+样式 属于基础
- ◆ 重点理解布局、流
- ◆ less、sass、stylus

CSS

- ◆ px、em、rem(root em)、vw、vh
- ◆ 设置html的font-size大小，统一标准
- ◆ 小程序单位rpx

布局

- ◆ 相对、绝对
- ◆ 流式 响应式布局
- ◆ flex布局



CSS3

- ◆ 透明度、文字阴影、圆角、渐变色
- ◆ 盒阴影，边框图片，媒体查询
- ◆ transition过度，transform变形，animation动画

CSS优化

- ◆ 了解选择器原理，从右向左查询
- ◆ 避免后台、链式、重复
- ◆ 避免!Important

BFC & IFC

- ◆ float不是none, 绝对定位，表格，overflow不是visible ，flex盒子
- ◆ 创建BFC后，元素会一个个的摆放
- ◆ 外边局折叠

最常见的就是`overflow:hidden`、`float:left/right`、`position:absolute`



精通CSS (第2版) : 高级Web标准解决方案

★★★★★ 8.6 (699人评价)

[英] Ancy Budd / [英] Simon Collison / [英] Cameron Moll /



CSS世界

★★★★★ 7.8 (127人评价)

张鑫旭 / 人民邮电出版社 / 2017-12 / CNY 69.00



全面系统讲解CSS 工作应用+面试 一步搞定

初级 ■ 1020

评价: 9.64分

CSS是前端的看家本领之一，从实际工作需求角度出发助你吃透CSS知识体系！

¥149.00

★ 收藏 | 加购物车

ES6 & ES7

业界标准

- ◆ let、const、解构
- ◆ 箭头函数、Symbol、Set
- ◆ Proxy、class、generator、模块化



如何统计网页里出现多少种html标签

```
// 获取所有标签
var doms = document.getElementsByTagName("*")

// 去重
var obj = {}
var ret = []
for (var j = 0; j < doms.length; j++) {
    var name = doms[j].nodeName
    if(!obj[name]){
        ret.push(name)
        obj[name] = true
    }
}
console.log(ret.length)

// es6

const names = [...document.getElementsByTagName("*")].map(v=>v.nodeName)
console.log( new Set(names).size)
```

函数

早晨起床 拥抱函数

- ◆ 默认参数，返回值、原型链
- ◆ 箭头函数、this、作用域、闭包、bind、apply
- ◆ 高阶函数，递归、Decorator，Compose，Currying。。。

面向对象

原型链的语法糖

- ◆ class、实例方法、静态属性和方法
- ◆ 构造函数、super、
- ◆ 继承

JS异步

解决回调地狱

- ◆ Promise
- ◆ Generator
- ◆ Async + await

模块化

最终解决方案

- ◆ 最早的AMD CMD (requirejs , seajs) commonjs
- ◆ import export
- ◆ ES6加载原理和node加载原理

面试题：小老弟 能手写Promise吗

Promise相当于一个状态机

- ◆ pending、rejected、fulfilled状态
- ◆ 维护callback队列，
- ◆ <https://juejin.im/post/5c41297cf265da613356d4ec>

面试题：小老弟 能手写bind call吗

```
Function.prototype.call = function(context) {
    context = context || window
    context.fn = this
    const args = [...arguments].slice(1)
    const result = context.fn(...args)
    delete context.fn
    return result
}
```

```
Function.prototype.bind = function (context) {
    const _this = this
    const args = [...arguments].slice(1)
    // 返回一个高阶函数函数
    return function F() {
        if (this instanceof F) {
            return new _this(...args, ...arguments)
        }
        return _this.apply(context, args.concat(...arguments))
    }
}
```



深入理解ES6

★★★★★ 9.4 (155人评价)

【美】Nicholas C. Zakas / 刘振涛 / 电子工业出版社 / 2017-7-1 / CNY 99.00



ES6 标准入门 (第2版)

★★★★★ 8.2 (190人评价)

阮一峰 / 电子工业出版社 / 2016-1 / 69.00元



ES6零基础教学 解析彩票项目

中级

2187

评价: 9.49分

ES6从零开始，量身设计的迷你案例，让你全面掌握ES6

¥ 188.00

★ 收藏 | 加购物车

满499减50



你不知道的JavaScript（上卷）

★★★★★ 9.4 (572人评价)

[美] Kyle Simpson / 赵望野 / 梁杰 / 人民邮电出版社 / 2015-4 / 49.00元



你不知道的JavaScript（中卷）

★★★★★ 8.9 (188人评价)

[美] Kyle Simpson / 单业 / 姜南 / 人民邮电出版社 / 2016-8 / 79.00元



你不知道的JavaScript（下卷）

★★★★★ 8.0 (57人评价)

[美] Kyle Simpson / 单业 / 人民邮电出版社 / 2018-1-1 / 79.00

Vue

最易用的前端框架，官网就是最好的教程

- ◆ Vue-cli3
- ◆ 组件化通信
- ◆ 生命周期

组件通信

不同的component怎么唠嗑

- ◆ 父子关系 props \$emit 兄弟可以让父母代理中转
- ◆ 祖先后代关系 eventbus或者自己实现dispatch & broadcast
- ◆ 没啥关系 eventbus 或者vuex

```
// 嘴角向上
function dispatch(componentName, eventName, params) {
  let parent = this.$parent || this.$root;
  let name = parent.$options.name;
  while (parent && (!name || name !== componentName)) {
    parent = parent.$parent;
    if (parent) {
      name = parent.$options.name;
    }
  }
  if (parent) {
    parent.$emit.apply(parent, [eventName].concat(params));
  }
}
```

```
// 嘴角向下
function broadcast(componentName, eventName, params) {
  this.$children.forEach(child => {
    const name = child.$options.name;

    if (name === componentName) {
      child.$emit.apply(child, [eventName].concat(params));
    } else {
      broadcast.apply(child, [componentName, eventName].concat([params]));
    }
  });
}
```



Vue

最易用的前端框架，官网就是最好的教程

- ◆ 组件化设计
- ◆ 源码
- ◆ 全家桶、服务端渲染

Vue组件化

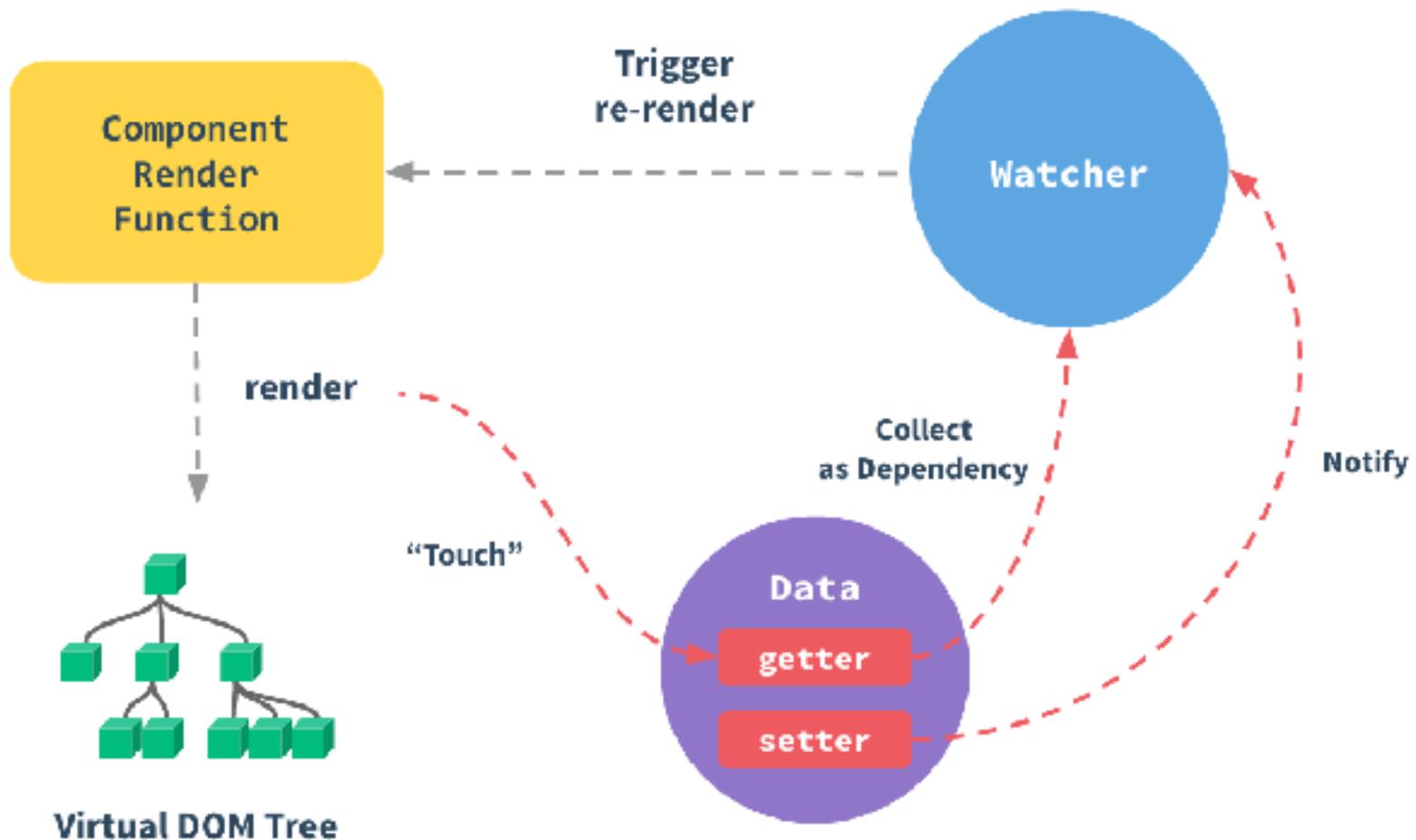
复用.Vue组件

- ◆ props. event. slot三大件
- ◆ 手动挂载的组件，\$mount
- ◆ 递归组件和动态组件

Vue原理

双向绑定 <https://cn.vuejs.org/v2/guide/reactivity.html>

- ◆ Object.defineProperty
- ◆ 依赖收集
- ◆ 异步更新队列



```
var obj = {};
Object.defineProperty(obj, "name", {
    get: function () {
        console.log('获取name')
        return document.querySelector('#name').innerHTML;
    },
    set: function (nick) {
        console.log('设置name')
        document.querySelector('#name').innerHTML = nick;
    }
});
```

```
obj.name = "imooc";
console.log(obj.name)
```

```
class Vue {
  constructor(options) {
    this._data = options.data;
    this.observer(this._data);
  }
  observer(value) {
    if (!value || typeof value !== "object") {
      return;
    }
    Object.keys(value).forEach(key => {
      this.defineReactive(value, key, value[key]);
    });
  }
  defineReactive(obj, key, val) {
    Object.defineProperty(obj, key, {
      get() {
        return val;
      },
      set(newVal) {
        if (newVal === val) return;
        this.cb(newVal);
      }
    });
  }
  cb(val) {
    console.log("更新了", val);
  }
}
```

```
class Dep {
    constructor () {
        // 存放所有的依赖
        this.deps = []
    }

    // 在deps中添加一个监听器对象
    addDep (dep) {
        this.deps.push(dep)
    }

    // 通知所有监听器去更新视图
    notify () {
        this.deps.forEach((dep) => {
            dep.update()
        })
    }
}
```

```
class Compile {
    constructor(el,vm) {
        this.$vm = vm
        this.$el = document.querySelector(el)
        if (this.$el) {
            this.$fragment = this.node2Fragment(this.$el)
            this.compileElement(this.$fragment)
            this.$el.appendChild(this.$fragment)
        }
    }
    node2Fragment(el) {
        // 新建文档碎片 dom接口
        let fragment = document.createDocumentFragment()
        let child
        // 将原生节点拷贝到fragment
        while (child = el.firstChild) {
            fragment.appendChild(child)
        }
        return fragment
    }
}
```

```
// 监听器
class Watcher {
    constructor(vm, key, cb) {
        // 然后触发属性的 getter 添加监听
        // 最后将 Dep.target 置空
        this.cb = cb
        this.vm = vm
        this.key = key
        this.value = this.get()
    }
    get() {
        Dep.target = this
        let value = this.vm[this.key]
        return value
    }
    // 更新视图的方法
    update() {
        this.value = this.get()
        this.cb.call(this.vm, this.value)
    }
}
```

```
class Vue {
  constructor(options) {
    this.$observer(this.$data)
    if(options.created){
      options.created.call(this)
    }
    // 编译解析
    this.$compile = new Compile(options.el, this)
  }
  observer(value) {
    if (!value || (typeof value !== 'object')) {
      return
    }
    Object.keys(value).forEach((key) => {
      this.proxyData(key)
      this.defineReactive(value, key, value[key])
    })
  }
  defineReactive(obj, key, val) {
    const dep = new Dep()
    Object.defineProperty(obj, key, {
      get() {
        // 添加依赖
        Dep.Target && dep.addDep(Dep.Target)
        return val
      },
      set(newVal) {
        if (newVal === val) return
        val = newVal
        // 数据更新，触发队列执行更新
        dep.notify()
      }
    })
  }
}
```

Vuex+vue-router

全家桶标配

- ◆ 单页应用 懒加载路由
- ◆ vue单项数据流
- ◆ 数据交给专门的store管理，全局数据中心

服务端渲染

服务端解析vue组件成html 渲染首屏

- ◆ 速度
- ◆ SEO
- ◆ nuxt.js



Vue2.5开发去哪儿网App 从零基础入门到实战项目

中级

▲ 4905

评价: 9.95分

从基础语法到完整项目，一套课程掌握Vue基础知识和项目实践。



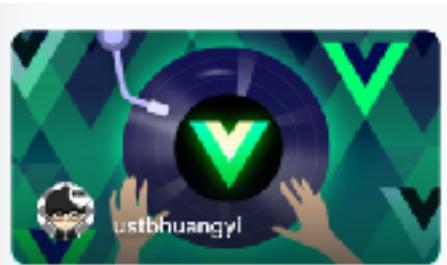
Vue.js 2.5 + cube-ui 重构饿了么App

中级

▲ 7577

评价: 9.76分

掌握Vue1.0到2.0再到2.5完全版本应用与迭代，打造具备国际化WebApp。



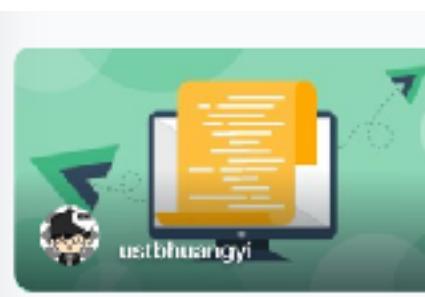
Vue 2.0开发企业级移动端音乐Web APP

高级

▲ 3869

评价: 8.81分

Vue.js高级知识应用大集合，实战企业级APP，教你精通组件化开发。



Vue.js 源码全方位深入解析

高级

▲ 1724

评价: 9.65分

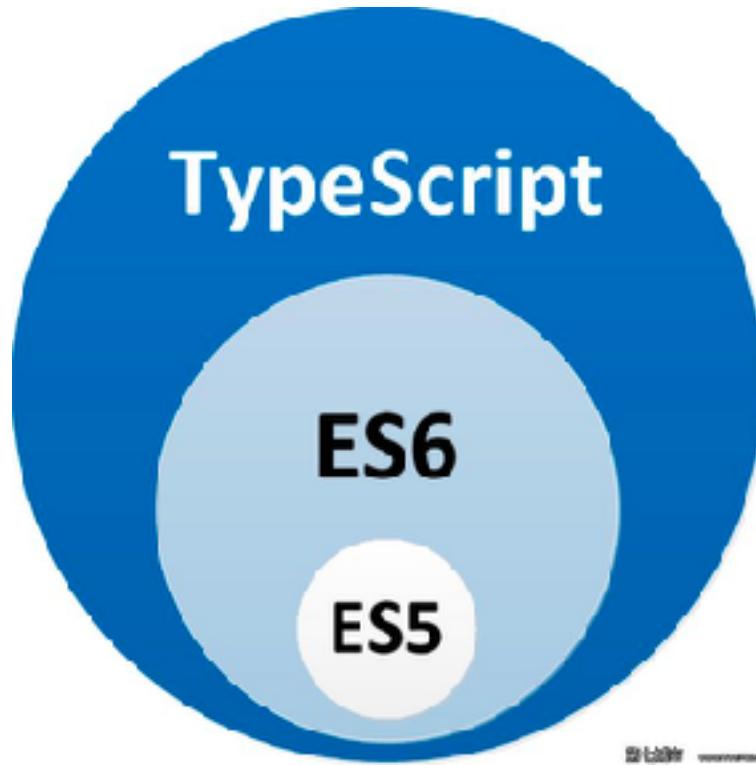
全方位讲解 Vue.js 源码，学精学透 Vue 原型实现，进阶高级工程师。

TypeScript

Vue3 React的最终选择

- ◆ 强类型的好处，ES6的超集
- ◆ Vue3使用TS完全重写
- ◆ React团队停止维护flow，全面拥抱TS

<https://www.imooc.com/learn/763>



React

开天辟地的虚拟DOM

- ◆ 组件化、高阶组件、Hooks、生命周期
- ◆ 源码、fiber
- ◆ 全家桶

组件通信

隔壁Vue都唠半天了，React里组件怎么唠

- ◆ 父子 props
- ◆ 跨层级 context
- ◆ 任意 redux eventBus , Mobx , dva

高阶组件

一个函数，传入组件，返回另外一个组件

- ◆ 属性代理，页面复用
- ◆ 鉴权、日志，性能打点
- ◆ 配合装饰器，贼好用 @connect, @inject等等

Hooks

发布正式版，拥抱函数

- ◆ 函数式组件能力扩展
- ◆ React以后的方向，从此class组件会越来越少
- ◆ 自定义Hooks，组合优于继承

```
class Imooc extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

```
import { useState } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

```
import { useState, useEffect } from 'react';
// 自定义Hooks
function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }

  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });

  return isOnline;
}
```

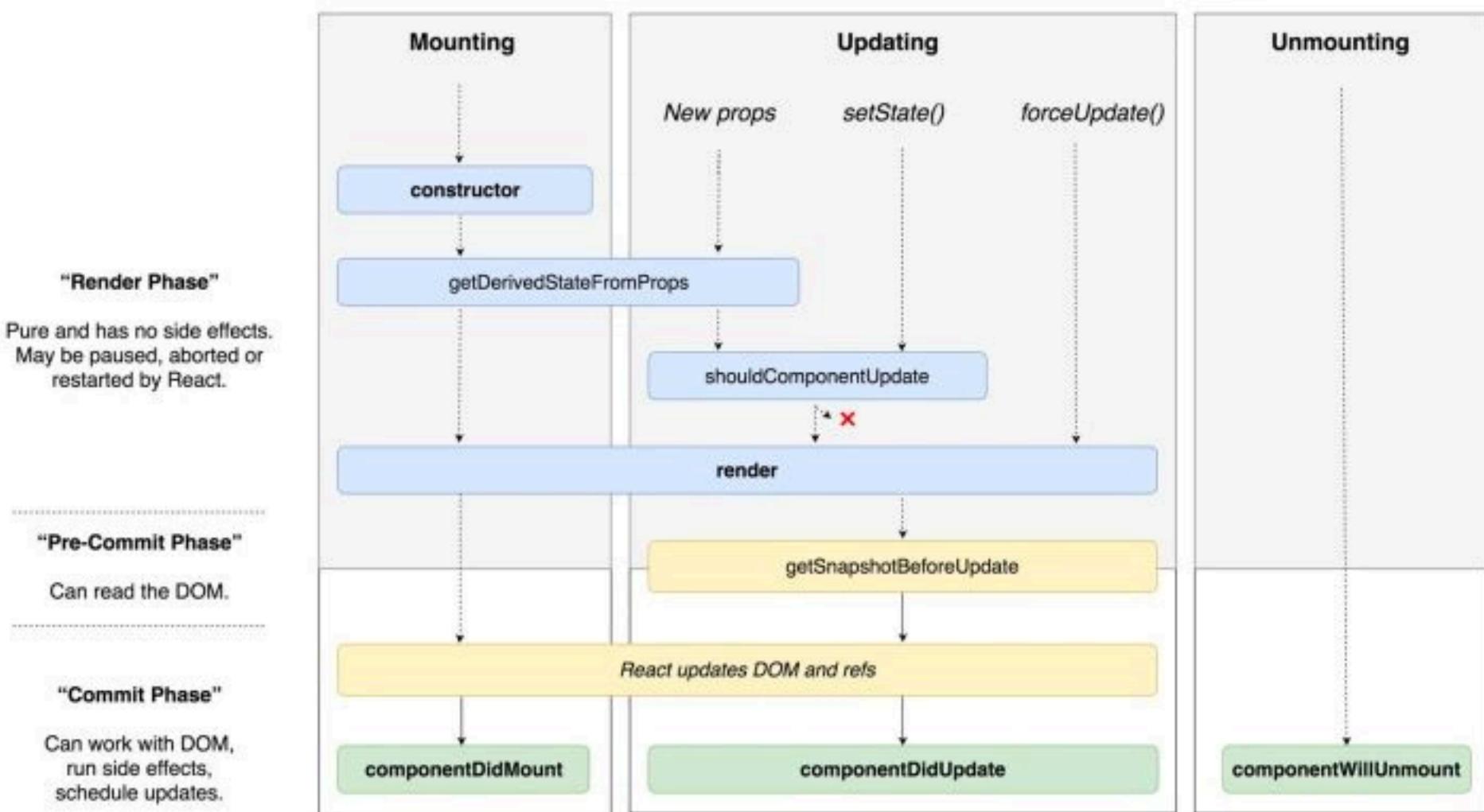
```
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```

生命周期

16版本，生命周期发生了一丢丢的变动

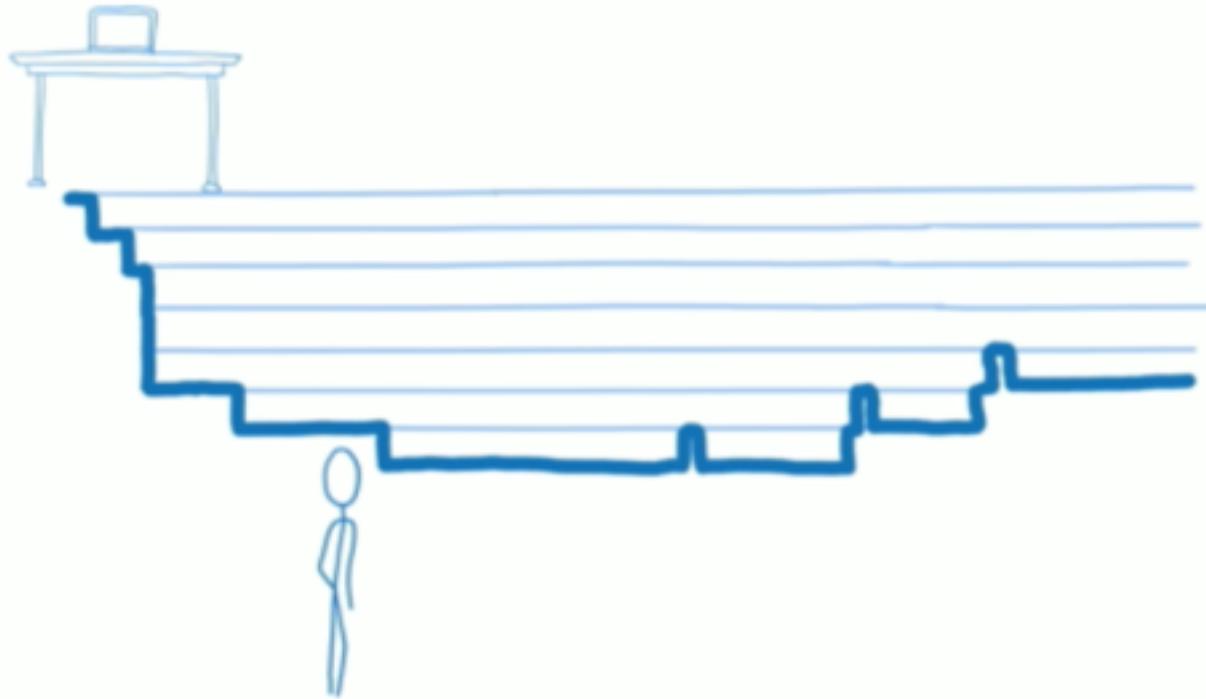
- ◆ 实例化新增getDerivedStateFromProps
- ◆ 更新新增 getDerivedStateFromProps ,
getSnapshotBeforeUpdate
- ◆ componentDidCatch错误处理，弃用willMount,
willReceiveProps,willuPDATE

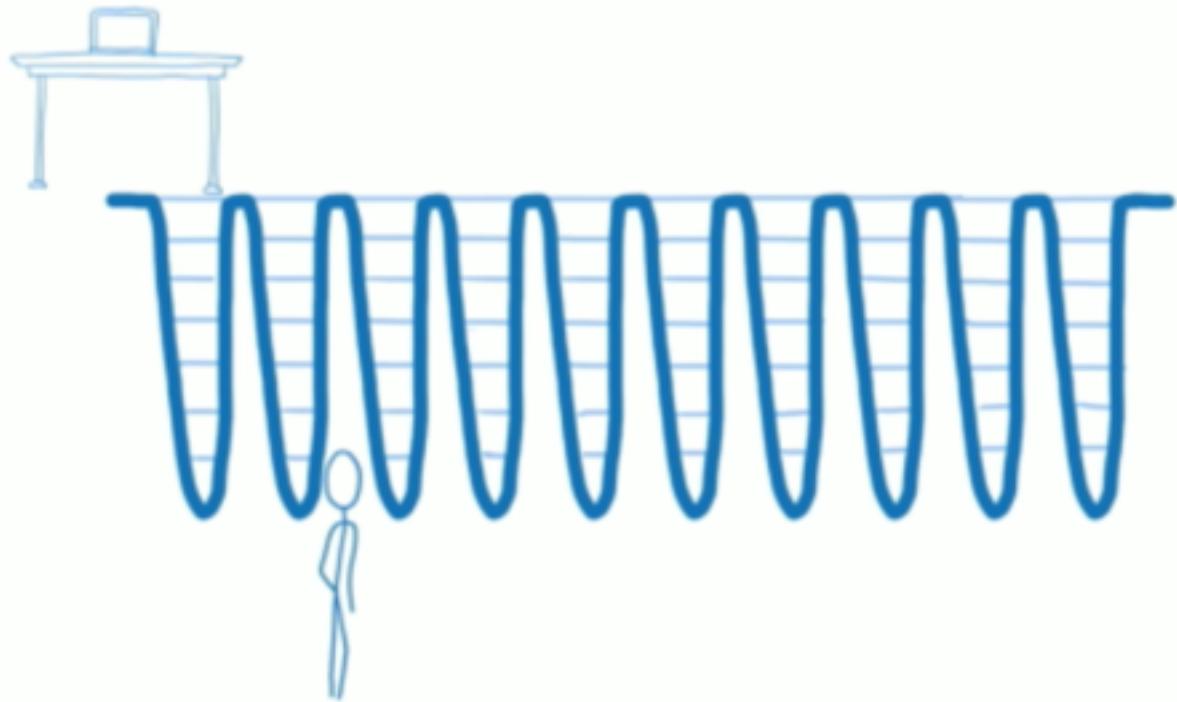


Fiber

React团队重构两年的结果

- ◆ JS单线程种，实现了调度逻辑，任务分解
- ◆ 任务优先级，任务切换
- ◆ 页面更流畅





React原理

虚拟dom , JSX , setState

- ◆ createElement
- ◆ Component
- ◆ Render

```
import Component from './Component'
import createElement from './createElement'

let React = {
  createElement,
  Component
}

export default React
```

```
// 第一个核心api 构建虚拟dom的object
export default function createElement(type, props, ...children) {
  let vtype = null
  if (typeof type === 'string') {
    vtype = VELEMENT
  } else if (typeof type === 'function') {
    if (type && type.isReactComponent) {
      vtype = VCOMPONENT
    } else {
      vtype = VSTATELESS
    }
  } else {
    throw new Error(`React.createElement: unexpect type [ ${type} ]`)
  }
  props.children = children

  return createVnode(vtype, type, finalProps, key, ref)
}
```

```
export function createVnode(vtype, type, props, key, ref) {
    // 创建vnode
    let vnode = {
        vtype: vtype,
        type: type,
        props: props,
        refs: refs,
        key: key,
        ref: ref,
    }
    if (vtype === VSTATELESS || vtype === VCOMPONENT) {
        vnode.uid = _.getUid()
    }
    return vnode
}
```

createElement

JSX被babel编译成createElement

```
const element = <h1>Hello, world!</h1>;
// babel编译为
var element = React.createElement(
  "h1",
  null,
  "Hello, world!"
);
```

Component

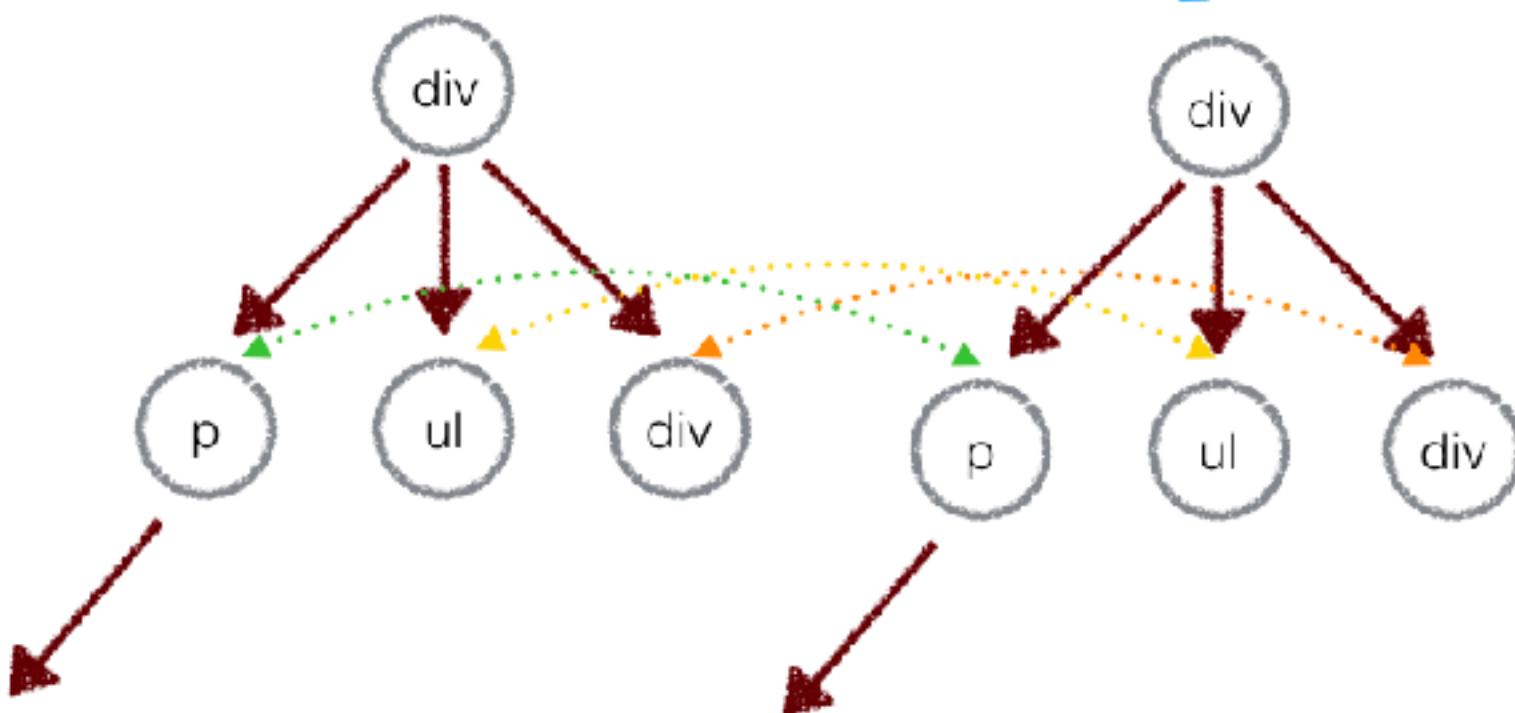
很浅层的封装

- ◆ <https://github.com/facebook/react/blob/master/packages/react/src/ReactBaseClasses.js>
- ◆ 主要提供forceUpdate和setState
- ◆ PureComponent就是设置了shouldComponentUpdate生命周期

setState

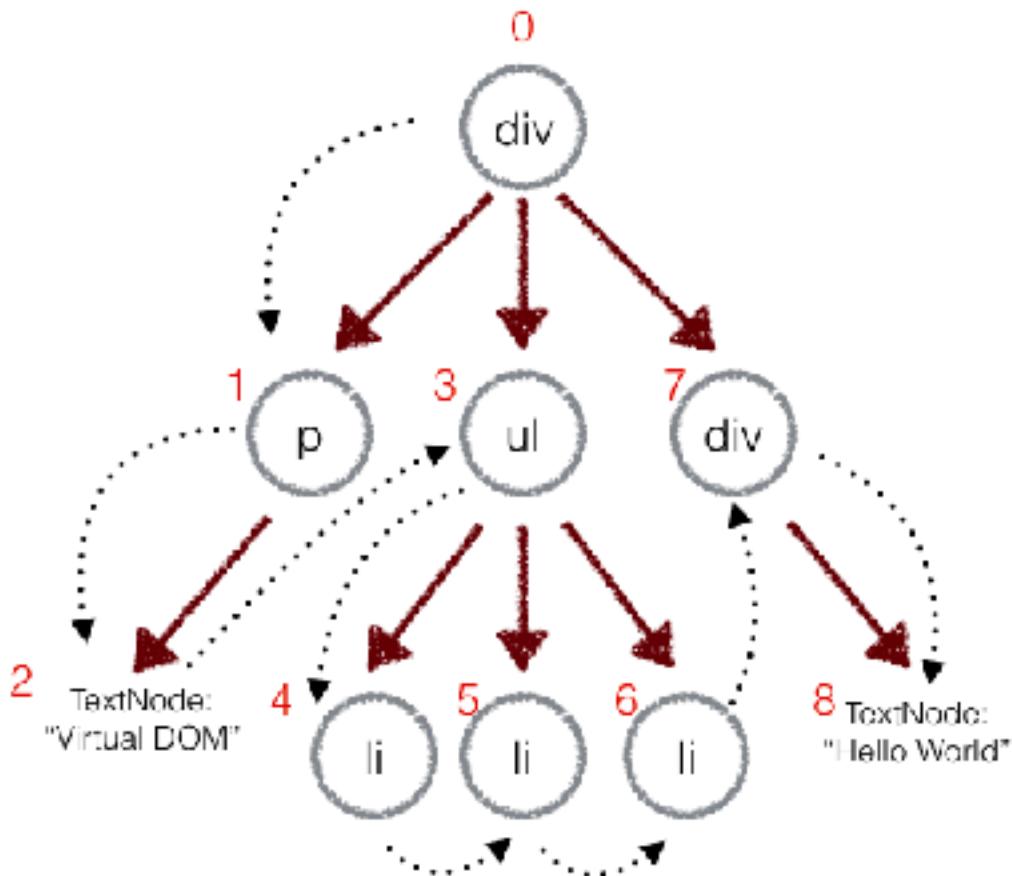
createElement新建虚拟dom，setState生成新的虚拟dom树，开始对比

- ◆ 虚拟dom树平级对比，找到需要修改的patch数组
- ◆ 遍历数组，insert，remove，或者move元素，少操作dom
- ◆ setState异步队列，多次执行，最终合并做一个渲染



TextNode:
"Virtual DOM"

TextNode:
"Virtual DOM"



```
emitUpdate(nextProps, nextContext) {
  this.nextProps = nextProps
  this.nextContext = nextContext
  // receive nextProps!! should update immediately
  nextProps || !updateQueue.isPending
    ? this.updateComponent()
    : updateQueue.add(this)
}

updateComponent() {
  let { instance, pendingStates, nextProps, nextContext } = this
  if (nextProps || pendingStates.length > 0) {
    nextProps = nextProps || instance.props
    nextContext = nextContext || instance.context
    this.nextProps = this.nextContext = null
    // getState 合并所有的state的数据，一次更新
    shouldUpdate(instance, nextProps, this.getState(), nextContext, this.clearCallbacks)
  }
}

addState(nextState) {
  if (nextState) {
    _addItem(this.pendingStates, nextState)
    if (!this.isPending) {
      this.emitUpdate()
    }
  }
}
```

```
export function compareTwoVnodes(vnode, newVnode, node, parentContext) {
    // 虚拟dom对比
    let newNode = node
    if (newVnode == null) {
        // remove
        destroyVnode(vnode, node)
        node.parentNode.removeChild(node)
    } else if (vnode.type !== newVnode.type || vnode.key !== newVnode.key) {
        // replace
        destroyVnode(vnode, node)
        newNode = initVnode(newVnode, parentContext, node.namespaceURI)
        node.parentNode.replaceChild(newNode, node)
    } else if (vnode !== newVnode || parentContext) {
        // same type and same key -> update
        newNode = updateVnode(vnode, newVnode, node, parentContext)
    }
    return newNode
}
```

```
function updateVChildren(vnode, newVnode, node, parentContext) {  
    // 更新children，产出三个patch数组  
    let patches = {  
        removes: [],  
        updates: [],  
        creates: [],  
    }  
    diffVchildren(patches, vnode, newVnode, node, parentContext)  
    _.flatEach(patches.removes, applyDestroy)  
    _.flatEach(patches.updates, applyUpdate)  
    _.flatEach(patches.creates, applyCreate)  
}
```

服务端渲染

和vue一样，有专门的api

- ◆ renderToString
- ◆ vue的nuxt react的next 服务端渲染框架
- ◆ 一切为了性能

redux原理

单项数据流，我的redux课里完全手写了一个

- ◆ 中间件机制
- ◆ connect高阶组件
- ◆ 使用context api

```
export function createStore(reducer, enhancer){
  if (enhancer) {
    return enhancer(createStore)(reducer)
  }
  let currentState = {}
  let currentListeners = []

  function getState(){
    return currentState
  }
  function subscribe(listener){
    currentListeners.push(listener)
  }
  function dispatch(action){
    currentState = reducer(currentState, action)
    currentListeners.forEach(v=>v())
    return action
  }
  dispatch({type: '@IM0OC/WONIU-REDUX'})
  return { getState, subscribe, dispatch}
}
```

```
export function applyMiddleware(...middlewares){
  return createStore=>(...args)=>{
    const store = createStore(...args)
    let dispatch = store.dispatch

    const midApi = {
      getState:store.getState,
      dispatch:(...args)=>dispatch(...args)
    }
    const middlewareChain = middlewares.map(middleware=>middleware(midApi))
    dispatch = compose(...middlewareChain)(store.dispatch)
    // dispatch = middleware(midApi)(store.dispatch)
    // middleware(midApi)(store.dispatch)(action)
    return {
      ...store,
      dispatch
    }
  }
  // middlewares.map...
}
```

```
// compose(fn1,fn2,fn3)
// fn1(fn2(fn3))
export function compose(...funcs){
    if (funcs.length==0) {
        return arg=>arg
    }
    if (funcs.length==1) {
        return funcs[0]
    }
    return funcs.reduce((ret,item)=> (...args)=>ret(item(...args)))
}
// {addGun, removeGun, addGunAsync}
// addGun(参数)
// dispatch(addGun(参数))
function bindActionCreator(creator, dispatch){
    return (...args) => dispatch(creator(...args))
}
export function bindActionCreators(creators,dispatch){
    // let bound = {}
    // Object.keys(creators).forEach(v=>{
    //     let creator = creators[v]
    //     bound[v] = bindActionCreator(creator, dispatch)
    // })
    // return bound
    return Object.keys(creators).reduce((ret,item)=>{
        ret[item] = bindActionCreator(creators[item],dispatch)
        return ret
    },{})
}
```

```
// 高阶组件
export const connect = (mapStateToProps:state=>state, mapDispatchToProps:{}=>{})(WrapComponent)=>{
    return class ConnectComponent extends React.Component{
        static contextTypes = {
            store:PropTypes.object
        }
        constructor(props, context){
            super(props, context)
            this.state = {
                props:{}
            }
        }
        componentDidMount(){
            const {store} = this.context
            store.subscribe(()=>this.update())
            this.update()
        }
        update(){
            // 获得mapStateToProps&mapDispatchToProps 放入this.props里
            const {store} = this.context
            const stateProps = mapStateToProps(store.getState())
            // 方法不能直接换，因为需要dispatch

            // 直接换掉addGun() 意义不大
            // 要 addGun = ()=>store.dispatch(addGun()) 才有意义，其实就是在dispatch把actionCreators缓存了一层
            const dispatchProps = bindActionCreators(mapDispatchToProps, store.dispatch)
            // console.log(stateProps)
            this.setState({
                props:{
                    ...this.state.props,
                    ...stateProps,
                    ...dispatchProps
                }
            })
        }
        render(){
            return <WrapComponent {...this.state.props}></WrapComponent>
        }
    }
}
```

<https://github.com/livoras/blog/issues/13>



Redux+React Router+Node.js全
栈开发

高级 · 1000 评价: 6.20分
全网唯一的一门React+Redux+React Router+
Node.js全栈开发课程，实战项目，学到手是你的真本领！



深入React技术栈

★★★★★ 8.2 (134人评价)

陈屹 / 人民邮电出版社 / 2016-11-1 / CNY 79.00



React 源码深度解析 高级前端工
程师必备技能

高级 · 479 评价: 10.00分
掌握React源码，让你的开发水平没有上限，
让你的前端未来更轻松！

Webpack

工程化巅峰之作

- ◆ 基础配置
- ◆ 性能优化，缩小搜索范围，DllPlugin，多进程、tree-shaking，代码抽取，按需加载
- ◆ 定制loader和plugin

Loader & plugin

进阶webpack必备

- ◆ loader 定义自己的转换规则
- ◆ plugin 整个 webpack 工作流程定义，有一个 apply 方法获取 compiler 对象
- ◆ loader 单一职责，链式组合，模块化，无状态

plugin

进阶webpack必备

- ◆ 修改输出资源
- ◆ 读取模块和依赖
- ◆ 监听文件变化

```
// Loader
function replace(source) {
    // 使用正则把 // @require '../style/index.css' 转换成 require('../style/index.css');
    return source.replace(/(\r\n| *@\r\nrequire) +((\r\n|.+\r\n)|").*/g, 'require($2);');
}

module.exports = function (content) {
    return replace(content);
};
```

```
module.exports = {
  module: {
    loaders: [
      {
        test: /\.js$/,
        loaders: ['comment-require-loader'],
      }
    ]
  }
};
```



<http://webpack.wuhaolin.cn/>

以上例子来源于这本书

Web安全

常见漏洞，如何防御

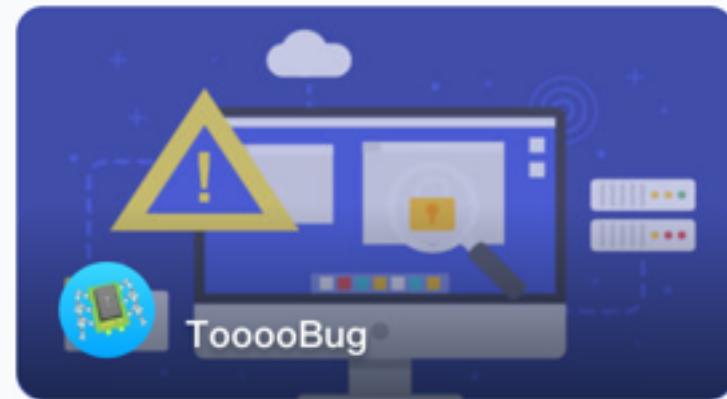
- ◆ XSS、CSRF、Cookie劫持、点击劫持
- ◆ 传输安全、接入层
- ◆ Oauth



Web前端黑客技术揭秘

★★★★★ 7.6 (244人评价)

钟晨鸣 / 徐少培 / 电子工业出版社 / 2013-1 / 59.00元



腾讯大牛亲授 Web 前后端漏洞分析与防御技巧

中级 603

评价：9.91分

提高每一行代码安全系数，突破前后端开发Web安全弱项

性能优化

天下武功，唯快不破

- ◆ 抛开场景谈优化，就是要流氓
- ◆ 常见性能优化策略, 文件少加载，代码少执行，多用缓存，少计算，服务端渲染
- ◆ 性能如何分析 devtools 代码打包压缩，图片压缩，gzip，缓存，cdn，SSR，框架对应的优化策略，lazy-load, 节流防抖



高效前端：Web高效编程与优化实践

★★★★★ 8.6 (17人评价)

李银城 著 / 机械工业出版社 / 2018-3-15 / 89.00元



让你页面速度飞起来 Web前端性能优化

中级 675

评价: 9.76分

【面试必备】采用Vue-SSR和PWA等新技术，
快速突破你的Web页面性能的瓶颈

02 多端 (Node , 小程序 , App)

- ◆ 微信小程序
- ◆ 小程序生态
- ◆ Node.js
- ◆ RN flutter
- ◆ 微服务
- ◆ 测试
- ◆ 监控
- ◆ 自动化
- ◆ 部署
- ◆ 云开发

Node.js

前端入侵后端

- ◆ Node核心概念
- ◆ Express、Koa等web开发
- ◆ 自动化，微服务

Node.js核心概念

不仅仅是会了JS 就会了Node 前端有这么高的薪资，都得感谢node

- ◆ events , fs , stream , buffer
- ◆ IO , event-loop , 线程池
- ◆ libuv , V8

JavaScript

Node Standard Library

C/C++

Node Bindings

(socket, http, file system, etc.)

**Chrome
V8**

(JS engine)

**Async
I/O**

(libuv)

**Event
Loop**

(libuv)

Node后端 02

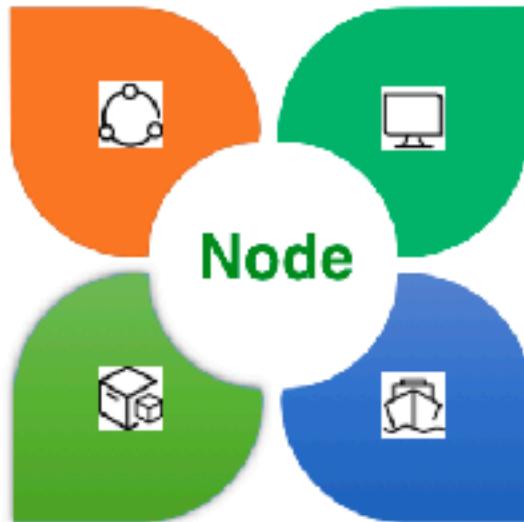
核心特性、Web应用、Api
rpc、测试、部署、最佳实践
微服务、厂商支持

跨平台 01

前端 (web+h5)

移动端 (hybrid)

PC端



前端 03

react\vue\angular

应用实践

架构

工具 04

各种预编译、构建工具
webpack/gulp、工程化，
hack技巧、npm等

Node应用场景

- ◆ express/koa网站
- ◆ 实时/[socket.io](#)
- ◆ 跨平台/ electron
- ◆ 微服务
- ◆ API/hapi
- ◆ 前端工具
webpack/ gulp
- ◆ 区块链/ ipfs
- ◆ 企业级框架eggjs
- ◆ Express/koa代理
- ◆ 命令行工具/
shell.js
- ◆ 硬件/ ruff
- ◆ 爬虫/ cheerio

Event-loop

代码到底咋执行的

- ◆ 同步代码，完事之后查询是否有异步
- ◆ 执行微任务 比如promise
- ◆ 执行宏任务，setTimeout,SetImmediate,等



Node.js实战

★★★★★ 8.2 (132人评价)

[美] Mike Cantelon / [美] TJ Holowaychuk / [美] Nathan



Nodejs + React 实战开发区块链
慕课Dapp 专门为前端工程师设计

中级 ▲ 132 评价: 10.00分

熟练掌握区块链技术原理+以太网开发技能
开发一个成熟的区块链Dapp项目



Node.js开发实战

★★★★★ (评价人数不足)

[美] Jim R. Wilson / 梅晴光 / 杜万智 / 陈琳 / 纪清华 /



深入浅出Node.js

★★★★★ 8.5 (844人评价)

朴灵 / 人民邮电出版社 / 2013-12-1 / CNY 69.00



Node.js 设计模式 (第2版)

★★★★★ 9.2 (17人评价)

【爱尔兰】Mario Casciaro / 【美国】Mattias Krikhaar / 【意大利】Lu-

小程序

腾讯爸爸给了新的工作机会

- ◆ 小程序开发流程
- ◆ 微信特有能力 支付、登录、消息
- ◆ 支付宝、百度、头条小程序



全网首发mpvue课程小程序全栈开发

中级

1313人评价

8.58分

慕课网独家首发 学习mpvue+Koa+mpvue 全栈开发小程序



纯正商业级应用-微信小程序开发 实战

中级

1191人评价

8.88分

真实数据的高质量小程序项目，学会真能工作

¥369.00

★ 收藏 | 1000+学员

498课时

云开发

继续弱化后端和运维，前端可以做的更多

- ◆ 云数据库
- ◆ 云函数
- ◆ 云存储

客户端

安卓 VS IOS

- ◆ React-native
- ◆ Weex(不是推荐)
- ◆ Flutter

多端融合

一个顶七个(安卓+ios+H5+小程序+百度+头条+支付宝+)

- ◆ Taro (腾讯imi)
- ◆ Uni-app(基于mpvue)
- ◆ 多端差异

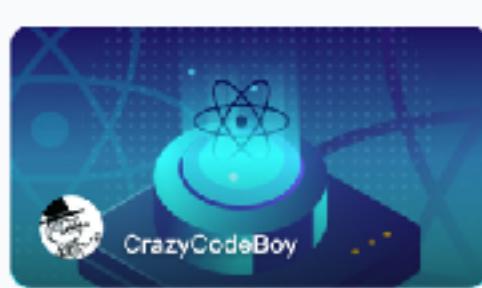


掌握Taro多端框架 快速上手小程序/H5开发

中级 ■ 180

评价: 10.00分

学会Taro框架，搞定多端应用开发（H5,小程序, ReactNative,快应用...）



新版React Native+Redux打造高质量上线App

中级 ■ 473

评价: 8.94分

解锁React Native开发应用新姿势，一网打尽React Native新版本热门技术

自动化测试

代码的健壮性，改代码不再胆战心惊

- ◆ 单元测试，mocha，jest，jasmine
- ◆ E2E测试Puppeteer，代码覆盖率istanbul
- ◆ 测试驱动开发TDD，先写测试，再写代码

```
// 打开截图
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://imooc.com');
  // 点击一个按钮
  await page.tap('.js_sale_buyalbum');
  await page.screenshot({path: 'yqq.png'});
  browser.close();
})();
```

```
module.exports = () => 'Hello world'

// hello.test.js
let hello = require('hello.js')

test('should get "Hello world"', () => {
  expect(hello()).toBe('Hello world') // 测试成功
  // expect(hello()).toBe('Hello') // 测试失败
})
```

PASS test/hello.test.js

✓ should get "Hello world" (4ms)

Test Suites: 1 **passed**, 1 total

Tests: 1 **passed**, 1 total

Snapshots: 0 total

Time: 0.927s, estimated 1s

Ran all test suites.

<http://www.ruanyifeng.com/blog/2015/12/a-mocha-tutorial-of-examples.html>



张轩_Viking

React16 组件化+测试+全流程 实战在线账本项目

中级 · 177 节 · 评估: 10.00 分

轻松上手，从设计到上线，精通组件化思想和组件测试，掌握大厂的开发模式和流程

前端监控

对运行状况了然于胸

- ◆ 前端性能监控 性能参数 berserkJS 屏幕补货，网络监控
- ◆ 前端错误监控 onerror Sentry
- ◆ 上报 img 的src

浏览器

最重要的一端

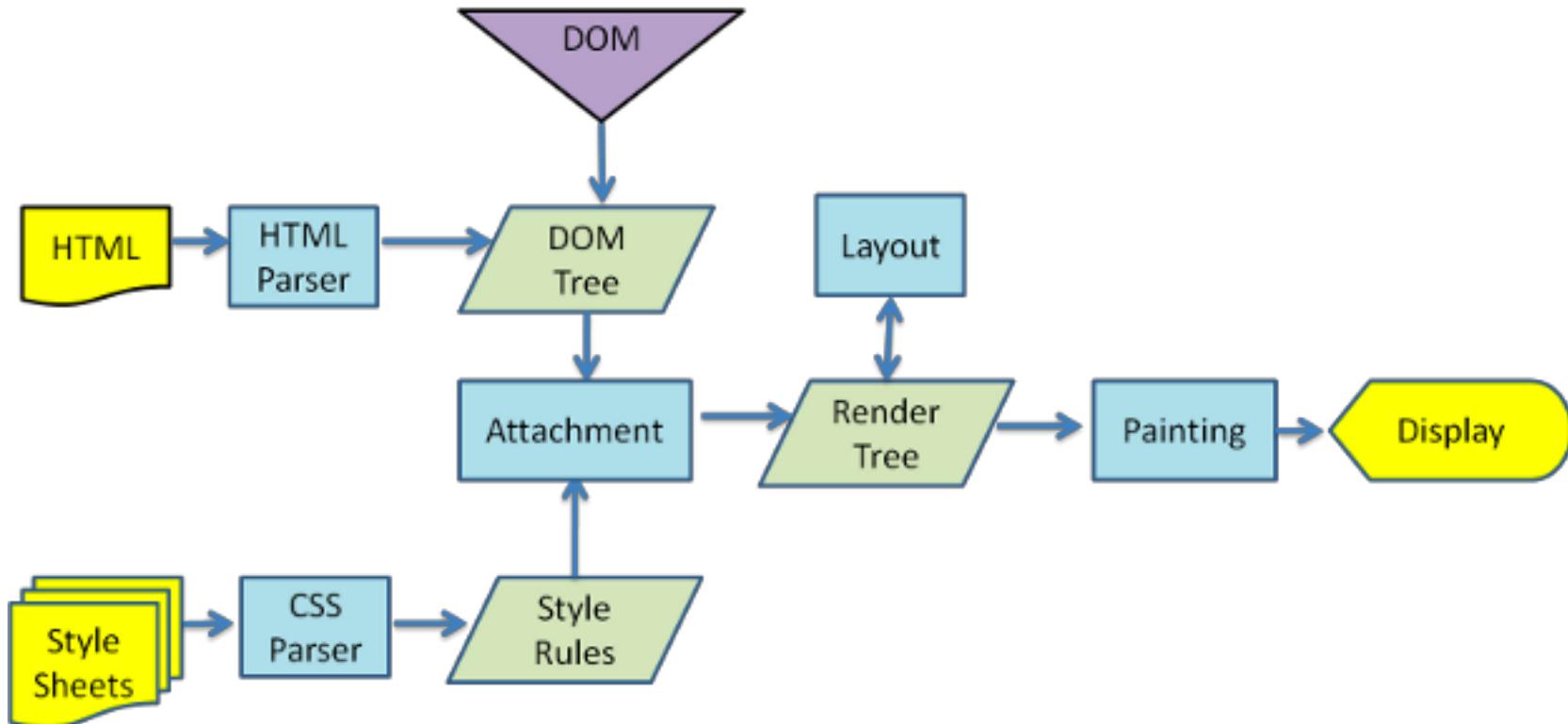
- ◆ 如何渲染的
- ◆ 缓存机制
- ◆ 输入url发生了啥

浏览器渲染

怎么画出这个网页的

- ◆ 收到html=》解析dom树
- ◆ css => css 树 和dom结合 形成render tree 开始渲染
- ◆ 少操作dom，重绘回流

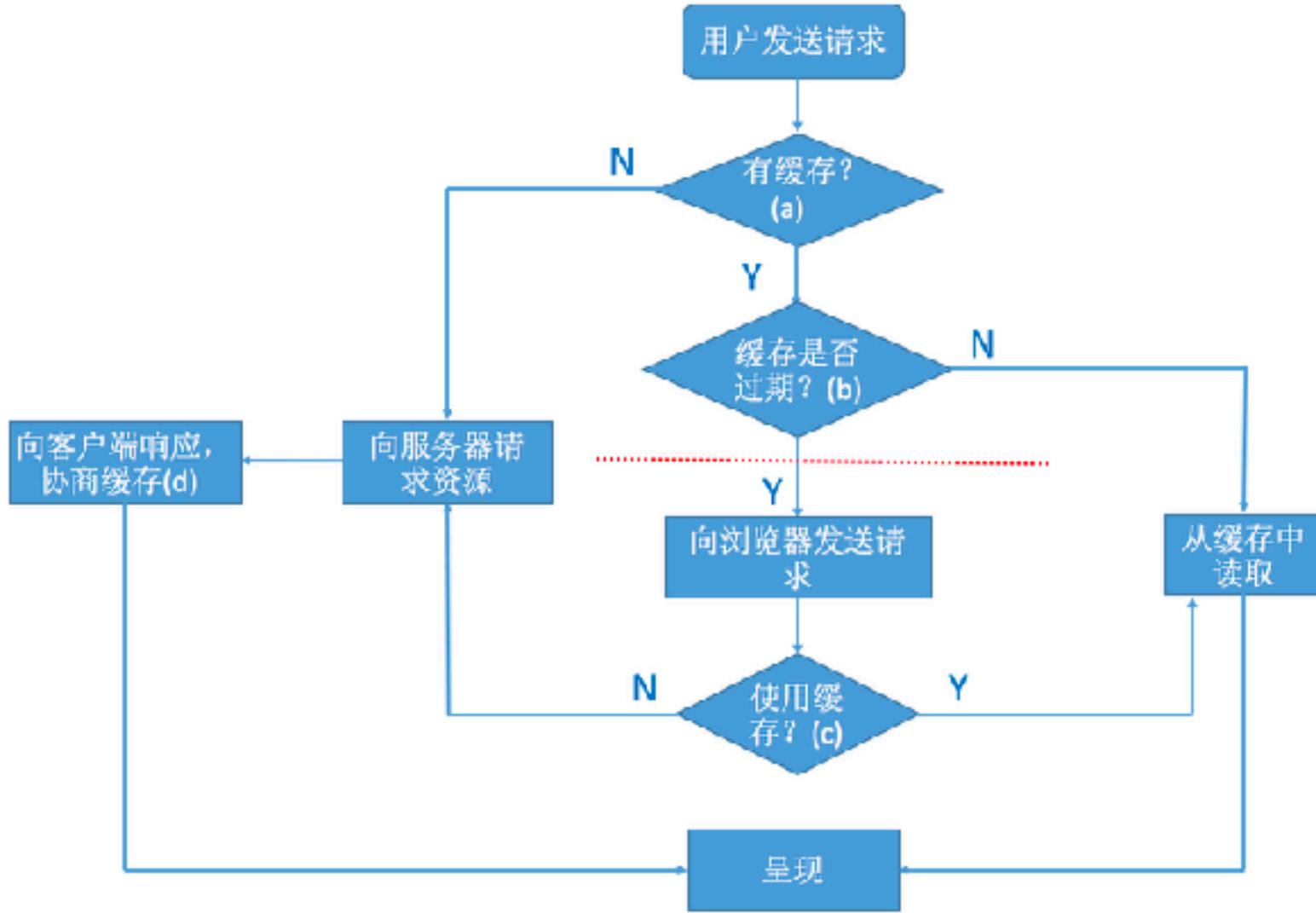
<https://www.html5rocks.com/zh/tutorials/internals/howbrowserswork/>



浏览器缓存

性能优化重要策略

- ◆ memory cache , disk cache
- ◆ 网络请求，强缓存弱缓存



部署

项目总要上线

- ◆ Pm2
- ◆ Nginx
- ◆ Docker + 自动化

特殊场景

不算通用能力，但是特殊业务需求

- ◆ 小游戏
- ◆ 可视化 echarts (canvas) , d3(svg) , three.js(webgl)
- ◆ PS切图

03 软件工程师

- ◆ 算法
- ◆ 数据结构
- ◆ 网络协议
- ◆ 设计模式
- ◆ 编译原理
- ◆ 数据库
- ◆ 数学基础
- ◆ 软件工程
- ◆ 编码

算法

面试必杀技

- ◆ 排序 搜索 遍历
- ◆ 贪婪
- ◆ 动态规划

数据结构

计算机组织数据的方式

- ◆ 数组、字符串、队列、堆、链表
- ◆ 二叉树
- ◆ 图



啊哈! 算法

★★★★★ 7.7 (395人评价)

啊哈磊 / 人民邮电出版社 / 2014-6-1 / 45.00元



算法 (第4版)

★★★★★ 9.4 (924人评价)

塞奇威克 (Robert Sedgewick) / 韦恩 (Kevin Wayne) / 译



JavaScript版 数据结构与算法

中级



858

评价: 0.80分

填补前端同学的算法短板，掌握面试中最常见的算法与数据结构

千万别看算法导论

网络协议

计算机如何唠嗑的

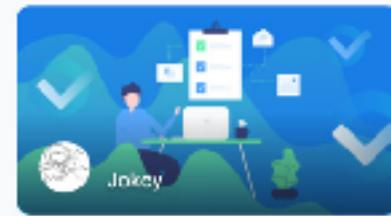
- ◆ IP
- ◆ TCP/UDP
- ◆ HTTP / HTTPS / SSH / FTP



图解HTTP

★★★★★ 8.1 (1803人评价)

【日】上舒宣 / 于均良 / 人民邮电出版社 / 2014-4-15 / 4



HTTP协议原理+实践 Web开发工程师必学

中級 | 21章 | 评价: 9.05分

解析HTTP协议原理,夯实HTTP技术基础,打造
前后端协作快速成长

设计模式

最佳实践，上分套路

- ◆ 常见设计模式，单例，装饰器，代理，观察者，发布订阅
- ◆ 前端常用的设计模式
- ◆ 如何使用、不要滥用



Javascript 设计模式系统讲解与应用

中级 ■ 703

评价: 0.85分

系统学习JS设计模式，增强设计思想，提高代码质量，赢得面试，赢得职场

数据库

数据存储的地方

- ◆ Mysql 关系型数据库，多表join
- ◆ Mongodb json数据库
- ◆ Redis 内存数据库 速度快

编译原理

前端近些年的发展，遍布了编译原理的实现

- ◆ babel , angular, webpack , vue都有编译原理的影子
- ◆ 词法分析，语法分析AST，代码生成，实现代码转换
- ◆ <https://github.com/livoras/blog/issues/14>

软件工程

软件是怎么被创造出来的

- ◆ 开发时的理论指导
- ◆ 需求分析，项目分解，代码测试，代码管理，bug追踪，敏捷开发
- ◆ 理论指导，实践抽象出理论，理论指导实战



构建之法 : 现代软件工程

★★★★★ 8.7 (458人评价)

邹欣 / 人民邮电出版社 / 2014-9 / 49.00元



人月神话

★★★★★ 8.4 (2968人评价)

[美] 弗雷德里克·布鲁克斯 / 汪颖 / 清华大学出版社 / 2002-

04 软技能

- ◆ 产品意识
- ◆ 业务
- ◆ 项目管理
- ◆ 如何学习
- ◆ 沟通表达
- ◆ 谈判
- ◆ 个人影响力
- ◆ 简历
- ◆ 公司
- ◆ 个人成长

产品

人人都是产品经理

- ◆ 和产品经理好好沟通，商业意识，用户意识，创新意识，团队意识
- ◆ 闷头写代码 活该被欺负
- ◆ 面试市场更吃香

个人影响力

睡后收入

- ◆ 博客 出课
- ◆ 开源 大会
- ◆ 出书

简历

程序员的简历，不需要特别花哨

- ◆ 要有技术亮点，2张A4以内，最好一张
- ◆ 一定要突出亮点，不写无关的技能，会html，css千万别写，会扣分
- ◆ 日常维护，每半年反思，简历驱动学习

谈判

谈判的来的，是纯利润

- ◆ 如何谈薪酬 你被压下的工资，都是HR小姐姐的KPI
- ◆ 敢于谈判，千万别说我对薪资没要求
- ◆ 先出价者输， 通过面试找准自己的实际定位，下次好开价

面试驱动学习

正确看待面试，是一个投入产出比贼高的事

- ◆ 面试官都是很合格的程序员，花钱咨询，一个小时好几百(在行)
- ◆ 好的学习方式，需要正确的反馈，面试就是免费的反馈机制
- ◆ 一定要问面试官问题，指点自己下一步学习方向

个人成长和规划

掌握正确的学习姿势

- ◆ 成长大于一切
- ◆ 外包和996尽量不去
- ◆ 刻意练习，学习英语，复盘和反思

刻意练习

正确的学习姿势

- ◆ 任务拆解 天天打荣耀的一定是个菜比，要分开研究，补刀，英雄，装备
- ◆ 练习 脱离舒适区，定期需要反馈
- ◆ 核心知识，完整的时间学习，比如Vue源码，React源码

回归面试

扯了半天，其实日常积累好了，面试是水到渠成的

- ◆ 一面，刷刷题
- ◆ 二面，准备一下项目描述，源码原理
- ◆ 三面，玄学，不好把握，人生观世界观价值观

分析公司和JD

第一步

- ◆ 天眼查，知乎
- ◆ JD描述看技术要求和特点

面试题

- ◆ 跨域方案
- ◆ 强缓存弱缓存
- ◆ 业余干啥
- ◆ 输入url发生啥
- ◆ 渲染原理
- ◆ 前后端分离JWT
- ◆ 箭头函数优点
- ◆ 可访问性
- ◆ 垃圾回收

<https://juejin.im/post/5c64d15d6fb9a049d37f9c20>

<https://juejin.im/post/5b44a485e51d4519945fb6b7>

<https://juejin.im/post/5c7646f26fb9a049fd108380>

输入URL

啥技术职位都用的面试题

- ◆ DNS解析，三次握手，建立链接
- ◆ 接受相应，查库查文件，等待数据返回，拼接响应报文
- ◆ 浏览器接受报文，解析html 渲染页面

996

抛开法律这一条，为什么反对996

- ◆ 时薪减半 $12*6/(8*5)=1.8$
- ◆ 工作是为了生活，不要搞反了
- ◆ 有时间，才能进步，才能进阶，低水平的勤奋，没鸟用

感想

编程就像修炼武功或者玩游戏

- ◆ 为啥我对如何上星耀的套路头头是道，还天天看教程喝视频，但自己还是个黄金，也是大家学编程的困境
- ◆ 光听不实战，等于0，光听我这个公开课，不能帮你通过面试
- ◆ 我推荐所有的书和课，都是看过的高质量的推荐，作者连一个肉夹馍都没给我

作为一个进大学才学编程的学生，如何能以后达到温赵轮三位大神的水平？



vczh

《C++Primer 5th》强势审校

徐飞 等 4484 人赞同了该回答

这十几年我一共做了三件事：

- 1、不以赚钱为目的选择学习的内容；
- 2、以自己是否能造出轮子来衡量学习的效果；
- 3、坚持每天写自己的代码，前10年每天至少6个小时，不包含学习和工作的时间。

就做了一点微小的工作，很惭愧，谢谢大家。

发布于 2015-09-23

▲ 4.5K

▼

● 248 条评论

↗ 分享

★ 收藏

♥ 感谢

...

收起 ^

小老弟 有啥问题吗？

金三银四 大家加油