面试

职位诱惑：

上市公司,大牛团队,福利优厚,晋升空间大

职位描述：

岗位职责：

1、参与项目、产品需求分析与设计，负责前端架构设计及技术路线；

2、组织前端开发团队完成前端功能设计、开发、测试和交付。

3、负责前端核心功能的架构与代码模板编写，对系统核心模块进行开发和维护；

4、制定前端开发规范，参与制定技术标准，编写相应的技术文档，对通用技术进行整理；

5、负责前端开发人员的招聘和考核，定期开展前端开发工程师培训，促进团队成员的进步。

岗位要求：

1、5年以上Web前端研发经验；

2、能够熟练的设计和开发基于Javascript的复杂web应用并有实践经验；

3、必须精通VueJS或Angular2/3/4/5框架，熟悉ReactJs，NodeJs等框架，有产品设计经验优先；

4、熟练掌握gulp，webpack，browserify等工具，并且对实现细节有研究，有ionic，Electron等Hybird框架经验尤佳；

5、能凭借丰富的开发经验快速定位并解决各种前端问题；

6、具有微信公众号开发经验，了解小程序的开发；

7、具备良好的识别和设计通用框架及模块的能力；

8、逻辑思维强、注重团队协作；

工作地址

北京 - 海淀区 - 中关村　　查看地图

课堂目标

1. 掌握前端工程师面试的方方面面
2. 掌握常见面试题
3. 简历优化
4. 职业生涯软技能

知识要点

1. javascript
2. ES6
3. this
4. promise
5. 浏览器
6. 安全
7. vue
8. reac
9. 工程化
10. 计算机基础

起步

###

面试准备

不打无准备之仗，面试就是一个考试 所谓台上三分钟，台下十年功，必须要通过某个门槛，才能通过，拿到理想的offer，所以需要我们之前提前2~3个月来精心的去准备这场战斗

talk is cheap show me the ~~money~~ code

JD分析

去考试之前，要先审题，大家首先要分析jd ，有针对的去准备，不是每个岗位都适合自己

职位诱惑：

上市公司,大牛团队,福利优厚,晋升空间大

职位描述：

岗位职责：

1、参与项目、产品需求分析与设计，负责前端架构设计及技术路线；

2、组织前端开发团队完成前端功能设计、开发、测试和交付。

3、负责前端核心功能的架构与代码模板编写，对系统核心模块进行开发和维护；

4、制定前端开发规范，参与制定技术标准，编写相应的技术文档，对通用技术进行整理；

5、负责前端开发人员的招聘和考核，定期开展前端开发工程师培训，促进团队成员的进步。

岗位要求：

1、5年以上Web前端研发经验；

2、能够熟练的设计和开发基于Javascript的复杂web应用并有实践经验；

3、必须精通VueJS或Angular2/3/4/5框架，熟悉ReactJs，NodeJs等框架，有产品设计经验优先；

4、熟练掌握gulp，webpack，browserify等工具，并且对实现细节有研究，有ionic，Electron等Hybird框架经验尤佳；

5、能凭借丰富的开发经验快速定位并解决各种前端问题；

6、具有微信公众号开发经验，了解小程序的开发；

7、具备良好的识别和设计通用框架及模块的能力；

8、逻辑思维强、注重团队协作；

工作地址

北京 - 海淀区 - 中关村　　查看地图

关键字 架构设计、精通vue或者angular、公众号、小程序、沟通

职位诱惑：

晋升空间,技术氛围

职位描述：

工作职责：

1、负责餐饮生态前端技术的架构设计，并参与整体的技术中台规划建设。保证架构的可持续性发展，并具备一定程度的应变能力。

2、负责产品需求到实现过程中的设计与技术选型工作，并参与核心业务模块、通用业务系统的开发工作。

3、负责提升各技术项目中的代码质量、设计质量和工程质量。

4、保持积极、负责的工作态度，对团队充满正能量、对项目充满强自驱力。

职位要求：

1、3年以上的前端领域开发经验，有主导前端架构设计的经历

2、扎实的计算机基础，对数据结构和算法有一定的了解

3、Javascript/Hybrid/Node至少有一个方向有深入的了解

4、至少了解一门服务器端相关技术，熟练掌握Linux

加分项：

1、对性能优化、多端适配、开发者框架有实际产品经验者优先

2、有长期维护技术专栏，个人项目或站点者优先

3、有React/React Native 或 Express/Koa 开发经验者优先

工作地址

北京 - 朝阳区 - 望京 - 北京市朝阳区望京东路4号恒基伟业C座（美团点评北京总部） 查看地图

关键字：餐饮生态(pc 移动端) 前端架构设计 、计算机基础、react/native 、nodejs

简历

程序员的简历不需要特别花哨，但是很多人不会写简历

1. 基本信息 姓名 年龄 手机 邮箱
2. 学历
3. 工作经历
4. 开源项目
5. 技术点(最好是源码级)

附加信息

有前端团队的管理经验。

使用 Html、Css 编写 PC 端页面，能够快速定位并解决浏览器兼容性问题，完美还原 UI 设计。

使用 Html5、Css3 编写移动端 H5 页面， 适配 Android、Ios 系统不同分辨率机型，解决移动端遇到的问题，完美还原 UI 设计。

使用 Jquery、Javascript、Zepto.js，实现 PC 端和移动端交互效果。

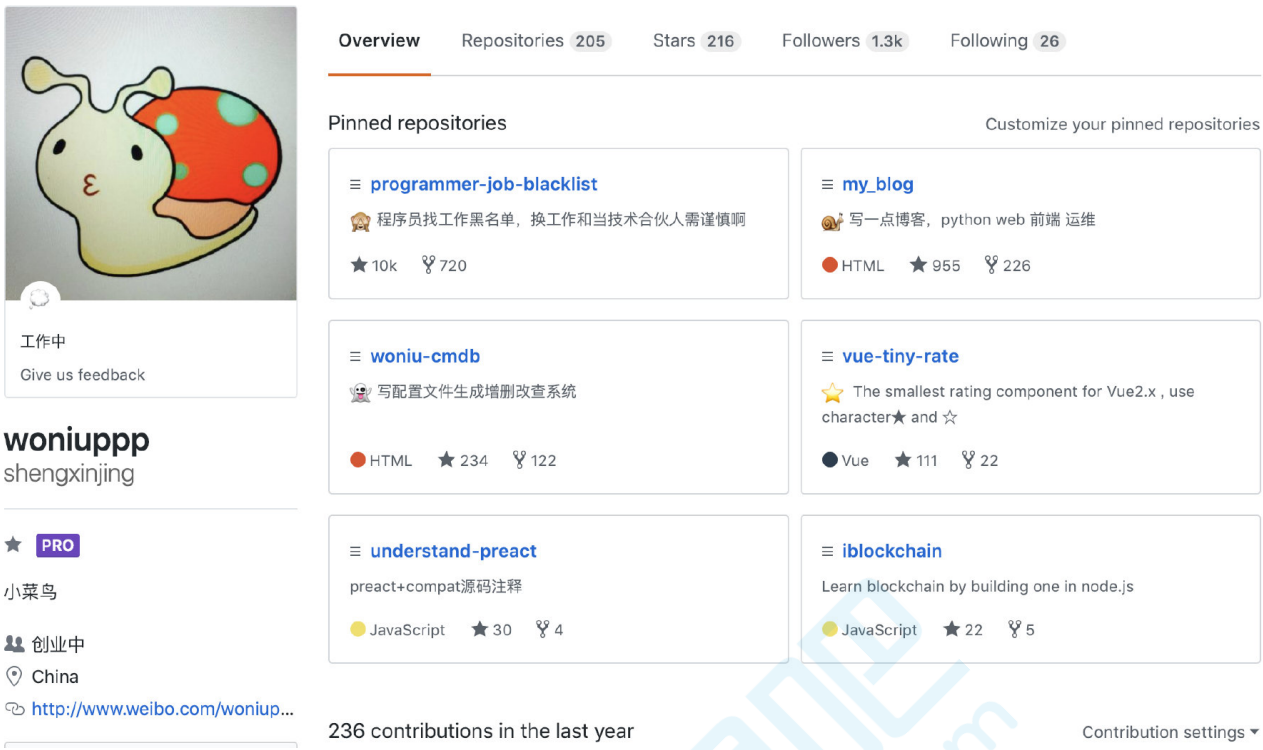
使用 Mvvm 框架 Vue.js 并应用到开发项目中。

个人简介

- 擅长前端和python开发
- 擅长react全家桶开发，react+redux+react-router，看过源码，了解原理
- 自己实现过迷你的reactjs
- 自己实现过迷你的vuejs
- 擅长前端工程化的架构设计
- 看过angular的源码并用es6+webpack写了一个简版angular，擅长mvvm库做组件化开发
- 懂运维,python实现过一套简单的监控系统，后端代码仿照memcached架构，基于epoll和自己定义的状态机，网络用优化后的netstring协议
- 熟悉区块链开发
- 熟练构建完整的交易所前端

能击中面试官内心的，就是要展示你是一个专业的程序员，最直观的 就是 把控项目的能力，比如开源项目和源码！ 需要日常做准备

https://github.com/shengxinjing 欢迎follow



技术栈

ES6

1. let const
2. 箭头函数
3. class
4. promise
5. 解构
6. import

http协议

浏览器里大部分都是http协议

HTTP 协议是个无状态协议，不会保存状态。 状态码

2XX 成功

- 200 OK，表示从客户端发来的请求在服务器端被正确处理
- 204 No content，表示请求成功，但响应报文不含实体的主体部分
- 205 Reset Content，表示请求成功，但响应报文不含实体的主体部分，但是与 204 响应不同在于要求请求方重置内容

- 206 Partial Content，进行范围请求

3XX 重定向

- 301 moved permanently，永久性重定向，表示资源已被分配了新的 URL
- 302 found，临时性重定向，表示资源临时被分配了新的 URL
- 303 see other，表示资源存在着另一个 URL，应使用 GET 方法获取资源
- 304 not modified，表示服务器允许访问资源，但因发生请求未满足条件的情况
- 307 temporary redirect，临时重定向，和302含义类似，但是期望客户端保持请求方法不变向新的地址发出请求

4XX 客户端错误

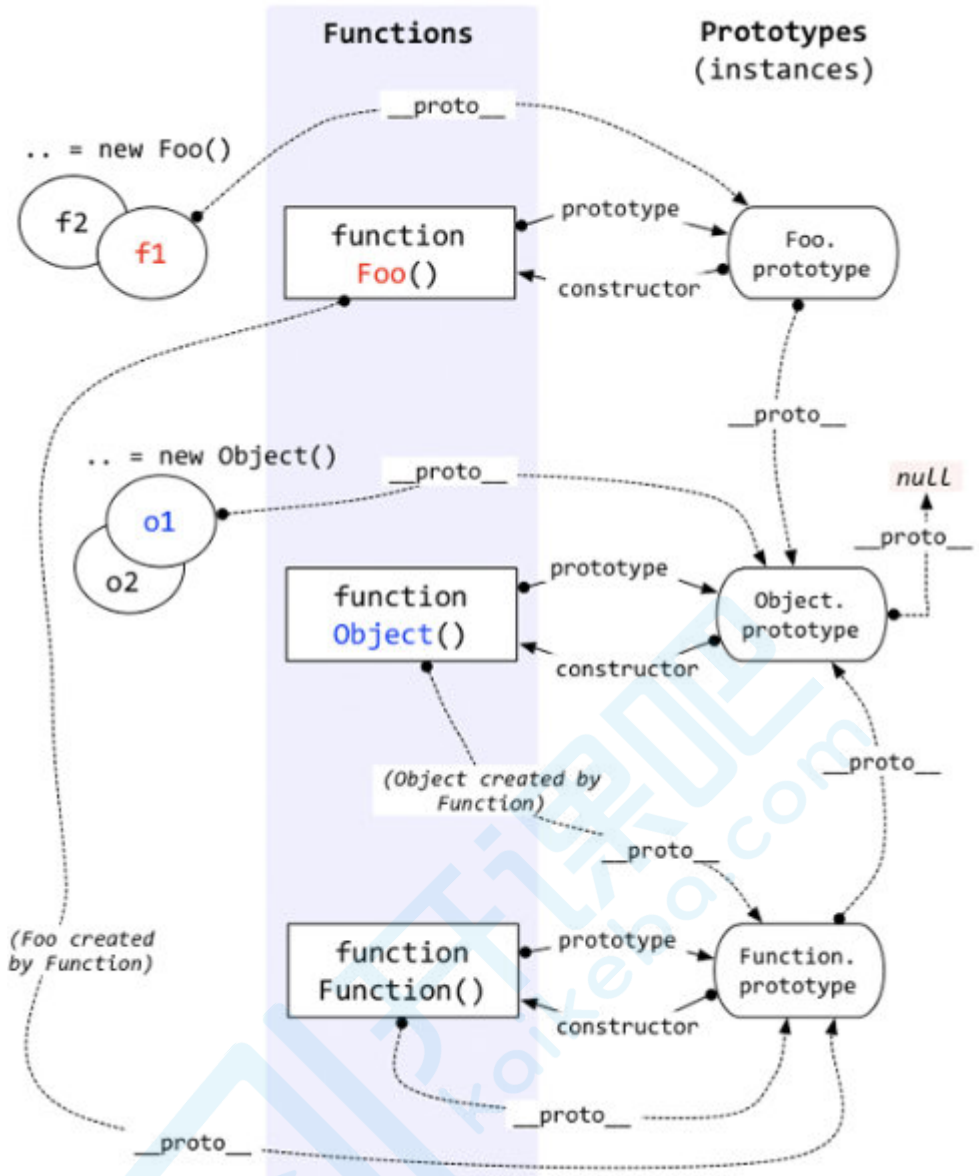
- 400 bad request，请求报文存在语法错误
- 401 unauthorized，表示发送的请求需要有通过 HTTP 认证的认证信息
- 403 forbidden，表示对请求资源的访问被服务器拒绝
- 404 not found，表示在服务器上没有找到请求的资源

5XX 服务器错误

- 500 internal sever error，表示服务器端在执行请求时发生了错误
- 501 Not Implemented，表示服务器不支持当前请求所需要的某个功能
- 503 service unavailable，表明服务器暂时处于超负载或正在停机维护，无法处理请求

#

原型链



this

this指向常见错误

1. Bind
2. apply
3. 箭头函数

```
function foo() {
    console.log(this.a)
}
var a = 1
foo()

var obj = {
    a: 2,
    foo: foo
}
obj.foo()
```

```
// 以上两者情况 `this` 只依赖于调用函数前的对象，优先级是第二个情况大于第一个情况

// 以下情况是优先级最高的，`this` 只会绑定在 `c` 上，不会被任何方式修改 `this` 指向
var c = new foo()
c.a = 3
console.log(c.a)

// 还有种就是利用 call，apply，bind 改变 this，这个优先级仅次于 new


let a = {
    value: 1
}
function getValue(name, age) {
    console.log(name)
    console.log(age)
    console.log(this.value)
}
getValue.call(a, 'yck', '24')
getValue.apply(a, ['yck', '24'])
```

promise

Promise 是 ES6 新增的语法，解决了回调地狱的问题

简单算法

复杂度

1. 冒泡
2. 快排

二面

更注意广度和深度

js运行机制

JS 在执行的过程中会产生执行环境，这些执行环境会被顺序的加入到执行栈中。如果遇到异步的代码，会被挂起并加入到 Task（有多种 task） 队列中。一旦执行栈为空，Event Loop 就会从 Task 队列中拿出需要执行的代码并放入执行栈中执行，所以本质上来说 JS 中的异步还是同步行为。

不同的任务源会被分配到不同的 Task 队列中，任务源可以分为 微任务（microtask） 和 宏任务（macrotask）。在 ES6 规范中，microtask 称为 `jobs`，macrotask 称为 `task`

```
console.log('script start')
```

```
setTimeout(function() {
  console.log('setTimeout')
}, 0)

new Promise(resolve => {
  console.log('Promise')
  resolve()
})
  .then(function() {
    console.log('promise1')
  })
  .then(function() {
    console.log('promise2')
  })

console.log('script end')
// script start => Promise => script end => promise1 => promise2 => setTimeout
```

以上代码虽然 `setTimeout` 写在 `Promise` 之前，但是因为 `Promise` 属于微任务而 `setTimeout` 属于宏任务，所以会有以上的打印。

微任务包括 `process.nextTick` ， `promise` ， `Object.observe` ， `MutationObserver`

宏任务包括 `script` ， `setTimeout` ， `setInterval` ， `setImmediate` ， `I/O` ， `UI rendering`

很多人有个误区，认为微任务快于宏任务，其实是错误的。因为宏任务中包括了 `script` ，浏览器会先执行一个宏任务，接下来有异步代码的话就先执行微任务。

所以正确的一次 Event loop 顺序是这样的

1. 执行同步代码，这属于宏任务
2. 执行栈为空，查询是否有微任务需要执行
3. 执行所有微任务
4. 必要的话渲染 UI
5. 然后开始下一轮 Event loop，执行宏任务中的异步代码

通过上述的 Event loop 顺序可知，如果宏任务中的异步代码有大量的计算并且需要操作 DOM 的话，为了更快的界面响应，我们可以把操作 DOM 放入微任务中。

页面性能

1. 网络
    1. dns解析
    2. 缓存
    3. 预加载
    4. 文件优化
    5. 代码执行

缓存

强缓存可以通过两种响应头实现：`Expires` 和 `Cache-Control` 。强缓存表示在缓存期间不需要请求，`state code` 为 200

如果缓存过期了，我们就可以使用协商缓存来解决问题。协商缓存需要请求，如果缓存有效会返回 304。

协商缓存需要客户端和服务端共同实现，和强缓存一样，也有两种实现方式。

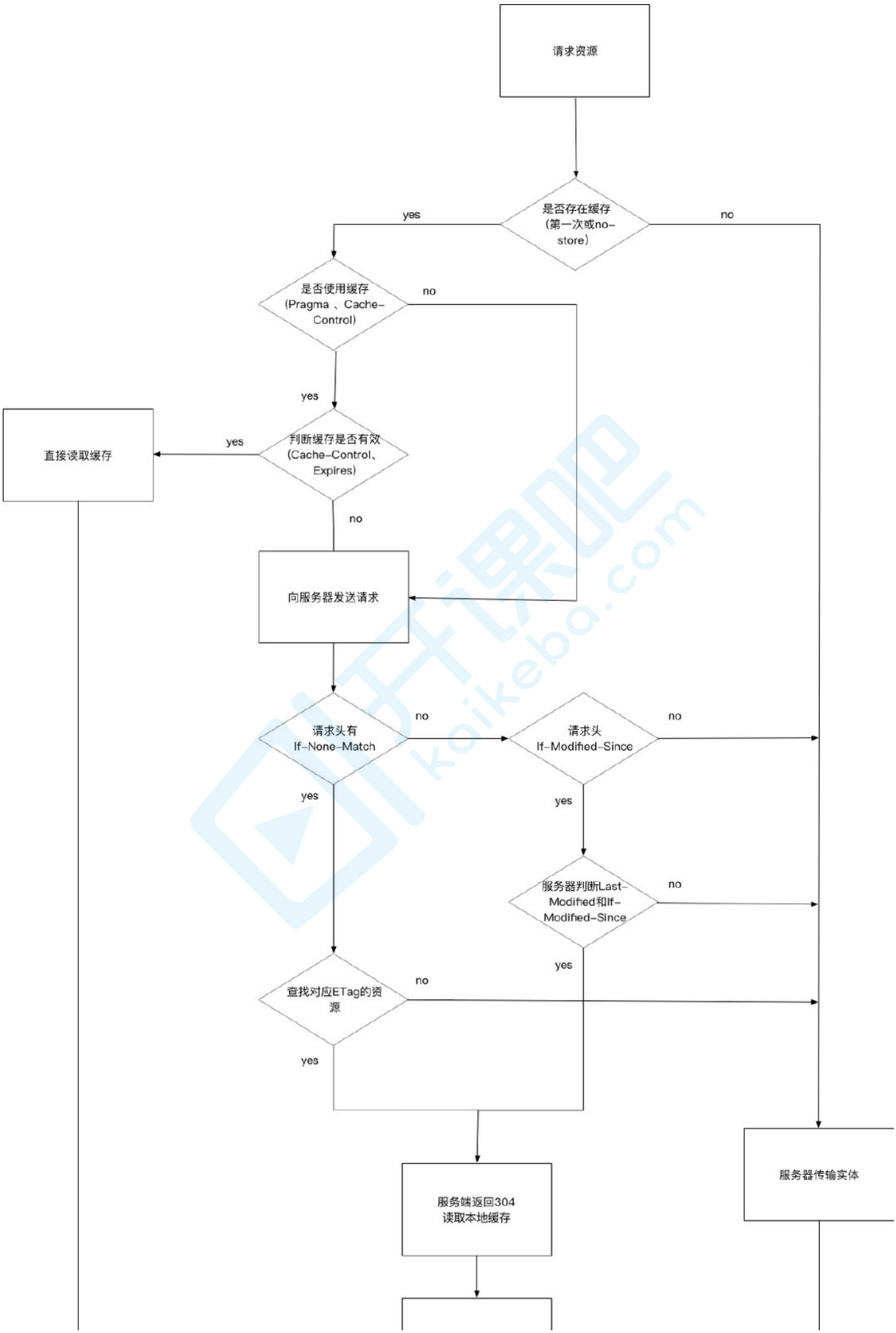
Last-Modified 和 If-Modified-Since

`Last-Modified` 表示本地文件最后修改日期，`If-Modified-Since` 会将 `Last-Modified` 的值发送给服务器，询问服务器在该日期后资源是否有更新，有更新的话就会将新的资源发送回来。

但是如果在本地打开缓存文件，就会造成 `Last-Modified` 被修改，所以在 HTTP / 1.1 出现了 `ETag` 。

ETag 和 If-None-Match

`ETag` 类似于文件指纹，`If-None-Match` 会将当前 `ETag` 发送给服务器，询问该资源 `ETag` 是否变动，有变动的话就将新的资源发送回来。并且 `ETag` 优先级比 `Last-Modified` 高。

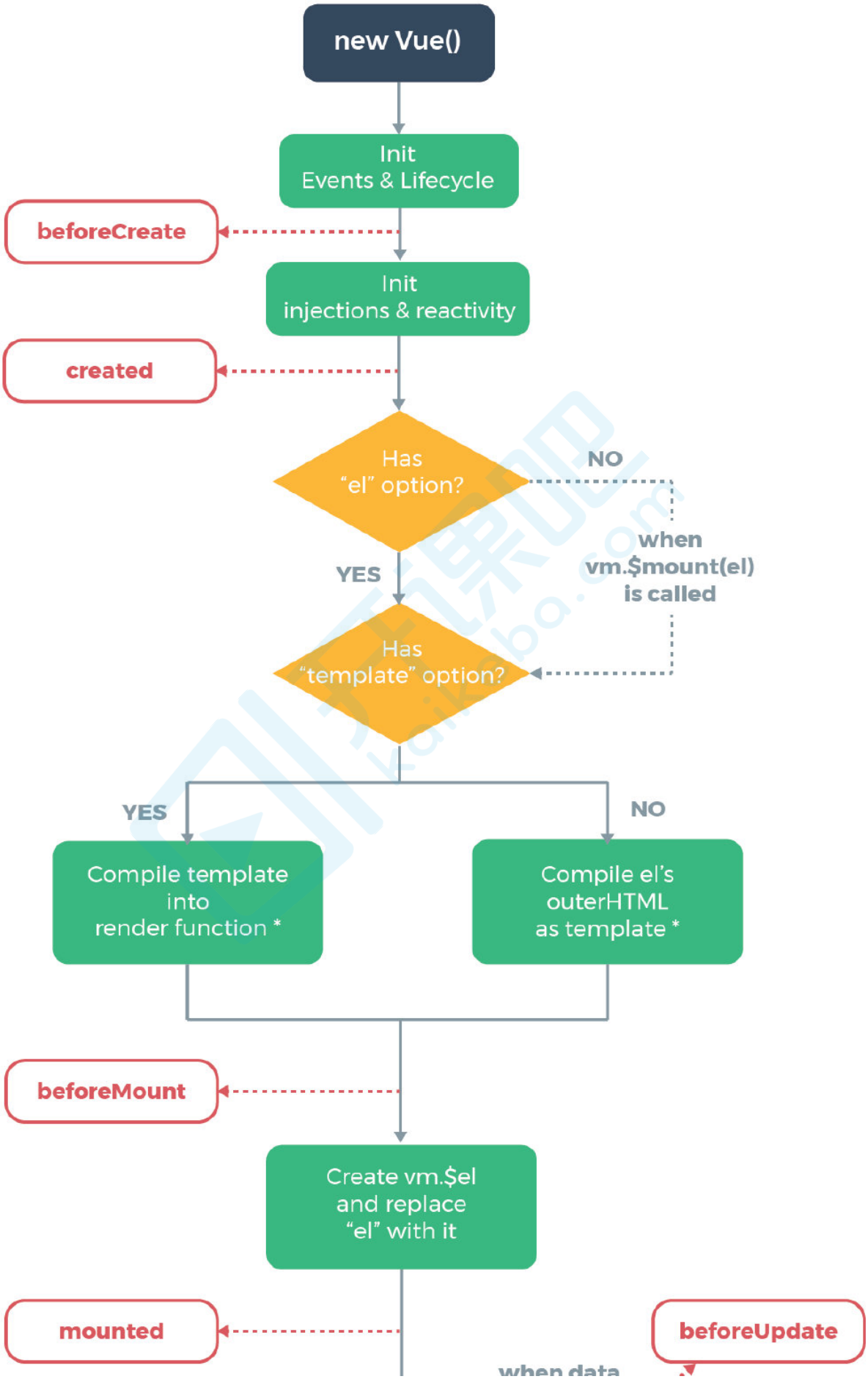

请求资源
是否存在缓存
(第一次或no-
store)
yes
no
是否使用缓存
(Pragma 、Cache-
Control)
no
yes
直接读取缓存
yes
判断缓存是否有效
(Cache-Control、
Expires)
no
向服务器发送请求
请求头有
If-None-Match
no
请求头
If-Modified-Since
no
yes
yes
服务器判断Last-
Modified和If-
Modified-Since
no
yes
查找对应ETag的资
源
no
yes
服务端返回304
读取本地缓存
服务器传输实体

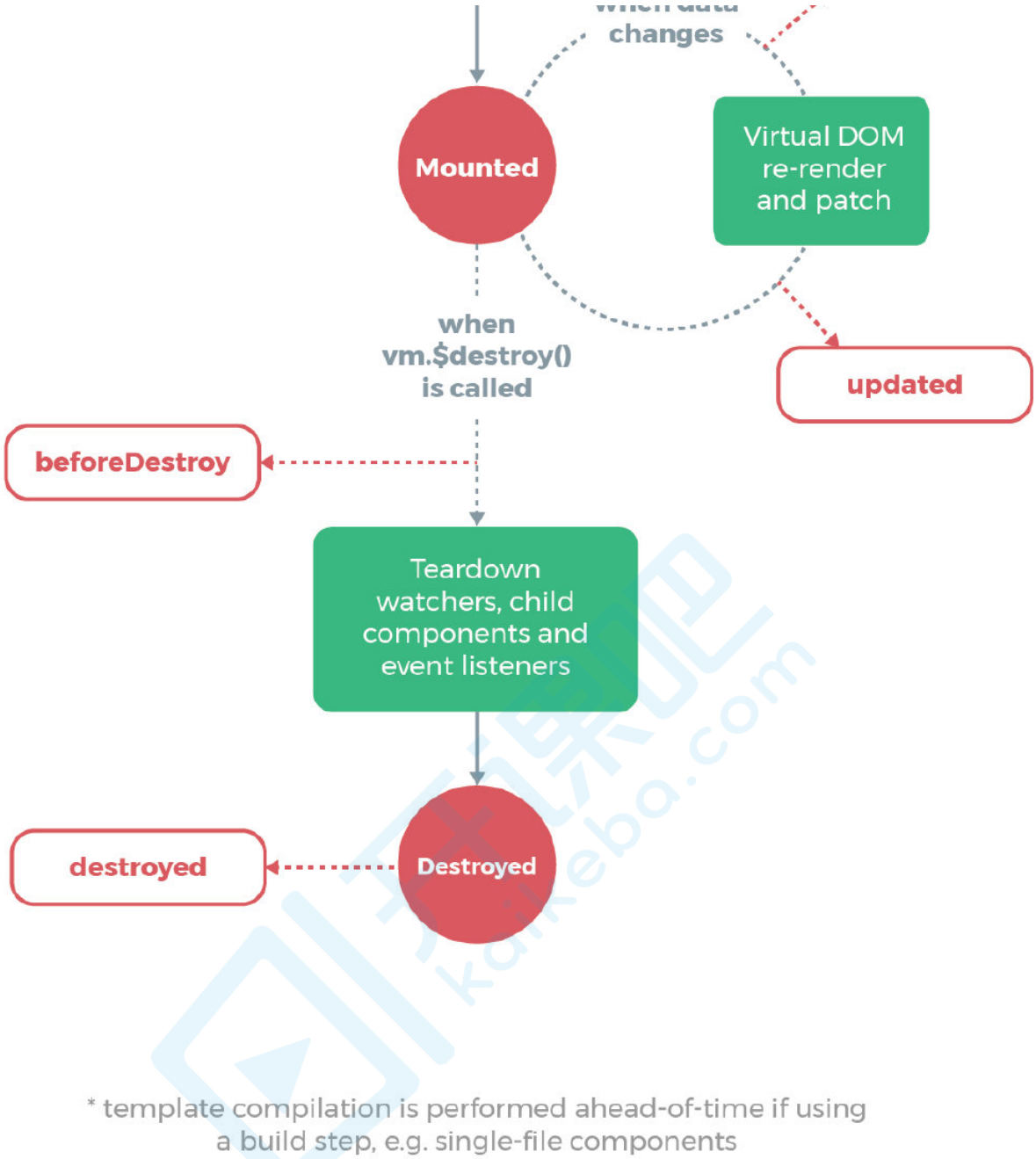
安全

1. xss
2. csrf
3. 密码安全

vue源码

生命周期


new Vue()
Init
Events & Lifecycle
beforeCreate
Init
injections & reactivity
created
Has
"el" option?
NO
when
vm.$mount(el)
is called
YES
Has
"template" option?
YES
NO
Compile template
into
render function *
Compile el's
outerHTML
as template *
beforeMount
Create vm.$el
and replace
"el" with it
mounted
beforeUpdate
when data



重点是响应式 defineProperty

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

  // 在deps中添加一个监听器对象
  addDep(dep) {
    if(this.deps.indexOf(dep)==-1){
      this.deps.push(dep)
    }
```

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


// 监听器
class Watcher {
  constructor(vm, key, cb) {
    // 在new一个监听器对象时将该对象赋值给Dep.target，在get中会用到
    // 将 Dep.target 指向自己
    // 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 KVue {
  constructor(options) {
    this.$data = options.data
    this.$options = options
    this.$store = options.store ||{}

    this.observer(this.$data)
    // 新建一个Watcher观察者对象，这时候Dep.target会指向这个Watcher对象
    // new Watcher()
    // 在这里模拟render的过程，为了触发test属性的get函数
    console.log('模拟render，触发test的getter', this.$data)
    if(options.created){
      options.created.call(this)
```

```
    }
    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, {
      enumerable: true,
      configurable: true,
      get() {
        // 将Dep.target（即当前的Watcher对象存入Dep的deps中）

        Dep.target && dep.addDep(Dep.target)
        return val
      },
      set(newVal) {
        if (newVal === val) return
        val = newVal

        // 在set的时候触发dep的notify来通知所有的Watcher对象更新视图
        dep.notify()
      }
    })
  }
  proxyData(key) {

    Object.defineProperty(this, key, {
      configurable: false,
      enumerable: true,
      get() {

        return this.$data[key]
      },
      set(newVal) {
        this.$data[key] = newVal
      }
    })
  }

}
```

compile

```
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
    }
    compileElement(el) {
      let childNodes = el.childNodes

      Array.from(childNodes).forEach((node) => {
        let text = node.textContent
        // 表达式文本
        // 就是识别{{}}中的数据
        let reg = /\{\{(.*)\}\}/
        // 按元素节点方式编译
        if (this.isElementNode(node)) {
          this.compile(node)
        } else if (this.isTextNode(node) && reg.test(text)) {
            // 文本 并且有{{}}
          this.compileText(node, RegExp.$1)

        }
        // 遍历编译子节点
        if (node.childNodes && node.childNodes.length) {
          this.compileElement(node)
        }
      })
    }

    compile(node) {
      let nodeAttrs = node.attributes
      Array.from(nodeAttrs).forEach( (attr)=>{
        // 规定：指令以 v-xxx 命名
        // 如 <span v-text="content"></span> 中指令为 v-text
        let attrName = attr.name    // v-text
        let exp = attr.value // content
        if (this.isDirective(attrName)) {
```

```
            let dir = attrName.substring(2) // text
            // 普通指令
            this[dir] && this[dir](node, this.$vm, exp)
        }
        if(this.isEventDirective(attrName)){
            let dir = attrName.substring(1) // text
            this.eventHandler(node, this.$vm, exp, dir)

        }
      })
    }
    compileText(node, exp) {
      this.text(node, this.$vm, exp)
    }

    isDirective(attr) {
      return attr.indexOf('k-') == 0
    }

    isEventDirective(dir) {
      return dir.indexOf('@') === 0
    }

    isElementNode(node) {
      return node.nodeType == 1
    }

    isTextNode(node) {
      return node.nodeType == 3
    }
    text(node, vm, exp) {
        this.update(node, vm, exp, 'text')
    }

    html(node, vm, exp) {
        this.update(node, vm, exp, 'html')
    }

    model(node, vm, exp) {
        this.update(node, vm, exp, 'model')
        let val = vm.exp
        node.addEventListener('input', (e)=>{
            let newValue = e.target.value
            vm[exp] = newValue
            val = newValue
        })
    }

    update(node, vm, exp, dir) {
        let updaterFn = this[dir + 'Updater']
        updaterFn && updaterFn(node, vm[exp])
        new Watcher(vm, exp, function(value) {
            updaterFn && updaterFn(node, value)
```

```
            })
        }

        // 事件处理
        eventHandler(node, vm, exp, dir) {
            let fn = vm.$options.methods && vm.$options.methods[exp]
            if (dir && fn) {
                node.addEventListener(dir, fn.bind(vm), false)
            }
        }
        textUpdater(node, value) {
            node.textContent = value
        }

        htmlUpdater(node, value) {
            node.innerHTML = value
        }

        modelUpdater(node, value) {
            node.value = value
        }
    }
```

react解析

1. jsx
2. 虚拟dom
3. setState
4. 单向数据流

三面

面试技巧

工程

团队合作

领导力

谈薪

职业生涯

个人技能树

1. 内力的修炼
2. git
3. 工程化

4. 算法数据结构

软技能

1. 英语
2. 如何成为一个高手
3. 刻意练习

https://zhuanlan.zhihu.com/p/23558753

回顾

职位诱惑：

上市公司,大牛团队,福利优厚,晋升空间大

职位描述：

岗位职责：

1、参与项目、产品需求分析与设计，负责前端架构设计及技术路线；

2、组织前端开发团队完成前端功能设计、开发、测试和交付。

3、负责前端核心功能的架构与代码模板编写，对系统核心模块进行开发和维护；

4、制定前端开发规范，参与制定技术标准，编写相应的技术文档，对通用技术进行整理；

5、负责前端开发人员的招聘和考核，定期开展前端开发工程师培训，促进团队成员的进步。

岗位要求：

1、5年以上Web前端研发经验；

2、能够熟练的设计和开发基于Javascript的复杂web应用并有实践经验；

3、必须精通VueJS或Angular2/3/4/5框架，熟悉ReactJs，NodeJs等框架，有产品设计经验优先；

4、熟练掌握gulp，webpack，browserify等工具，并且对实现细节有研究，有ionic，Electron等Hybird框架经验尤佳；

5、能凭借丰富的开发经验快速定位并解决各种前端问题；

6、具有微信公众号开发经验，了解小程序的开发；

7、具备良好的识别和设计通用框架及模块的能力；

8、逻辑思维强、注重团队协作；

工作地址

北京 - 海淀区 - 中关村　　查看地图

简历

技术栈

ES6

http协议

#

原型链

this

promise

简单算法

二面

js运行机制

页面性能

缓存

Last-Modified 和 If-Modified-Since

ETag 和 If-None-Match

安全

vue源码

react解析

三面

面试技巧

- 工程
- 团队合作
- 领导力
- 谈薪
- 职业生涯

个人技能树

软技能

回顾