# ­­­­ES6

interview：开发环境的使用 + 重点语法的掌握

ES6在浏览器中支持不好，所以需要开发环境编译。

问题：

1. ES6 模块化如何使用，开发环境如何打包？
2. Class和普通构造函数有何区别？
3. Promise 的基本使用和原理？
4. 总结一下ES6常用的功能？
5. ES6 模块化如何使用，开发环境如何打包？
6. 模块化的基本语法

答：略……（export、import）

1. 开发环境配置

答：略……（babel）

1. 关于JS众多模块化标准

答：略……AMD、CMD、（webpack、babel-loader、webpack.config、package.js、npm start）

1. Class 构造函数
2. 基本语法
3. 继承
4. 原生JS构造函数
5. Promise
6. promise语法
7. 手写promise
8. let/const
9. 多行字符串/模板变量
10. 解构赋值
11. 块级作用域
12. 箭头函数
13. 函数默认参数

# 原型

1. 说一个原型的实际应用。
2. 分析jQuery.js、zepto.js源码
   1. 描述jQuery和zepto 这两个库对原型的使用。（70——80 分）
   2. 结合自己的项目经验去回答，说自己开发的例子。
3. 原型如何体现它的扩展性。
4. 说下jQuery和zepto的插件机制。
5. 结合自己的开发经验，做过的基于原型的插件。

# 异步

1. 什么是单线程，单线程和异步的关系？

1、单线程，同一时间只能做一件事，两段JS不能同时执行。（单线程：只有一个线程，只能做一件事情。）

2、原因就是为了避免DOM渲染冲突（浏览器渲染DOM，JS修改DOM）

3、异步是一种“无奈”的解决单线程的方案，虽然有很多问题（具体解决方案：event-loop，回调先后顺序不确定）。

* 问题1：没按照书写方式执行，可读性差。
* 问题2：callback中不容易模块化。

1. event-loop事件轮询是什么？（执行JS的引擎监控轮询）
2. 事件轮询，JS实现异步的具体解决方案
3. 同步代码，直接执行
4. 异步函数先放在异步队列中
5. 待同步函数执行完毕，轮询执行异步队列的函数。
6. 是否用过jQuery的Deferred？（jQuery1.5.0以后ajax返回的是deferred，可以链式操作，Promise的前生。）
7. **deferred对象就是jQuery的回调函数解决方案**。
8. **无法改变JS异步和单线程的本质**。
9. **只能从写法上杜绝callback这种形式**
10. **他是一种语法糖形式，但是解耦了代码。**
11. **很好的体现了：开放封闭原则。（只扩展不修改）**
12. **说明promise和deferred的区别**
13. Promise的基本使用和原理？
    1. 基本语法
    2. 异常捕获（Error或reject都有考虑）
    3. 多个串联（链式执行的好处）
    4. Promise.all 和 Promise.race（race是赛跑的意思，谁先成功就先返回）
    5. Promise的标准（状态变化、then函数）
       1. 三种状态：pending、fulfilled、rejected
       2. 初始状态：pending
       3. pending变为fulfilled、或者pending变为rejected
       4. 状态不可逆
       5. Promise实例必须实现then这个方法
       6. then()必须接收两个函数作为参数
       7. then()返回的必须是一个Promise实例
14. 介绍一下async/await（和Promise的区别、联系）？
15. then只是将callback拆分了
16. async/await是最直接的同步写法
    1. 使用await，函数必须用async标识
    2. await后面跟的是一个Promise实例
    3. 需要babel-polyfill（polyfill是兼容的意思）
17. 使用了Promise，并没有和Promise冲突
18. 完全是同步的写法，再也没有回调函数
19. 但是：改变不了JS单线程，异步的本质
20. 总结下当前JS解决异步的方案？
21. jQuery Defferred
22. Promise
23. Async/await
24. Generator
    1. 原理比较复杂，学习成本高
    2. 不是异步的直接替代方式
    3. 有更好更简洁的解决方案async/await
    4. koa也早已“弃暗投明”

# 虚拟DOM

1. 问题
   1. vdom是什么？为何会存在vdom？
      1. Virtual dom，虚拟DOM。
      2. 用JS模拟DOM结构。
      3. DOM操作非常“昂贵”。
      4. 将DOM对比操作JavaScript层，提高效率。
   2. vdom如何应用，核心API是什么？
      1. 可以使用snabbdom的用法来举例。
      2. 核心API：h函数、patch函数
   3. 介绍下diff算法？
      1. 是Linux的基本命令
      2. vdom中应用diff算法是为了找出需要更新的节点。
      3. diff算法的实现，pach(container, vnode) patch(vnode,newVnode)。
      4. 核心逻辑，creatElement 和 updateChildren。

# MVVM和vue

1. 问题
   1. 如何理解MVVM
   2. 如何实现MVVM
   3. 是否解读过vue源码：（学习能力、理解能力、自学能力、是否有技术弱性【自己愿意读取一些生层次的东西】、解决问题的能力）。
2. 问题
   1. 说一下使用jquery和使用框架的区别。
      1. 数据和视图的分离，解藕（开放封闭原则）
      2. 以数据驱动视图，只关心数据变化，DOM操作被封装。
   2. 说一下对MVVM的理解。
      1. MVC：用户----view（界面）----controller（控制view、model的变化）----model（数据源）----view（界面）
      2. MVVM：View（视图、模板 【视图和模型是分离的】）、Model（模型、数据）、ViewModel（连接Model和View 【事件绑定和数据绑定】）
      3. 三者之间的联系，以及如何对应到各段代码
      4. ViewModel的理解，联系View和Model

MVVM框架的三要素：分析vue的demo；

三要素总结：

1. 响应式：vue如何监听到data的每个属性的变化？
2. 模板引擎：vue的模板如何被解析，指令如何处理？
3. 渲染：vue的模板如何被渲染成html？以及data变化后html重新渲染的过程？
   1. vue中如何实现响应式。
      1. 什么是响应式？
         1. 修改data属性之后，vue立即监听到。
         2. data属性被代理到vm上。
      2. Object.defineProperty
      3. 模拟

d

关键是理解Object.defineProperty。

将data的属性代理到vm上。

* 1. vue如何解析模板。
     1. 模板是什么
        1. 本质是字符串
        2. 有逻辑，如v-if v-for 等
        3. 可以嵌入JS变量
        4. 与html格式很像，但有很大区别
        5. 最终还要转换为html来显示
        6. 模板必须转换成JS代码，因为：
           1. 有逻辑（v-if v-for），必须用JS才能实现（图灵完备的语言）
           2. 转换为html渲染页面，必须用JS才能实现
           3. 因此，模板最终要转换成JS函数（render函数）。
     2. 模板转化成render函数
        1. with的用法
        2. render函数
           1. 模板中所有信息都包含在render函数
        3. 根据todo-list demo 的render函数
           1. 已经解决了模板中“逻辑”（v-for v-if）的问题

v-model是怎么实现的？

v-on:click 是怎么实现的？

v-for是怎么实现的？

* + - * 1. 还剩下生成html的问题

Snabbdom： pathch函数、h函数、

* + - * 1. 另外，vm.\_c是什么？render函数返回了什么？

Vm.\_c其实就相当于snabbdom中的h函数

render函数执行之后，返回的是vnode

* 1. 渲染
     1. render函数与vdom
        1. updateComponent中实现了vdom的patch
        2. 页面首次渲染执行updateComponent
        3. data中每次修改属性，执行updateComponent
  2. vue的整个实现流程
     1. 解析模板成render函数
        1. width的应用
        2. 模板中的所有信息都被render函数包含
        3. 模板中用到的data属性，都变成成JS变量
        4. 模板中的v-model v-for v-on 都变成了JS逻辑
        5. render函数返回vnode
     2. 响应式开始监听
        1. Obejcet.defineProperty
        2. 将data的属性代理到vm上
     3. 首次渲染，显示页面，且绑定依赖。
        1. 初次渲染执行updateComponent，执行vm.\_render()
        2. 执行render函数，会访问到vm.list和vm.title
        3. 会被响应式的get方法监听到（Object.defineProperty()）
        4. 执行updateComponent，会走到vdom的patch方法
        5. patch将vnode渲染成DOM，初次渲染完成。
     4. data属性变化触发render函数
        1. 修改属性，被响应式的set监听到
        2. set中执行updateComponent
        3. updateComponent重新执行vm.\_render()
        4. 生成的vnode和preVnode, 通过patch进行对比。

# 组件化和React

1. 问题
   1. 说一下对组件化的理解？
      1. 组件的封装（为了给外边使用，外边不需要关心里边的内在逻辑）
         1. 视图
         2. 数据
         3. 变化逻辑（数据驱动视图变化）
      2. 组件的复用
         1. props传递
         2. 传递不同的props实现组件复用
   2. JSX的本质是什么？
      1. JSX语法
         1. html形成
            1. 所有html支持的标签它都支持
         2. 引入JS变量和表达式
            1. 可以使用大括号引入JS变量或者JS表达式
         3. If …… else……
            1. 在html标签中使用打括号进行条件判断
         4. 循环
            1. 在html标签中使用打括号，用map循环数组，return返回html标签或者是变量
         5. style和className
            1. Style={{fontSize: ‘40px’, color: ‘blue’}}
            2. className 定义html标签的类名
         6. 事件绑定
      2. JSX是语法糖，需要被解析成JS才能运行。
         1. JSX其实是语法糖
         2. 开发环境会将JSX编译成JS代码
         3. JSX的写法大大降低了学习成本和编码工作量
         4. 同时，JSX也会增加debug成本
      3. JSX是独立的标准，可被其他项目使用
         1. JSX是React引入的，但不是React独有的
         2. React已经将它作为一个独立标准开发，其他项目也可以。
         3. React.createElement是可以自定义修改的
         4. 说明：本身功能已经完备；和其他标准兼容和扩展性没问题。
   3. JSX和vdom的关系？
      1. 分析：为何需要vdom
         1. vdom是React初次推广开来的，结合JSX
         2. JSX就是模板，最终要渲染成html
         3. 初次渲染+修改state后的re-render
         4. 正好符合vdom的应用场景
      2. React.createlement 和 h （React.createlement 使用了vdom的h函数返回vnode）
         1. 所有的JSX代码都要解析(React.createlement)成JS，执行JS之后返回vnode（h函数）
      3. 何时patch？(ReactDom.render(....) 和setState使用的是vdom的patch函数)
         1. 初次渲染-ReactDOM.Render(<APP/>, container)
         2. 会触发patch(container, vnode)
         3. re-render - setState
         4. 会触发patch(vnode, newVnode)
      4. 自定义组件的解析
         1. ‘div’ - 直接渲染成<div>即可，vdom可以做到
         2. Input和List，是自定义组件（class），vdom默认不认识
         3. 因此Input和List定义的时候必须声明render函数。
         4. 根据props初始化实例，然后执行实例的render函数。
         5. render函数返回的还是vnode对象。
2. 问题
   1. React VS vue
      1. 两者的本质区别
         1. Vue - 本质是MVVM框架，由MVC发展而来
         2. React - 本质是前端组件化框架，由后端组件化发展而来
      2. 看模板和组件化的区别
         1. Vue - 使用模板（最初由angular提出）
         2. React - 使用JSX
         3. 模板语法上，我更倾向于JSX
         4. 模板分离上，我更倾向于vue
            1. 模板应该和JS逻辑分离
            2. 回顾“开放封闭原则”
      3. 两者共同点
         1. 都支持组件化
         2. 都是数据驱动视图
      4. 总结问题的答案
         1. 国内使用，首推vue。文档更易读易懂，易学，社区够大
         2. 如果团队的水平高，推荐使用React，组件化和JSX
   2. 说下React setState的过程
      1. react的setState是异步
      2. vue修改属性也是异步
      3. setState的过程

# hybrid

1. 问题
   1. hybrid是什么？为何用hybrid？
      1. hybrid即“混合”，即前端和客户端的混合开发
      2. 某些环节也可能涉及到server端
      3. 可以快速迭代更新【关键】（无需app审核，思考为何？）
      4. 体验流畅（和NA的体验基本类似）
      5. 减少开发和沟通成本，双端公用一套代码。
   2. 介绍下hybrid更新和上线的流程？
   3. hybrid和h5的区别
      1. 优点
         1. 体验更好，更NA体验一致
         2. 可快速迭代，无需app审核
      2. 缺点
         1. 开发成本高，联调、测试、查bug都比较麻烦
         2. 运维成本高，参考更新上线流程
      3. 适用场景
         1. Hybrid：产品的稳定功能，体验要求高，迭代频繁
         2. H5：单次运营活动（如：XX红包）或不常用功能
   4. 前端JS和客户端如何通讯？
      1. 通讯的基本形式：调用能力、传递参数、监听回调
      2. 对schema协议的理解和使用
      3. 调用schema代码的封装
      4. 内置上线的好处：更快、更安全
   5. hybrid的具体实现？
      1. 具体实现流程：
         1. 前端做好静态页面）（html \ css \ js，将文件交给客户端。
         2. 客户端拿到前端静态页面，以文件的形式存储在app中。
         3. 客户端在一个webview中
         4. 使用file协议加载静态页面
      2. app发布之后，静态文件如何实时更新？
      3. 随时
      4. 静态页面如何动态获取内容？
      5. 使用场景
         1. 使用NA：体验要求极致，变化不频繁（如：头条首页）
         2. 使用hybrid：体验要求高，变化频繁（如：头条的新闻详情页）
         3. 使用h5：体验无要求，不常用（如：举报、反馈等页面）
   6. file协议和http协议的区别
      1. file协议：本地文件，快
      2. http协议：网络加载，慢
   7. Webview
      1. 是app中的一个组件（app可以有webview，也可以没有）
      2. 用于加载h5页面，即一个小型的浏览器内核。

# 发挥题

1. 做项目的过程中遇到过什么问题

1、图片压缩：（提示：canvas压缩的实现过程）

# HTTP协议

1. HTPP协议是什么？

