Vue

1.vue.js 的两个核心是什么？

数据驱动和组件化。

2.vue 的双向绑定的原理是什么？

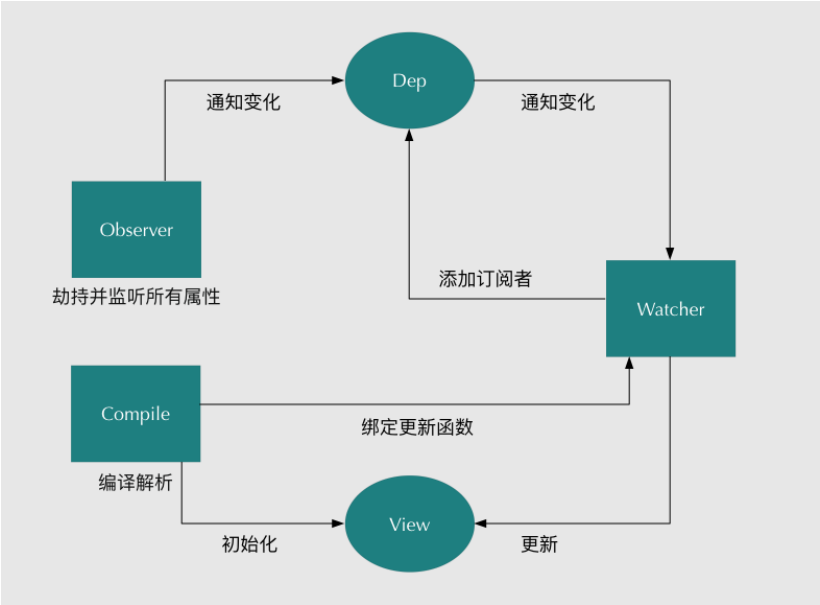
vue 数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的。具体实现过程：我们已经知道实现数据的双向绑定，首先要对数据进行劫持监听，所以我们需要设置一个监听器 Observer，用来监听所有属性。如果属性发上变化了，就需要告诉订阅者 Watcher 看是否需要更新。因为订阅者是有很多个，所以我们需要有一个消息订阅器 Dep 来专门收集这些订阅者，然后在监听器 Observer 和订阅者 Watcher 之间进行统一管理的。接着，我们还需要有一个指令解析器 Compile，对每个节点元素进行扫描和解析，将相关指令对应初始化成一个订阅者 Watcher，并替换模板数据或者绑定相应的函数，此时当订阅者 Watcher 接收到相应属性的变化，就会执行对应的更新函数，从而更新视图。因此接下去我们执行以下 3 个步骤，实现数据的双向绑定：

1.实现一个监听器 Observer，用来劫持并监听所有属性，如果有变动的，就通知订阅者。

2.实现一个订阅者 Watcher，可以收到属性的变化通知并执行相应的函数，从而更新视图。

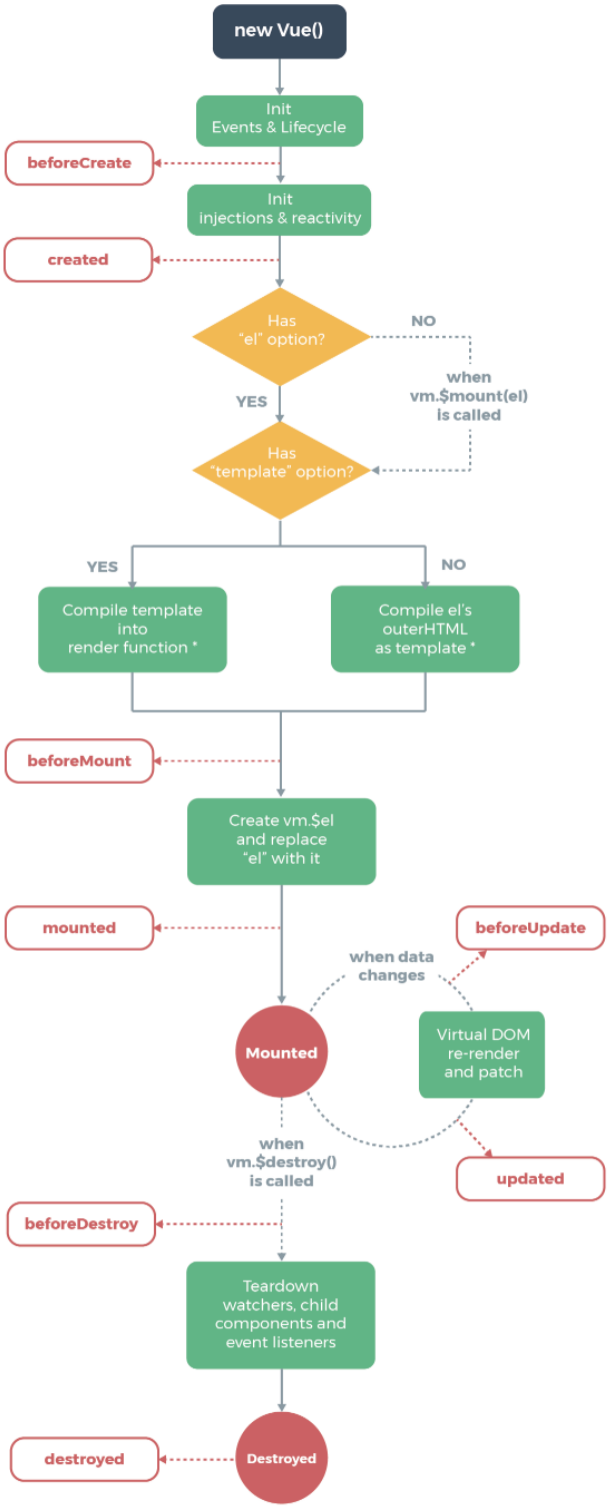
3.实现一个解析器 Compile，可以扫描和解析每个节点的相关指令，并根据初始化模板数据以及初始化相应的订阅器。

流程图如下：



3.vue 生命周期钩子函数有哪些？

总共分为 8 个阶段创建前/后，载入前/后，更新前/后，销毁前/后。具体执行流程查看下图。



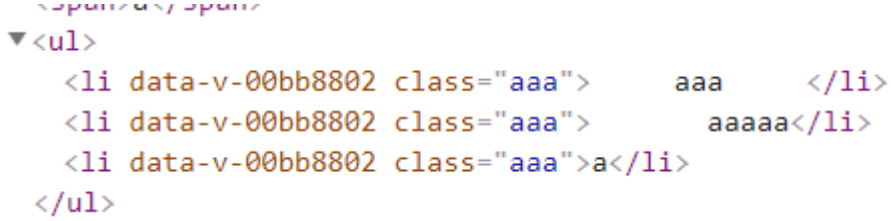
4.请问 v-if 和 v-show 有什么区别？

相同点： 两者都是在判断 DOM 节点是否要显示。
不同点：
a.实现方式： v-if 是根据后面数据的真假值判断直接从 Dom 树上删除或重建元素节点。v-show 只是在修改元素的 css 样式，也就是 display 的属性值，元素始终在 Dom 树上。
b.编译过程：v-if 切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件； v-show 只是简单的基于 css 切换；
c.编译条件：v-if 是惰性的，如果初始条件为假，则什么也不做；只有在条件第一次变为真时才开始局部编译； v-show 是在任何条件下（首次条件是否为真）都被编译，然后被缓存，而且 DOM 元素始终被保留；
d.性能消耗：v-if 有更高的切换消耗，不适合做频繁的切换； v-show 有更高的初始渲染消耗，适合做频繁的额切换。

5.vue 常用的修饰符

a、按键修饰符
如：.delete（捕获“删除”和”退格“键） 用法上和事件修饰符一样，挂载在 v-on: 后面，语法：v-on:keyup.xxx= ’ yyy ’ <inputclass = 'aaa' v-model="inputValue" @keyup.delete="onKey"/>
b、系统修饰符
可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器
.ctrl
.alt
.shift
.meta
c、鼠标按钮修饰符
.left
.right
.middle
这些修饰符会限制处理函数仅响应特定的鼠标按钮。如：<button @click.middle ="onClick">A</button> 鼠标滚轮单击触发 Click 默认是鼠标左键单击
d、其他修饰符
.lazy
在默认情况下，v-model 在每次 input 事件触发后将输入框的值与数据进行同步 ，我们可以添加 lazy 修饰符，从而转变为使用 change 事件进行同步：
<inputv-model.lazy="msg" >
.number
如果想自动将用户的输入值转为数值类型，可以给 v-model 添加 .number 修饰符：
<input v-model.number="age" type="number">

这通常很有用，因为即使在 type="number"时，HTML 输入元素的值也总会返回字符串。如果这个值无法被 parseFloat()解析，则会返回原始的值。
.trim
如果要自动过滤用户输入的首尾空白字符，可以给 v-model 添加 trim 修饰符：
<inputv-model.trim="msg">



同样前面都有空格加上.trim 后 将前后空格都去掉了。

6.nextTick

在下次 dom 更新循环结束之后执行延迟回调，可用于获取更新后的 dom 状态
新版本中默认是 microtasks, v-on 中会使用 macrotasks
macrotasks 任务的实现:
osetImmediate / MessageChannel / setTimeout

7.什么是 vue 生命周期

Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载 Dom→渲染、更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。

8.数据响应(数据劫持)

看完生命周期后，里面的 watcher 等内容其实是数据响应中的一部分。数据响应的实现由两部分构成: 观察者(watcher) 和 依赖收集器(Dep)，其核心是 defineProperty 这个方法，它可以 重写属性的 get 与 set 方法，从而完成监听数据的改变。
Observe (观察者)观察 props 与 state
o 遍历 props 与 state，对每个属性创建独立的监听器(watcher)
使用 defineProperty 重写每个属性的 get/set(defineReactive）
oget: 收集依赖
Dep.depend()
watcher.addDep()
oset: 派发更新

```
    Dep.notify()
    watcher.update()
    queenWatcher()
    nextTick
    flushScheduleQueue
    watcher.run()
    updateComponent()
```

大家可以先看下面的数据相应的代码实现后，理解后就比较容易看懂上面的简单脉络了。

```
let data = {a: 1}// 数据响应性
observe(data)
// 初始化观察者 new Watcher(data, 'name', updateComponent)
data.a = 2
// 简单表示用于数据更新后的操作 function updateComponent() {
    vm._update() // patchs
}
// 监视对象 function observe(obj) {
    // 遍历对象，使用 get/set 重新定义对象的每个属性值
    Object.keys(obj).map(key => {
        defineReactive(obj, key, obj[key])
    })
}
function defineReactive(obj, k, v) {
    // 递归子属性
    if (type(v) == 'object') observe(v)

    // 新建依赖收集器
    let dep = new Dep()
    // 定义 get/set
    Object.defineProperty(obj, k, {
        enumerable: true,
        configurable: true,
        get: function reactiveGetter() {
            // 当有获取该属性时，证明依赖于该对象，因此被添加进收集器中
            if (Dep.target) {
                dep.addSub(Dep.target)
            }
            return v
        },
        // 重新设置值时，触发收集器的通知机制
        set: function reactiveSetter(nV) {
            v = nV
            dep.nofify()
        },
    })
```

```
}
// 依赖收集器 class Dep {
    constructor() {
        this.subs = []
    }
    addSub(sub) {
        this.subs.push(sub)
    }
    notify() {
        this.subs.map(sub => {
            sub.update()
        })
    }
}

Dep.target = null
// 观察者 class Watcher {
    constructor(obj, key, cb) {
        Dep.target = this
        this.cb = cb
        this.obj = obj
        this.key = key
        this.value = obj[key]
        Dep.target = null
    }
    addDep(Dep) {
        Dep.addSub(this)
    }
    update() {
        this.value = this.obj[this.key]
        this.cb(this.value)
    }
    before() {
        callHook('beforeUpdate')
    }
}
```

9.virtual dom 原理实现

创建 dom 树
树的 diff，同层对比，输出 patchs(listDiff/diffChildren/diffProps)
o 没有新的节点，返回
o 新的节点 tagName 与 key 不变， 对比 props，继续递归遍历子树

对比属性(对比新旧属性列表):
旧属性是否存在与新属性列表中
都存在的是否有变化
是否出现旧列表中没有的新属性
otagName 和 key 值变化了，则直接替换成新节点
渲染差异
o 遍历 patchs， 把需要更改的节点取出来
o 局部更新 dom

```
// diff 算法的实现 function diff(oldTree, newTree) {
    // 差异收集
    let pathchs = {}
    dfs(oldTree, newTree, 0, pathchs)
    return pathchs
}
function dfs(oldNode, newNode, index, pathchs) {
    let curPathchs = []
    if (newNode) {
        // 当新旧节点的 tagName 和 key 值完全一致时
        if (oldNode.tagName === newNode.tagName && oldNode.key === newNode.key) {
            // 继续比对属性差异
            let props = diffProps(oldNode.props, newNode.props)
            curPathchs.push({ type: 'changeProps', props })
            // 递归进入下一层级的比较
            diffChildrens(oldNode.children, newNode.children, index, pathchs)
        } else {
            // 当 tagName 或者 key 修改了后，表示已经是全新节点，无需再比
            curPathchs.push({ type: 'replaceNode', node: newNode })
        }
    }

    // 构建出整颗差异树
    if (curPathchs.length) {
        if(pathchs[index]){
            pathchs[index] = pathchs[index].concat(curPathchs)
        } else {
            pathchs[index] = curPathchs
        }
    }
}
// 属性对比实现 function diffProps(oldProps, newProps) {
    let propsPathchs = []
    // 遍历新旧属性列表
    // 查找删除项
    // 查找修改项
```

```
    // 查找新增项
    forin(olaProps, (k, v) => {
        if (!newProps.hasOwnProperty(k)) {
            propsPathchs.push({ type: 'remove', prop: k })
        } else {
            if (v !== newProps[k]) {
                propsPathchs.push({ type: 'change', prop: k , value: newProps[k] })
            }
        }
    })
    forin(newProps, (k, v) => {
        if (!oldProps.hasOwnProperty(k)) {
            propsPathchs.push({ type: 'add', prop: k, value: v })
        }
    })
    return propsPathchs
}
// 对比子级差异 function diffChildrens(oldChild, newChild, index, pathchs) {
        // 标记子级的删除/新增/移动
    let { change, list } = diffList(oldChild, newChild, index, pathchs)
    if (change.length) {
        if (pathchs[index]) {
            pathchs[index] = pathchs[index].concat(change)
        } else {
            pathchs[index] = change
        }
    }

     // 根据 key 获取原本匹配的节点，进一步递归从头开始对比
    oldChild.map((item, i) => {
        let keyIndex = list.indexOf(item.key)
        if (keyIndex) {
            let node = newChild[keyIndex]
            // 进一步递归对比
            dfs(item, node, index, pathchs)
        }
    })
}
// 列表对比，主要也是根据 key 值查找匹配项// 对比出新旧列表的新增/删除/移动
function diffList(oldList, newList, index, pathchs) {
    let change = []
    let list = []
    const newKeys = getKey(newList)
    oldList.map(v => {
```

```
            if (newKeys.indexOf(v.key) > -1) {
                list.push(v.key)
            } else {
                list.push(null)
            }
        })

        // 标记删除
        for (let i = list.length - 1; i>= 0; i--) {
            if (!list[i]) {
                list.splice(i, 1)
                change.push({ type: 'remove', index: i })
            }
        }

        // 标记新增和移动
        newList.map((item, i) => {
            const key = item.key
            const index = list.indexOf(key)
            if (index === -1 || key == null) {
                // 新增
                change.push({ type: 'add', node: item, index: i })
                list.splice(i, 0, key)
            } else {
                // 移动
                if (index !== i) {
                    change.push({
                        type: 'move',
                        form: index,
                        to: i,
                    })
                    move(list, index, i)
                }
            }
        })

        return { change, list }
}
```

10.Proxy 相比于 defineProperty 的优势

数组变化也能监听到

不需要深度遍历监听

```
let data = { a: 1 }let reactiveData = new Proxy(data, {
    get: function(target, name){
        // ...
    },
    // ...
})
```

6. vue-router

mode

ohash

ohistory

跳转

othis.$router.push()

o<router-link to=""></router-link>

占位

o<router-view></router-view>

11.vuex

state: 状态中心
mutations: 更改状态
actions: 异步更改状态
getters: 获取状态
modules: 将 state 分成多个 modules，便于管理

12.vue 中 key 值的作用

使用 key 来给每个节点做一个唯一标识
key 的作用主要是为了高效的更新虚拟 DOM。另外 vue 中在使用相同标签名元素的过渡切换时，也会使用到 key 属性，其目的也是为了让 vue 可以区分它们，
否则 vue 只会替换其内部属性而不会触发过渡效果。

13.Vue 组件中 data 为什么必须是函数？

在 new Vue() 中，data 是可以作为一个对象进行操作的，然而在 component 中，data 只能以函数的形式存在，不能直接将对象赋值给它。
当 data 选项是一个函数的时候，每个实例可以维护一份被返回对象的独立的拷贝，这样各

个实例中的 data 不会相互影响，是独立的。

14.v-for 与 v-if 的优先级

v-for 的优先级比 v-if 高。

15.说出至少 4 种 vue 当中的指令和它的用法

v-if(判断是否隐藏)
v-for(把数据遍历出来)
v-bind(绑定属性)
v-model(实现双向绑定)

16.vue 中子组件调用父组件的方法

第一种方法是直接在子组件中通过 this.$parent.event 来调用父组件的方法。
第二种方法是在子组件里用$emit 向父组件触发一个事件，父组件监听这个事件就行了。
第三种是父组件把方法传入子组件中，在子组件里直接调用这个方法。

17.vue 中父组件调用子组件的方法

父组件利用 ref 属性操作子组件方法。
父：

```
<child ref="childMethod"></child>
```

子：

```
method: {
   test() {
        alert(1)
   }
}
```

在父组件里调用 test 即 this.$refs.childMethod.test()

18.vue 页面级组件之间传值

1.使用 vue-router 通过跳转链接带参数传参。
2.使用本地缓存 localStorge。
3.使用 vuex 数据管理传值。

19.说说 vue 的动态组件。

多个组件通过同一个挂载点进行组件的切换，is 的值是哪个组件的名称，那么页面就会显示哪个组件。
主要考查面试这 component 的 is 属性。

20.keep-alive 内置组件的作用

可以让当前组件或者路由不经历创建和销毁，而是进行缓存，凡是被 keep-alive 组件包裹的组件，除了第一次以外。不会经历创建和销毁阶段的。第一次创建后就会缓存到缓存当中。

21.递归组件的用法

组件是可以在它们自己的模板中调用自身的。不过它们只能通过 name 选项来做这件事。首先我们要知道，既然是递归组件，那么一定要有一个结束的条件，否则就会使用组件循环引用，最终出现“max stack size exceeded”的错误，也就是栈溢出。那么，我们可以使用 v-if="false"作为递归组件的结束条件。当遇到 v-if 为 false 时，组件将不会再进行渲染。

22.怎么定义 vue-router 的动态路由？怎么获取传过来的值？

动态路由的创建，主要是使用 path 属性过程中，使用动态路径参数，以冒号开头，如下：

```
{
   path: '/details/:id'
name: 'Details'
```

components: Details
}
访问 details 目录下的所有文件，如果 details/a，details/b 等，都会映射到 Details 组件上。
当匹配到/details 下的路由时，参数值会被设置到 this.$route.params 下，所以通过这个属性可以获取动态参数
this.$route.params.id

23.vue-router 有哪几种路由守卫?

路由守卫为：
全局守卫：beforeEach
后置守卫：afterEach
全局解析守卫：beforeResolve
路由独享守卫：beforeEnter

24.$route 和 $router 的区别是什么？

$router 为 VueRouter 的实例，是一个全局路由对象，包含了路由跳转的方法、钩子函数等。
$route 是路由信息对象||跳转的路由对象，每一个路由都会有一个 route 对象，是一个局部对象，包含 path,params,hash,query,fullPath,matched,name 等路由信息参数。

25. vue-router 响应路由参数的变化

```
// 监听当前路由发生变化的时候执行
watch: {
  $route(to, from){
    console.log(to.path)
    // 对路由变化做出响应
  }
}
```

(1)用 watch 检测
(2)组件内导航钩子函数

```
beforeRouteUpdate(to, from, next){
  // to do somethings
}
```

26. vue-router 传参

(1)使用 Params:
只能使用 name，不能使用 path
参数不会显示在路径上
浏览器强制刷新参数会被清空

```
// 传递参数
this.$router.push({
  name: Home,
  params: {
      number: 1 ,
      code: '999'
  }
})
// 接收参数
const p = this.$route.params
```

（2）使用 Query：
参数会显示在路径上，刷新不会被清空
name 可以使用 path 路径

```
// 传递参数
this.$router.push({
  name: Home,
  query: {
  number: 1 ,
  code: '999'
}
                  })
// 接收参数
const q = this.$route.query
```

27.不用 Vuex 会带来什么问题？

一、可维护性会下降，你要想修改数据，你得维护三个地方
二、可读性会下降，因为一个组件里的数据，你根本就看不出来是从哪来的
三、增加耦合，大量的上传派发，会让耦合性大大的增加，本来 Vue 用 Component 就是为了减少耦合，现在这么用，和组件化的初衷相背。

28.vuex 有哪几种属性？

有五种，分别是 State、 Getter、Mutation 、Action、 Module。

29.vuex 的 State 特性是？

一、Vuex 就是一个仓库，仓库里面放了很多对象。其中 state 就是数据源存放地，对应于与一般 Vue 对象里面的 data
二、state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新
三、它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中

30.　vuex 的 Getter 特性是？

一、getters 可以对 State 进行计算操作，它就是 Store 的计算属性
二、 虽然在组件内也可以做计算属性，但是 getters 可以在多组件之间复用
三、 如果一个状态只在一个组件内使用，是可以不用 getters

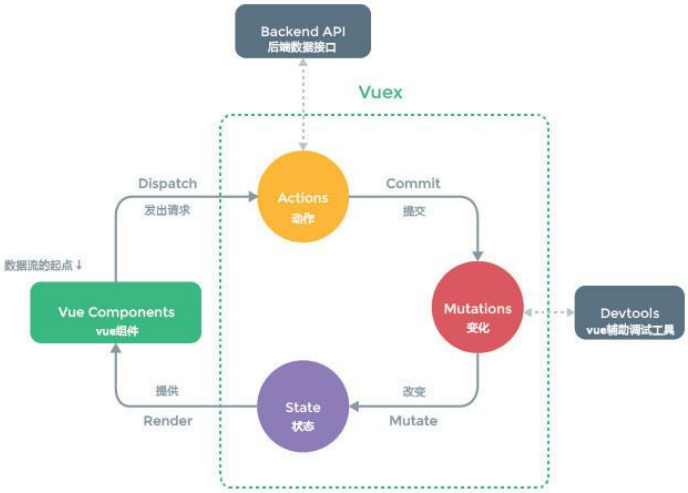
31.vuex 的 Mutation 特性是？

一、Action 类似于 mutation，不同在于：
二、Action 提交的是 mutation，而不是直接变更状态。
三、Action 可以包含任意异步操作

32.Vue.js 中 ajax 请求代码应该写在组件的 methods 中还是 vuex 的 actions 中？

一、如果请求来的数据是不是要被其他组件公用，仅仅在请求的组件内使用，就不需要放入 vuex 的 state 里。

二、如果被其他地方复用，这个很大几率上是需要的，如果需要，请将请求放入 action 里，方便复用，并包装成 promise 返回，在调用处用 async await 处理返回的数据。如果不要复用这个请求，那么直接写在 vue 文件里很方便。



33.什么是 MVVM？

MVVM 是 Model-View-ViewModel 的缩写。MVVM 是一种设计思想。Model 层代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑；View 代表 UI 组件，它负责将数据模型转化成 UI 展现出来，ViewModel 是一个同步 View 和 Model 的对象。
在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互，Model 和 ViewModel 之间的交互是双向的， 因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反应到 View 上。
ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而 View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作 DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

34.mvvm 和 mvc 区别？它和其它框架（jquery）的区别是什么？哪些场景适合？

mvc 和 mvvm 其实区别并不大。都是一种设计思想。主要就是 mvc 中 Controller 演变成 mvvm 中的 viewModel。mvvm 主要解决了 mvc 中大量的 DOM 操作使页面渲

染性能降低，加载速度变慢，影响用户体验。
区别：vue 数据驱动，通过数据来显示视图层而不是节点操作。
场景：数据操作比较多的场景，更加便捷。

35.vue 的优点是什么？

低耦合。视图（View）可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的"View"上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。
可重用性。你可以把一些视图逻辑放在一个 ViewModel 里面，让很多 view 重用这段视图逻辑。
独立开发。开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。
可测试。界面素来是比较难于测试的，而现在测试可以针对 ViewModel 来写。

36.组件之间的传值？

父组件与子组件传值
父组件通过标签上面定义传值
子组件通过 props 方法接受数据
子组件向父组件传递数据
子组件通过$emit 方法传递参数

37.路由之间跳转

声明式（标签跳转） 编程式（ js 跳转）

38.vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？

第一步：在 components 目录新建你的组件文件（indexPage.vue），script 一定要 export default {}
第二步：在需要用的页面（组件）中导入：import indexPage from

‘@/components/indexPage.vue’
第三步：注入到 vue 的子组件的 components 属性上面,components:{indexPage}
第四步：在 template 视图 view 中使用，
例如有 indexPage 命名，使用的时候则 index-page

39.vue 如何实现按需加载配合 webpack 设置

webpack 中提供了 require.ensure()来实现按需加载。以前引入路由是通过 import 这样的方式引入，改为 const 定义的方式进行引入。
不进行页面按需加载引入方式：import home from '../../common/home.vue'
进行页面按需加载的引入方式：

```
const home = r => require.ensure( [], () => r (require('../../common/home.vue')))
```

40.Vue 中引入组件的步骤?

1）采用 ES6 的 import … from …语法或 CommonJS 的 require()方法引入组件
2）对组件进行注册,代码如下

```
// 注册
Vue.component('my-component', {
    template: '<div>A custom component!</div>'
})
```

3）使用组件

41. 指令 v-el 的作用是什么?

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标.可以是 CSS 选择器，也可以是一个 HTMLElement 实例。

42. 在 Vue 中使用插件的步骤

采用 ES6 的 import … from …语法或 CommonJSd 的 require()方法引入插件
使用全局方法 Vue.use(plugin)使用插件,可以传入一个选项对象 Vue.use(MyPlugin, { someOption: true })

43.vue 生命周期的作用是什么

它的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。

44.vue 生命周期总共有几个阶段

可以总共分为 8 个阶段：创建前/后，载入前/后,更新前/后,销毁前/销毁后

45.第一次页面加载会触发哪几个钩子

第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子

46.DOM 渲染在 哪个周期中就已经完成

DOM 渲染在 mounted 中就已经完成了。

47.简单描述每个周期具体适合哪些场景

生命周期钩子的一些使用方法：
beforecreate : 可以在这加个 loading 事件，在加载实例时触发
created : 初始化完成时的事件写在这里，如在这结束 loading 事件，异步请求也适宜在这里调用
mounted : 挂载元素，获取到 DOM 节点
updated : 如果对数据统一处理，在这里写上相应函数
beforeDestroy : 可以做一个确认停止事件的确认框
nextTick : 更新数据后立即操作 dom

48.vue-loader 是什么？使用它的用途有哪些？

解析.vue 文件的一个加载器。
用途：js 可以写 es6、style 样式可以 scss 或 less、template 可以加 jade 等

49.scss 是什么？在 vue.cli 中的安装使用步骤是？有哪几大特性？

css 的预编译。
使用步骤：
第一步：先装 css-loader、node-loader、sass-loader 等加载器模块
第二步：在 build 目录找到 webpack.base.config.js，在那个 extends 属性中加一个拓展.scss
第三步：在同一个文件，配置一个 module 属性
第四步：然后在组件的 style 标签加上 lang 属性 ，例如：lang=”scss”
特性:
可以用变量，例如（$变量名称=值）；
可以用混合器，例如（）
可以嵌套

50.为什么使用 key？

当有相同标签名的元素切换时，需要通过 key 特性设置唯一的值来标记以让 Vue 区分它们，否则 Vue 为了效率只会替换相同标签内部的内容。

51.为什么避免 v-if 和 v-for 用在一起

当 Vue 处理指令时，v-for 比 v-if 具有更高的优先级，通过 v-if 移动到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 v-if 为否的时候运算 v-for。

52.VNode 是什么？虚拟 DOM 是什么？

Vue 在 页面上渲染的节点，及其子节点称为“虚拟节点 (Virtual Node)”，简写为“VNode”。“虚拟 DOM”是由 Vue 组件树建立起来的整个 VNode 树的称呼。

53.vue-loader 是什么？使用它的用途有哪些？

解析.vue 文件的一个加载器，跟 template/js/style 转换成 js 模块。
用途：js 可以写 es6、style 样式可以 scss 或 less、template 可以加 jade 等

54.请说出 vue.cli 项目中 src 目录每个文件夹和文件的用法？

assets 文件夹是放静态资源；components 是放组件；router 是定义路由相关的配置;view 视图；app.vue 是一个应用主组件；main.js 是入口文件

55.vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？

第一步：在 components 目录新建你的组件文件（smithButton.vue），script 一定要 export default {
第二步：在需要用的页面（组件）中导入：import smithButton from ‘…/components/smithButton.vue’
第三步：注入到 vue 的子组件的 components 属性上面,components:{smithButton}
第四步：在 template 视图 view 中使用，
问题有：smithButton 命名，使用的时候则 smith-button。

56.聊聊你对 Vue.js 的 template 编译的理解？

简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点）
详情步骤：
首先，通过 compile 编译器把 template 编译成 AST 语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用

以创建编译器的。另外 compile 还负责合并 option。
然后，AST 会经过 generate（将 AST 语法树转化成 render funtion 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）

57.vue 路由跳转的几种方式

1、先绑定路由

```
const RouterModel = new Router({
  routes: [
        {
          path: '/cart',
          name: 'cartTest',
          meta: {
              keepAlive: true
          },
          components: {
              default: asyncLoader('cart/shop-cart')
          }
    }
  ]
});
```

2、不带参数路由跳转

```
<router-link :to="{name: 'cartTest'}">name 不带参数</router-link>
<router-link :to="{path: '/cart'}">path 不带参数</router-link>
```

3、 带参数路由跳转

```
<router-link :to="{name: 'cartTest', params: {id: 1}}">name 带参数 params 传参数 (类似 post),html 取参 $route.params.id script, 取参 this.$route.params.id</router-link>
<router-link :to="{name: 'cartTest', query: {id: 1}}">name 带参数,query 传参数 (类似 get,url 后面会显示参数),html 取参 $route.query.id, script 取参 this.$route.query.id</router-link>
```

4、 this.$router.push() 函数内调用

不带参数

```
@click="this.$router.push({name: 'cartTest'})"
@click="this.$router.push({path: '/cart'})"
@click="this.$router.push('/cart')"
```

带参数

```
@click="this.$router.push({name: 'cartTest', params: {id: 1}})"
@click="this.$router.push({name: 'cartTest', query: {id: 1}})"
```

说明：

query 和 params 区别

query 类似 get, 跳转之后页面 url 后面会拼接参数,类似?id=1, 非重要性的可以这样传, 密码之类还是用 params 刷新页面 id 还在

params 类似 post, 跳转之后页面 url 后面不会拼接参数 , 但是刷新页面 id 会消失

58.vue-cli 创建自定义组件

1、新建一个.vue 文件 (一般 IDE 会自动生成 script export default)
2、编写 dom, 组件的 name 使用短横线分隔（component-name）方式。
3、在需要的页面 import componentName from ‘…’
4、组件注册，compontens :{ [componentName.name]: componentName}
5、在需要的页面 template 中显示

59.<keep-alive</keep-alive 的作用是什么?

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染。
在 vue 2.1.0 版本之后，keep-alive 新加入了两个属性：include(包含的组件缓存) 与 exclude(排除的组件不缓存，优先级大于 include) 。
使用方法

```
<keep-alive include='include_components' exclude='exclude_components'>
  <component>
    <!-- 该组件是否缓存取决于 include 和 exclude 属性 -->
  </component>
</keep-alive>
```

参数解释
include - 字符串或正则表达式，只有名称匹配的组件会被缓存
exclude - 字符串或正则表达式，任何名称匹配的组件都不会被缓存
include 和 exclude 的属性允许组件有条件地缓存。二者都可以用“，”分隔字符串、正则表达式、数组。当使用正则或者是数组时，要记得使用 v-bind 。
使用示例

```
<!-- 逗号分隔字符串，只有组件 a 与 b 被缓存。 -->
<keep-alive include="a,b">
  <component></component>
</keep-alive>
<!-- 正则表达式 (需要使用 v-bind，符合匹配规则的都会被缓存) -->
<keep-alive :include="/a|b/">
  <component></component>
</keep-alive>
<!-- Array (需要使用 v-bind，被包含的都会被缓存) -->
<keep-alive :include="['a', 'b']">
  <component></component>
</keep-alive>
```

大白话: 比如有一个列表和一个详情，那么用户就会经常执行打开详情=>返回列表=>打开详情…这样的话列表和详情都是一个频率很高的页面，那么就可以对列表组件使用 <keep-alive</keep-alive 进行缓存，这样用户每次返回列表的时候，都能从缓存中快速渲染，

而不是重新渲染。

60.vue 如何实现按需加载配合 webpack 设置?

webpack 中提供了 require.ensure()来实现按需加载。以前引入路由是通过 import 这样的方式引入，改为 const 定义的方式进行引入。
不进行页面按需加载引入方式：
import home from '../../common/home.vue'
进行页面按需加载的引入方式：
const home = r => require.ensure([], () => r (require('../../common/home.vue')))

61.Vue 实现数据双向绑定的原理 Object.defineProperty()

vue 实现数据双向绑定主要是：采用数据劫持结合发布者-订阅者模式的方式，通过**Object.defineProperty（）**来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应监听回调。当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。
vue 的数据双向绑定 将 MVVM 作为数据绑定的入口，整合 Observer，Compile 和 Watcher 三者，通过 Observer 来监听自己的 model 的数据变化，通过 Compile 来解析编译模板指令（vue 中是用来解析 {{msg}}），最终利用 watcher 搭起 observer 和 Compile 之间的通信桥梁，达到数据变化 —>视图更新；视图交互变化（input）—>数据 model 变更双向绑定效果。

62.Vue 的路由实现：hash 模式和 history 模式

hash 模式
在浏览器中符号“#”，#以及#后面的字符称之为 hash，用 window.location.hash 读取；
特点：hash 虽然在 URL 中，但不被包括在 HTTP 请求中；用来指导浏览器动作，对服务端安全无用，hash 不会重加载页面。
history 模式
history 采用 HTML5 的新特性；如果你使用 history 的话，改变路由的时候，后台会给你响应；且提供了两个新方法：pushState（），replaceState（）可以对浏览器历史记录栈进行修改，以及 popState 监听浏览器历史记录变化，但是 pushState(),replaceState()不会触发该函数。

63.Vue 与 Angular 以及 React 的区别？

1.与 AngularJS 的区别
相同点：
都支持指令：内置指令和自定义指令。
都支持过滤器：内置过滤器和自定义过滤器。
都支持双向数据绑定。
都不支持低端浏览器。
不同点：
1.AngularJS 的学习成本高，比如增加了 Dependency Injection 特性，而 Vue.js 本身提供的 API 都比较简单、直观。
2.在性能上，AngularJS 依赖对数据做脏检查，所以 Watcher 越多越慢。
Vue.js 使用基于依赖追踪的观察并且使用异步队列更新。所有的数据都是独立触发的。 对于庞大的应用来说，这个优化差异还是比较明显的。
2.与 React 的区别
相同点：
1.React 采用特殊的 JSX 语法，Vue.js 在组件开发中也推崇编写.vue 特殊文件格式，对文件内容都有一些约定，两者都需要编译后使用。
2.中心思想相同：一切都是组件，组件实例之间可以嵌套。
3.都提供合理的钩子函数，可以让开发者定制化地去处理需求。
4.都不内置 AJAX，Route 等功能到核心包，而是以插件的方式加载。 5.在组件开发中都支持 mixins 的特性。
不同点：
React 依赖 Virtual DOM,而 Vue.js 使用的是 DOM 模板。React 采用的 Virtual DOM 会对渲染出来的结果做脏检查。
Vue.js 在模板中提供了指令，过滤器等，可以非常方便，快捷地操作 DOM。

64.vue 路由的钩子函数

第一种：全局导航钩子
router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。
第二种：组件内的钩子；
beforeRouteEnter
beforeRouteUpdate (2.2 新增)
beforeRouteLeave
第三种：单独路由独享组件。
beforeEnter
每个钩子方法接收三个参数：
to: Route: 即将要进入的目标 路由对象
from: Route: 当前导航正要离开的路由

next: Function: 一定要调用该方法来 resolve 这个钩子。执行效果依赖 next 方法的调用参数。

65.route 和 router 的区别

route 是“路由信息对象”，包括 path，params，hash，query，fullPath，matched，name 等路由信息参数。而 router 是“路由实例”对象包括了路由的跳转方法，钩子函数等。

66.什么是 vue 的计算属性？

在模板中放入太多的逻辑会让模板过重且难以维护，在需要对数据进行复杂处理，且可能多次使用的情况下，尽量采取计算属性的方式。好处：①使得数据处理结构清晰；②依赖于数据，数据更新，处理结果自动更新；③计算属性内部 this 指向 vm 实例；④在 template 调用时，直接写计算属性名即可；⑤常用的是 getter 方法，获取数据，也可以使用 set 方法改变数据；⑥相较于 methods，不管依赖的数据变不变，methods 都会重新计算，但是依赖数据不变的时候 computed 从缓存中获取，不会重新计算。

67.vue 等单页面应用（spa）及其优缺点

优点： Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。MVVM、数据驱动、组件化、轻量、简洁、高效、快速、模块友好；即第一次就将所有的东西都加载完成，因此，不会导致页面卡顿。
缺点： 不支持低版本的浏览器，最低只支持到 IE9；不利于 SEO 的优化（如果要支持 SEO，建议通过服务端来进行渲染组件）；第一次加载首页耗时相对长一些；不可以使用浏览器的导航按钮需要自行实现前进、后退。

68.vue-cli 如何新增自定义指令？

1.创建局部指令

```
var app = new Vue({
    el: '#app',
    data: {
    },
    // 创建指令(可以多个)
```

```
    directives: {
        // 指令名称
        dir1: {
            inserted(el) {
                // 指令中第一个参数是当前使用指令的 DOM
                console.log(el);
                console.log(arguments);
                // 对 DOM 进行操作
                el.style.width = '200px';
                el.style.height = '200px';
                el.style.background = '#000';
            }
        }
    }
})
2.全局指令
Vue.directive('dir2', {
    inserted(el) {
        console.log(el);
    }
})
3.指令的使用

<div id="app">
    <div v-dir1></div>
    <div v-dir2></div>
</div>
```

69.v-on 可以绑定多个方法吗？

可以

70. vue 中 key 值的作用？

避免 dom 节点复用，让每一次数据改变都重新渲染 dom 节点
当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序， 而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。key 的作用主要是为了高效的更新虚拟 DOM。

71.active-class 是哪个组件的属性？嵌套路由怎么定义？

vue-router 模块的 router-link 组件的属性。
嵌套路由：
嵌套路由顾名思义就是路由的多层嵌套。
重构 router/index.js 的路由配置，需要使用 children 数组来定义子路由 ，路由定义：

```
{
        path: '/me',
        name: 'Me',
        component: Me,
        children: [
          {
            path: 'collection',
            name: 'Collection',
            component: Collection
          },
          {
            path: 'trace',
            name: 'Trace',
            component: Trace
          }
        ]
      }
```

以“/”开头的嵌套路径会被当作根路径，所以子路由上不用加“/”;
在生成路由时，主路由上的 path 会被自动添加到子路由之前，所以子路由上的 path 不用在重新声明主路由上的 path 了。
在外层路由组件中，如下写法。

```
<template>
  <div class="me">
    <div class="tabs">
      <ul>
        <!--<router-link :to="{name: 'Default'}" tag="li" exact>默认内容</router-link>-->
        <router-link :to="{name: 'Collection'}" tag="li" >我的收藏</router-link>
        <router-link :to="{name: 'Trace'}" tag="li">我的足迹</router-link>
      </ul>
    </div>
    <div class="content">
      <router-view></router-view>
    </div>
  </div>
```

</template>

72.axios 是什么？怎么使用？描述使用它实现登录功能的流程？

请求后台资源的模块。npm install axios -S 装好，然后发送的是跨域，需在配置文件中 config/index.js 进行设置。后台如果是 Tp5 则定义一个资源路由。js 中使用 import 进来，然后.get 或.post。返回在.then 函数中如果成功，失败则是在.catch 函数中。

73.axios+tp5 进阶中，调用 axios.post(‘api/user’)是进行的什么操作？axios.put(‘api/user/8′)呢？

跨域，添加用户操作，更新操作。

74.什么是 RESTful API？怎么使用？

是一个 api 的标准，无状态请求。请求的路由地址是固定的，如果是 tp5 则先路由配置中把资源路由配置好。标准有：.post .put .delete

75.请说下封装 vue 组件的过程？

首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决了我们传统项目开发：效率低、难维护、复用性等问题。
然后，使用 Vue.extend 方法创建一个组件，然后使用 Vue.component 方法注册组件。子组件需要数据，可以在 props 中接受定义。而子组件修改好数据后，想把数据传递给父组件。可以采用 emit 方法。

76.watch 和 computed 区别

区别一：
watch 可以允许你没有返回值，对数据做一些处理，
但是 computed 必须要有返回值，他才能根据返回值，知道你函数依赖的是谁，当哪个数据改变的时候，触发该方法。
区别二：
computed 可以函数依赖很多值，但是 watch 只能依赖一个值。

77.vuex 有哪几种属性？

一、Vuex 就是一个仓库，仓库里面放了很多对象。其中 state 就是数据源存放地，对应于与一般 Vue 对象里面的 data
二、state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新
三、它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中。

78.vuex 的 Mutation 特性是？

一、Action 类似于 mutation，不同在于：
二、Action 提交的是 mutation，而不是直接变更状态。
三、Action 可以包含任意异步操作

79.mint-ui 或其他前端组件库在 vue 中怎么使用

基于 vue 的前端组件库通过 npm 安装，然后 import 样式和 js，vue.use（mintUi）全局引入。在单个组件局部引入：import {Toast} from 'mint-ui'。

80.自定义指令（v-check、v-focus）的方法有哪些？它有哪些钩子函数？还有哪些钩子函数参数？

全局定义指令：在 vue 对象的 directive 方法里面有两个参数，一个是指令名称，另外一个是函数。组件内定义指令：directives
钩子函数：bind（绑定事件触发）、inserted(节点插入的时候触发)、update（组件内相关更新）
钩子函数参数：el、binding。

80.pwa 是什么？

渐进式网页应用，PWA 应该具有一下特性：

渐进式：能确保每个用户都能打开网页响应式：PC，手机，平板，不管哪种格式，网页格式都能完美适配
2.离线应用：支持用户在没网的条件下也能打开网页，这里就需要 Service Worker 的帮助
3.APP 化：能够像 APP 一样和用户进行交互
4.常更新：一旦 Web 网页有什么改动，都能立即在用户端体现出来
5.安全：安全第一，给自己的网站加上一把绿锁–HTTPS
6.可搜索：能够被引擎搜索到
7.推送：做到在不打开网页的前提下，推送新的消息
8.可安装：能够将 Web 想 APP 一样添加到桌面
9.可跳转：只要通过一个连接就可以跳转到你的 Web 页面

81.怎么定义 vue-router 的动态路由？怎么获取传过来的动态参数？

在 router 目录下的 index.js 文件中，对 path 属性加上/:id。
使用 router 对象的 params.id。

82.v-model 是什么？怎么使用？ vue 中标签怎么绑定事件？

可以实现双向绑定，指令（v-class、v-for、v-if、v-show、v-on）。vue 的 model 层的 data 属性。绑定事件：<input @click=doLog()/>

83.iframe 的优缺点？

iframe 也称作嵌入式框架，嵌入式框架和框架网页类似，它可以把一个网页的框架和内容嵌入在现有的网页中。
优点：
解决加载缓慢的第三方内容如图标和广告等的加载问题
Security sandbox
并行加载脚本
方便制作导航栏
缺点：
iframe 会阻塞主页面的 Onload 事件

即时内容为空，加载也需要时间
没有语意

84.简述一下 Sass、Less，且说明区别？

他们是动态的样式语言，是 CSS 预处理器,CSS 上的一种抽象层。他们是一种特殊的语法/语言而编译成 CSS。
变量符不一样，less 是@，而 Sass 是$;
Sass 支持条件语句，可以使用 if{}else{},for{}循环等等。而 Less 不支持;
Sass 是基于 Ruby 的，是在服务端处理的，而 Less 是需要引入 less.js 来处理 Less 代码输出 Css 到浏览器。

85.vuex 是什么？怎么使用？哪种功能场景使用它？

vue 框架中状态管理。在 main.js 引入 store，注入。新建了一个目录 store，….. export 。场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

86.vue-router 是什么？它有哪些组件？

vue 用来写路由一个插件。router-link、router-view。

87.Vue 的双向数据绑定原理是什么？

vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty()来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。
具体步骤：
第一步：需要 observe 的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter 和 getter
这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化
第二步：compile 解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图
第三步：Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁，主要做的事情是:
1、在自身实例化时往属性订阅器(dep)里面添加自己

2、自身必须有一个 update()方法
3、待属性变动 dep.notice()通知时，能调用自身的 update()方法，并触发 Compile 中绑定的回调，则功成身退。
第四步：MVVM 作为数据绑定的入口，整合 Observer、Compile 和 Watcher 三者，通过 Observer 来监听自己的 model 数据变化，通过 Compile 来解析编译模板指令，最终利用 Watcher 搭起 Observer 和 Compile 之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据 model 变更的双向绑定效果。

88.你是怎么认识 vuex 的？

vuex 可以理解为一种开发模式或框架。比如 PHP 有 thinkphp，java 有 spring 等。
通过状态（数据源）集中管理驱动组件的变化（好比 spring 的 IOC 容器对 bean 进行集中管理）。
应用级的状态集中放在 store 中；改变状态的方式是提交 mutations，这是个同步的事物；异步逻辑应该封装在 action 中。

89.简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点）

详情步骤：
首先，通过 compile 编译器把 template 编译成 AST 语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用以创建编译器的。另外 compile 还负责合并 option。
然后，AST 会经过 generate（将 AST 语法树转化成 render funtion 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）
vue 的历史记录
history 记录中向前或者后退多少步
vuejs 与 angularjs 以及 react 的区别？
1.与 AngularJS 的区别
相同点：
都支持指令：内置指令和自定义指令。
都支持过滤器：内置过滤器和自定义过滤器。
都支持双向数据绑定。
都不支持低端浏览器。
不同点：
1.AngularJS 的学习成本高，比如增加了 Dependency Injection 特性，而 Vue.js 本身提供的 API 都比较简单、直观。

2.在性能上，AngularJS 依赖对数据做脏检查，所以 Watcher 越多越慢。
Vue.js 使用基于依赖追踪的观察并且使用异步队列更新。所有的数据都是独立触发的。
对于庞大的应用来说，这个优化差异还是比较明显的。
2.与 React 的区别
相同点：
React 采用特殊的 JSX 语法，Vue.js 在组件开发中也推崇编写.vue 特殊文件格式，对文件内容都有一些约定，两者都需要编译后使用。
中心思想相同：一切都是组件，组件实例之间可以嵌套。
都提供合理的钩子函数，可以让开发者定制化地去处理需求。
都不内置列数 AJAX，Route 等功能到核心包，而是以插件的方式加载。
在组件开发中都支持 mixins 的特性。
不同点：
React 依赖 Virtual DOM,而 Vue.js 使用的是 DOM 模板。React 采用的 Virtual DOM 会对渲染出来的结果做脏检查。
Vue.js 在模板中提供了指令，过滤器等，可以非常方便，快捷地操作 DOM。

90. v-show 和 v-if 指令的共同点和不同点

v-show 指令是通过修改元素的 display 的 CSS 属性让其显示或者隐藏
v-if 指令是直接销毁和重建 DOM 达到让元素显示和隐藏的效果

91.如何让 CSS 只在当前组件中起作用

将当前组件的<style>修改为<style scoped>

92.<keep-alive></keep-alive>的作用是什么？

<keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例，主要用于保留组件状态或避免重新渲染。

93.vue 中标签怎么绑定事件

绑定事件：<input @click="rdhub.cn" />

94.vue 与 react 的异同

相同点：
都支持组件化与虚拟 DOM
都支持 props 进行父子组件通信
都支持数据驱动视图，不支持操作 真实 DOM，更新状态视图自动更新
都支持服务器端渲染
不同点：
设计思想不同，vue 是 MVVM，react 是 MVC
vue 数据双向绑定，react 数据单向绑定
组件写法不同。react 推荐 jsx 写法，即把 html 与 css 都写进 js; vue 推荐 webpack+vue-loader 的单文件组件格式，即 html、css 与 js 都写进同一个文件
state 对象在 react 中是不可变的，需要使用 setState 方法更新状态；而 vue 中 state 不是必须的，数据由 data 在 vue 对象中管理
虚拟 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；react 每当应用状态改变时，全部组件都会重新渲染，所以 react 需要 shouldComponentUpdate 这个生命周期方法来进行控制

95.组件 data 为什么要用函数

vue 组件中 data 值不能为对象，因为对象是引用类型，组件可能会被多个实例同时引用。如果 data 值为对象，将导致多个实例共享一个对象，其中一个组件改变 data 属性值，其它实例也会受到影响。
只有当 data 为函数时，通过 return 返回对象的拷贝，致使每个实例都有自己独立的对象，实例之间可以互不影响的改变 data 属性值。

96.ajax 和 axios、fetch 的区别？

ajax
传统 Ajax 指的是 XMLHttpRequest（XHR）， 最早出现的发送后端请求技术，隶属于原始 js 中，核心使用 XMLHttpRequest 对象，多个请求之间如果有先后关系的话，就会出现回调地狱。
juery.ajax

```
$.ajax({
    type: 'POST',
    url: url,
    data: data,
    dataType: dataType,
```

```
    success: function () {},
    error: function () {}
});
```

JQuery ajax 是对原生 XHR 的封装，除此以外还增添了对 JSONP 的支持,举出几个缺点

本身是针对 MVC 的编程,不符合现在前端 MVVM 的浪潮

基于原生的 XHR 开发，XHR 本身的架构不清晰。

JQuery 整个项目太大，单纯使用 ajax 却要引入整个 JQuery 非常的不合理（采取个性化打包的方案又不能享受 CDN 服务）

不符合关注分离（Separation of Concerns）的原则

配置和调用方式非常混乱，而且基于事件的异步模型不友好

axios

```
axios({
      method: 'post',
      url: '/user/12345',
      data: {
            firstName: 'Fred',
            lastName: 'Flintstone'
      }
})
.then(function (response) {
      console.log(response);
})
.catch(function (error) {
      console.log(error);
});
```

axios 是一个基于 Promise 用于浏览器和 nodejs 的 HTTP 客户端，本质上也是对原生 XHR 的封装，只不过它是 Promise 的实现版本，符合最新的 ES 规范，它本身具有以下特征：

1.从浏览器中创建 XMLHttpRequest

2.支持 Promise API

3.客户端支持防止 CSRF

4.提供了一些并发请求的接口（重要，方便了很多的操作）

5.从 node.js 创建 http 请求

6.拦截请求和响应

7.转换请求和响应数据

8.取消请求

9.自动转换 JSON 数据

fetch

```
try {
   let response = await fetch(url);
   let data = response.json();
   console.log(data);
} catch(e) {
   console.log("Oops, error", e);
```

}
fetch 号称是 AJAX 的替代品，是在 ES6 出现的，使用了 ES6 中的 promise 对象。Fetch 是基于 promise 设计的。Fetch 的代码结构比起 ajax 简单多了，参数有点像 jQuery ajax。但是，一定记住 fetch 不是 ajax 的进一步封装，而是原生 js，没有使用 XMLHttpRequest 对象。
fetch 的优点：
1.符合关注分离，没有将输入、输出和用事件来跟踪的状态混杂在一个对象里
2.更好更方便的写法
3.语法简洁，更加语义化
基于标准 Promise 实现，支持 async/await
同构方便，使用 isomorphic-fetch
4.更加底层，提供的 API 丰富（request, response）
5.脱离了 XHR，是 ES 规范里新的实现方式

97.说下对 Virtual DOM 算法的理解

包括几个步骤：
1、用 JavaScript 对象结构表示 DOM 树的结构，然后用这个树构建一个真正的 DOM 树，插到文档当中；
2、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；
3、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。
Virtual DOM 本质上就是在 JS 和 DOM 之间做了一个缓存。可以类比 CPU 和硬盘，既然硬盘这么慢，我们就在它们之间加个缓存：既然 DOM 这么慢，我们就在它们 JS 和 DOM 之间加个缓存。CPU（JS）只操作内存（Virtual DOM），最后的时候再把变更写入硬盘（DOM）。

98.解释单向数据流和双向数据绑定

单向数据流：顾名思义，数据流是单向的。数据流动方向可以跟踪，流动单一，追查问题的时候可以更快捷。缺点就是写起来不太方便。要使 UI 发生变更就必须创建各种 action 来维护对应的 state。
双向数据绑定：数据之间是相通的，将数据变更的操作隐藏在框架内部。优点是在表单交互较多的场景下，会简化大量与业务无关的代码。缺点就是无法追踪局部状态的变化，增加了出错时 debug 的难度。

99.Vue 如何去除 URL 中的#

vue-router 默认使用 hash 模式，所以在路由加载的时候，项目中的 URL 会自带“#”。如果不想使用“#”，可以使用 vue-router 的另一种模式 history：new Router ({ mode : 'history', routes: [ ]})
需要注意的是，当我们启用 history 模式的时候，由于我们的项目是一个单页面应用，所以在路由跳转的时候，就会出现访问不到静态资源而出现“404”的情况，这时候就需要服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个“index.html”页面。

100. NextTick 是做什么的

$nextTick 是在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后使用 $nextTick，则可以在回调中获取更新后的 DOM。

101. Vue 组件 data 为什么必须是函数

因为 JS 本身的特性带来的，如果 data 是一个对象，那么由于对象本身属于引用类型，当我们修改其中的一个属性时，会影响到所有 Vue 实例的数据。如果将 data 作为一个函数返回一个对象，那么每一个实例的 data 属性都是独立的，不会相互影响了。

102. 计算属性 computed 和事件 methods 有什么区别

我们可以将同一函数定义为一个 method 或者一个计算属性。对于最终的结果，两种方式是相同的。
不同点：
computed：计算属性是基于它们的依赖进行缓存的，只有在它的相关依赖发生改变时才会重新求值。
method：只要发生重新渲染， method 调用总会执行该函数。

103. 对比 jQuery ，Vue 有什么不同

jQuery 专注视图层，通过操作 DOM 去实现页面的一些逻辑渲染；Vue 专注于数据层，通过数据的双向绑定，最终表现在 DOM 层面，减少了 DOM 操作。
Vue 使用了组件化思想，使得项目子集职责清晰，提高了开发效率，方便重复利用，便于协同开发。

104. Vue 中怎么自定义指令

全局注册
// 注册一个全局自定义指令 v-focus Vue .directive('focus' , { // 当被绑定的元素插入到 DOM 中时…… inserted: function (el) { // 聚焦元素 el.focus() } })
局部注册
directives: { focus: { // 指令的定义 inserted: function (el) { el.focus() } } }

105. Vue 中怎么自定义过滤器

可以用全局方法 Vue.filter() 注册一个自定义过滤器，它接收两个参数：过滤器 ID 和过滤器函数。过滤器函数以值为参数，返回转换后的值。
Vue .filter('reverse' , function (value) { return value.split('').reverse().join('') })

过滤器也同样接受全局注册和局部注册。

106. 对 keep-alive 的了解

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染。
<keep-alive><component><!-- 该组件将被缓存！--></component></keep-alive>
可以使用 API 提供的 props，实现组件的动态缓存。

107.