1. 请简述一下MVVM框架是什么？它和Jquery的区别是什么？

MVVM是Model-View-ViewModel的简写。

MVVM 是 Web 前端一种非常流行的开发模式，利用 MVVM 可以使我们的代码更专注于处理业务逻辑而不是像Jquery去关心 DOM 操作。目前著名的 MVVM 框架有 vue, angular 等，这些框架各有千秋，但是实现的思想大致上是相同的：数据绑定 + 视图刷新。跟MVC一样，主要目的是分离视图（View）和模型（Model）。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。 在MVVM中，数据是核心，由于VIewModel与View之间的双向绑定，操作了ViewModel中的数据（当然只能是监控属性），就会同步到DOM，我们透过DOM事件监控用户对DOM的改动，也会同步到ViewModel。

2. 你了解前端路由是如何实现的吗?

前端路由本质就是监听 URL 的变化，然后匹配路由规则，显示相应的页面，并且无须刷新。目前单页面使用的路由就只有两种实现方式。

- hash 模式

  hash路由一个明显的标志是带有#,我们主要是通过监听url中的hash变化来进行路由跳转。

  hash的优势就是兼容性更好,在老版IE中都有运行,问题在于url中一直存在#不够美观,而且hash路由更像是Hack而非标准。实现代码如下：

```
class Routers {
    constructor() {
        this.routes = {};
        this.currentUrl = '';
        this.refresh = this.refresh.bind(this);
        window.addEventListener('load', this.refresh, false);
        window.addEventListener('hashchange', this.refresh, false);
    }
    route(path, callback) {
        this.routes[path] = callback || function() {};
    }
    refresh() {
        this.currentUrl = location.hash.slice(1) || '/';
        this.routes[this.currentUrl]();
    }
}
```

- history 模式

  History API的出现使得我们的路由更加便捷美观。我们可以直接使用History API的方法和属性。代码实现如下：

```
class Routers {
  constructor() {
    this.routes = {};
    this._bindPopState();
  }
  init(path) {
    history.replaceState({path: path}, null, path);
    this.routes[path] && this.routes[path]();
  }

  route(path, callback) {
    this.routes[path] = callback || function() {};
  }

  go(path) {
    history.pushState({path: path}, null, path);
    this.routes[path] && this.routes[path]();
  }
  _bindPopState() {
    window.addEventListener('popstate', e => {
      const path = e.state && e.state.path;
      this.routes[path] && this.routes[path]();
    });
  }
}
```

3. 什么是虚拟dom，解决了什么问题?

在浏览器渲染网页的过程中，加载到HTML文档后，会将文档解析并构建DOM树，然后将其与解析CSS生成的CSSOM树一起结合产生爱的结晶——RenderObject树，然后将RenderObject树渲染成页面（当然中间可能会有一些优化，比如RenderLayer树）。这些过程都存在与渲染引擎之中，渲染引擎在浏览器中是于JavaScript引擎（JavaScriptCore也好V8也好）分离开的，但为了方便JS操作DOM结构，渲染引擎会暴露一些接口供JavaScript调用。由于这两块相互分离，通信是需要付出代价的，因此JavaScript调用DOM提供的接口性能不咋地。各种性能优化的最佳实践也都在尽可能的减少DOM操作次数。

而虚拟DOM干了什么？它直接用JavaScript实现了DOM树（大致上）。组件的HTML结构并不会直接生成DOM，而是映射生成虚拟的JavaScript DOM结构，又通过在这个虚拟DOM上实现了一个 diff 算法找出最小变更，再把这些变更写入实际的DOM中。这个虚拟DOM以JS结构的形式存在，计算性能会比较好，而且由于减少了实际DOM操作次数，性能会有较大提升。

4. 请详细说下你对vue生命周期的理解?

总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后： 在beforeCreate阶段，vue实例的挂载元素el还没有。

载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

更新前/后：当data变化时，会触发beforeUpdate和updated方法。

销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

5.聊聊你对Vue.js的template编译的理解?

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）。

6. 描述一下React 生命周期

整个 React 生命周期有3个阶段：创建、更新、卸载。

- 第一阶段

  这是虚拟 DOM 创建的阶段，会依次执行 5 个方法，这 5 个方法中除了 render 方法，其余四个方法在整个生命周期中只调用 1 次，而且一定会调用 1 次：

  `getDefaultProps()`

  这个方法在组件实例创建前，也就是构造函数执行前执行，获取父组件传来的参数，你可以在这里编辑参数并返回新的参数作为 props

  `getInitalState()`

  组件创建的一开始会调用这个方法初始化组件的 state

  `componentWillMount()`

  在组件 render 之前执行该方法，可用来修改 state。React 先调用父组件的这个函数，再调用子组件的这个函数

  `render()`

  开始组件渲染函数，返回一个只有一个根节点的虚拟 DOM。该函数中不能同步的修改组件的状态(state)。

  `componentDidMount()`

  在 render 渲染之后，通知组件已经加载完成。React 先调用子组件的这个函数，再调用父组件的这个函数。从这个函数开始，该组件就可以和其他框架交互了。比如设置计时器或者发起网络请求。

- 第二阶段

  此时该组件已经进入了稳定运行阶段，这个阶段组件可以处理用户交互，或者接收事件更新界面。以下方法在整个生命周期中可以执行很多次，也可以一次也不执行。

  `componentWillReceiveProps()`

  当父容器中对应的参数改变将会调用子组件的该函数。新的 props 将会作为参数传递进来，老的 props 可以根据 this.props 来获取。我们可以在该函数中对state作一些处理。并且在该函数中更新 state 不会发生二次渲染

  `shouldComponentUpdate()`

该函数传递过来两个参数，新的 state 和新的 props。state 和 props 的改变都会调到该函数。该函数主要对传递过来的 nextProps 和 nextState 作判断。如果返回 true 则重新渲染(默认都是返回 true)，如果返回 false 则不重新渲染。在某些特定条件下，我们可以根据传递过来的 props 和 state 来选择更新或者不更新，从而提高效率。

`componentWillUpdate()`

与 componentWillMount 方法类似，在 render 渲染之前被调用。组件上会接收到新的 props 或者 state。这个函数调用之后，就会把 nextProps 和 nextState 分别设置到 this.props 和 this.state 中。

`componentDidUpdate()`

与 componentDidMount 方法类似，在 render 渲染之后被调用，真实 DOM 生成之后调用该函数。传递过来的参数是之前的 props 和 state。

- 第三阶段

这就是消亡的阶段，主要进行内存的清理和释放的工作。这个阶段只有一个方法，该方法在整个生命周期内调用且仅调用一次。

`componentWillUnmount()`

当组件要被从界面上移除的时候，就会调用 componentWillUnmount。在这里进行一些相关的销毁操作，比如撤销定时器，事件监听等等。

7. 当组件的setState函数被调用之后，发生了什么?

React会做的第一件事就是把你传递给setState的参数对象合并到组件原先的state。这个事件会导致一个“reconciliation”（调和）的过程。reconciliation的最终目标就是，尽可能以最高效的方法，去基于新的state来更新UI。为了达到这个目的，React会构建一个React元素树（你可以把这个想象成一个表示UI的一个对象）。一旦这个树构建完毕，React为了根据新的state去决定UI要怎么进行改变，它会找出这棵新树和旧树的不同之处。React能够相对精确地找出哪些位置发生了改变以及如何发生了什么变化，并且知道如何只通过必要的更新来最小化重渲染。