2022前端面试题

1.浏览器渲染机制、重绘、重排

网页生成过程：

- `HTML` 被HTML解析器解析成 `DOM` 树
- `css` 则被css解析器解析成 `CSSOM` 树
- 结合 `DOM` 树和 `CSSOM` 树，生成一棵渲染树
- 生成布局，即将所有渲染树的所有节点进行平面合成
- 将布局绘制在屏幕上

重排(也称回流): 当 `DOM` 的变化影响了元素的几何信息(`DOM` 对象的位置和尺寸大小)，浏览器需要重新计算元素的几何属性，将其安放在界面中的正确位置，这个过程叫做重排。 触发：

1. 添加或者删除可见的DOM元素
2. 元素尺寸改变——边距、填充、边框、宽度和高度

重绘： 当一个元素的外观发生改变，但没有改变布局,重新把元素外观绘制出来的过程，叫做重绘。 触发：

- 改变元素的 `color、background、box-shadow` 等属性

重排优化建议：

1. 样式集中修改
2. 缓存需要修改的 `DOM` 元素
3. 尽量只修改 `position：absolute` 或 `fixed` 元素，对其他元素影响不大
4. 动画开启 `GPU` 加速， `translate` 使用 `3D` 变化

`transform` 不重绘，不回流 是因为 `transform` 属于合成属性，对合成属性进行 `transition/animate` 动画时，将会创建一个合成层。这使得动画元素在一个独立的层中进行渲染。当元素的内容没有发生改变，就没有必要进行重绘。浏览器会通过重新复合来创建动画帧。

2.什么是BFC?

`BFC` 即块格式化上下文。 `BFC` 是CSS布局的一个概念，是一个环境，里面的元素不会影响外面的元素。 1.分属于不同的 `BFC` 时,可以防止 `margin` 重叠 2.清除内部浮动 3.自适应多栏布局

3.js数据类型、typeof、instanceof、类型转换

1. `string、number、boolean、null、undefined、object(function、array)、symbol(ES10 BigInt)`
2. `typeof` 主要用来判断数据类型 返回值有 `string、boolean、number、function、object、undefined。`
3. `instanceof` 判断该对象是谁的实例。
4. `null` 表示空对象 `undefined` 表示已在作用域中声明但未赋值的变量

4.闭包(高频)

- 闭包就是在函数里面声明函数，使得子函数可以访问父函数中所有的局部变量;
- 保护变量不受外界污染，使其一直存在内存中不被释放，因为闭包太消耗内存，过多的闭包可能会导致内存泄漏;

>>apply, call和bind区别

三个函数的作用：都是将函数绑定到上下文中，用来改变函数中this的指向；三者的不同点在于语法的不同。

>>判断数据类型的方法

typeof

- 优点：能够快速区分基本数据类型
- 缺点：不能将Object、Array和Null区分，都返回object

instanceof

- 能够区分Array、Object和Function，适合用于判断自定义的类实例对象
- 缺点：Number，Boolean，String基本数据类型不能判断

Object.prototype.toString.call()

- 优点：精准判断数据类型
- 缺点：写法繁琐不容易记，推荐进行封装后使用

5.原型、原型链(高频)

原型: 对象中固有的 `__proto__` 属性，该属性指向对象的 `prototype` 原型属性。

原型链: 当我们访问一个对象的属性时，如果这个对象内部不存在这个属性，那么它就会去它的原型对象里找这个属性，这个原型对象又会有自己的原型，于是就这样一直找下去，也就是原型链的概念。原型链的尽头一般来说都是 `Object.prototype` 所以这就是我们新建的对象为什么能够使用 `toString()` 等方法的原因。

特点: `JavaScript` 对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变。

6.this指向

`this` 对象是是执行上下文中的一个属性，它指向最后一次调用这个方法的对象，在全局函数中， `this` 等于 `window` ，而当函数被作为某个对象调用时，this等于那个对象。 在实际开发中， `this` 的指向可以通过四种调用模式来判断。

1. 函数调用，当一个函数不是一个对象的属性时，直接作为函数来调用时， `this` 指向全局对象。
2. 方法调用，如果一个函数作为一个对象的方法来调用时， `this` 指向这个对象。
3. 构造函数调用， `this` 指向这个用 `new` 新创建的对象。
4. 第四种是 `apply` 、 `call` 和 `bind` 调用模式，这三个方法都可以显示的指定调用函数的 this 指向。 `apply` 接收参数的是数组， `call` 接受参数列表，`` bind 方法通过传入一个对象，返回一个 this 绑定了传入对象的新函数。这个函数的 this 指向除了使用 new `时会被改变，其他情况下都不会改变。

7.箭头函数与普通函数的区别

- 箭头函数是匿名函数，不能作为构造函数，不能使用new
- 箭头函数不绑定 `this` ，会捕获其所在的上下文的this值，作为自己的this值
- `call` ， `apply` ， `bind` 无法改变箭头函数中this的指向
- 箭头函数没有原型属性

8.取消ajax请求

原生js：xhr.abort()

axios：axios.CancelToken()

9.EventLoop

JS 是单线程的，为了防止一个函数执行时间过长阻塞后面的代码，所以会先将同步代码压入执行栈中，依次执行，将异步代码推入异步队列，异步队列又分为宏任务队列和微任务队列，因为宏任务队列的执行时间较长，所以微任务队列要优先于宏任务队列。微任务队列的代表就是， Promise.then ， MutationObserver ，宏任务的话就是 setImmediate setTimeout setInterval

10.原生ajax

ajax是一种异步通信的方法,从服务端获取数据,达到局部刷新页面的效果。 过程：

1. 创建 XMLHttpRequest 对象;
2. 调用 open 方法传入三个参数 请求方式 (GET/POST)、url、同步异步(true/false)；
3. 监听 onreadystatechange 事件，当 readystate 等于4时返回 responseText；
4. 调用send方法传递参数。

11.ES6

1. 新增symbol类型 表示独一无二的值，用来定义独一无二的对象属性名;
2. const/let 都是用来声明变量,不可重复声明，具有块级作用域。存在暂时性死区，也就是不存在变量提升。(const一般用于声明常量);
3. 变量的解构赋值(包含数组、对象、字符串、数字及布尔值,函数参数),剩余运算符(...rest);
4. 模板字符串(${data});
5. 扩展运算符(数组、对象);;
6. 箭头函数;
7. Set和Map数据结构;
8. Proxy/Reflect;
9. Promise;
10. async函数;
11. Class;
12. Module语法(import/export)。

12.简述MVVM

MVVM是 `Model-View-ViewModel` 缩写，也就是把 `MVC` 中的 `Controller` 演变成 `ViewModel`。`Model` 层代表数据模型， `View` 代表UI组件， `ViewModel` 是 `View` 和 `Model` 层的桥梁，数据会绑定到 `viewModel` 层并自动将数据渲染到页面中，视图变化的时候会通知 `viewModel` 层更新数据。

13.谈谈对Vue生命周期的理解?

每个 `Vue` 实例在创建时都会经过一系列的初始化过程， `vue` 的生命周期钩子，就是说在达到某一阶段或条件时去触发的函数，目的就是为了完成一些动作或者事件

- `beforeCreate` 组件实例被创建之初，组件的属性生效之前
- `created` 组件实例已经完全创建，属性也绑定，但真实 dom 还没有生成，$el 还不可用
- `beforeMount` 在挂载开始之前被调用：相关的 render 函数首次被调用
- `mounted` el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子
- `beforeUpdate` 组件数据更新之前调用，发生在虚拟 DOM 打补丁之前
- `update` 组件数据更新之后
- `activited` keep-alive 专属，组件被激活时调用
- `deactivated` keep-alive 专属，组件被销毁时调用
- `beforeDestory` 组件销毁前调用
- `destoryed` 组件销毁后调用

>>Vue 的父组件和子组件生命周期钩子函数执行顺序?

Vue 的父组件和子组件生命周期钩子函数执行顺序可以归类为以下 4 部分：

- 加载渲染过程

  父 beforeCreate -> 父 created -> 父 beforeMount -> 子 beforeCreate -> 子 created -> 子 beforeMount -> 子 mounted -> 父 mounted

- 子组件更新过程

  父 beforeUpdate -> 子 beforeUpdate -> 子 updated -> 父 updated

- 父组件更新过程

  父 beforeUpdate -> 父 updated

- 销毁过程

  父 beforeDestroy -> 子 beforeDestroy -> 子 destroyed -> 父 destroyed

>>父组件可以监听到子组件的生命周期吗?

比如有父组件 Parent 和子组件 Child，如果父组件监听到子组件挂载 mounted 就做一些逻辑处理，可以通过以下写法实现：

typescript 复制代码

```
// Parent.vue
<Child @mounted="doSomething"/>

// Child.vue
mounted() {
  this.$emit("mounted");
}
```

复制代码

以上需要手动通过 $emit 触发父组件的事件，更简单的方式可以在父组件引用子组件时通过 @hook 来监听即可，如下所示：

typescript 复制代码

```
//  Parent.vue
<Child @hook:mounted="doSomething" ></Child>

doSomething() {
   console.log('父组件监听到 mounted 钩子函数 ...');
},

//  Child.vue
mounted(){
   console.log('子组件触发 mounted 钩子函数 ...');
},

// 以上输出顺序为:
```

```
// 子组件触发 mounted 钩子函数 ...
// 父组件监听到 mounted 钩子函数 ...
复制代码
```

当然 @hook 方法不仅仅是可以监听 mounted，其它的生命周期事件，例如：created，updated 等都可以监听。

>>在哪个生命周期内调用异步请求?

可以在钩子函数 created、beforeMount、mounted 中进行调用，因为在这三个钩子函数中，data 已经创建，可以将服务端端返回的数据进行赋值。但是本人推荐在 created 钩子函数中调用异步请求，因为在 created 钩子函数中调用异步请求有以下优点：

- 能更快获取到服务端数据，减少页面 loading 时间；
- ssr 不支持 beforeMount 、mounted 钩子函数，所以放在 created 中有助于一致性；

14.Computed与Watch

- computed的属性是具备缓存的，依赖的值不发生变化，对其取值时计算属性方法不会重复执行。除非依赖的响应式属性变化时才会重新计算，主要当做属性来使用 `computed` 中的函数必须用 `return` 返回最终的结果
- watch是监控值的变化，当值发生改变的时候，会调用回调函数

使用场景 `computed` ：当一个属性受多个属性影响的时候使用，例：购物车商品结算功能
`watch` ：当一条数据影响多条数据的时候使用，例：搜索数据

15.v-for中key的作用

1. `key` 的作用主要是为了更高效的对比虚拟DOM中每个节点是否是相同节点;
2. 触发过渡

>>scoped样式穿透

1. 使用/deep/
2. 使用两个style标签

>>Vue 组件间通信有哪几种方式?

Vue 组件间通信是面试常考的知识点之一，这题有点类似于开放题，你回答出越多方法当然越加分，表明你对 Vue 掌握的越熟练。Vue 组件间通信只要指以下 3 类通信：父子组件通信、隔代组件通信、兄弟组件通信，下面我们分别介绍每种通信方式且会说明此种方法可适用于哪类组件间通信。

(1) `props / $emit` 适用 父子组件通信

这种方法是 Vue 组件的基础，相信大部分同学耳闻能详，所以此处就不举例展开介绍。

(2) `ref` 与 `$parent / $children` 适用 父子组件通信

- `ref`：如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例
- `$parent` / `$children`：访问父 / 子实例

(3) `EventBus （$emit / $on）` 适用于 父子、隔代、兄弟组件通信

这种方法通过一个空的 Vue 实例作为中央事件总线（事件中心），用它来触发事件和监听事件，从而实现任何组件间的通信，包括父子、隔代、兄弟组件。

(4) `$attrs` / `$listeners` 适用于 隔代组件通信

- `$attrs`：包含了父作用域中不被 prop 所识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 `v-bind="$attrs"` 传入内部组件。通常配合 inheritAttrs 选项一起使用。
- `$listeners`：包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 `v-on="$listeners"` 传入内部组件

(5) `provide / inject` 适用于 隔代组件通信

祖先组件中通过 provider 来提供变量，然后在子孙组件中通过 inject 来注入变量。 provide / inject API 主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。

(6) Vuex 适用于 父子、隔代、兄弟组件通信

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。

- Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。
- 改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化。

>>你使用过 Vuex 吗?

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。

（1）Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。

（2）改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化。

主要包括以下几个模块：

- State：定义了应用状态的数据结构，可以在这里设置默认的初始状态。
- Getter：允许组件从 Store 中获取数据，mapGetters 辅助函数仅仅是将 store 中的 getter 映射到局部计算属性。
- Mutation：是唯一更改 store 中状态的方法，且必须是同步函数。
- Action：用于提交 mutation，而不是直接变更状态，可以包含任意异步操作。
- Module：允许将单一的 Store 拆分为多个 store 且同时保存在单一的状态树中。

17.常用指令

- `v-if`： 判断是否隐藏；
- `v-for`： 数据循环出来；
- `v-bind:class`： 绑定一个属性；
- `v-model`： 实现双向绑定

18.双向绑定实现原理

当一个**Vue**实例创建时，Vue会遍历data选项的属性，用 **Object.defineProperty** 将它们转为 `getter/setter` 并且在内部追踪相关依赖，在属性被访问和修改时通知变化。每个组件实例都有相应的 `watcher` 程序实例，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 `setter` 被调用时，会通知 `watcher` 重新计算，从而致使它关联的组件得以更新。

19.Vue中$set的用法

向响应式对象中添加一个属性，并确保这个新属性同样是响应式的，且触发视图更新。它必须用于向响应式对象上添加新属性

kotlin 复制代码

```
数组：this.$set(Array, index, newValue);
对象：this.$set(Object, key, value);
```

20.Vue组件的API主要包含三部分：prop、event、slot

`props` 表示组件接收的参数，最好用对象的写法，这样可以针对每个属性设置类型、默认值 或自定义校验属性的值，此外还可以通过 `type` 、 `validator` 等方式对输入进行验证

`slot` 可以给组件动态插入一些内容或组件，是实现高阶组件的重要途径； 当需要多个插槽时，可以使用具名 `slot`

`event` 是子组件向父组件传递消息的重要途径

21.谈谈对组件的理解

- 组件化开发能大幅提高应用开发效率、测试性、复用性
- 常用的组件化技术：属性、自定义事件、插槽
- 降低更新范围，只重新渲染变化的组件
- 高内聚、低耦合、单向数据流

22.异步组件原理

先渲染异步占位符节点 -> 组件加载完毕后调用 `forceUpdate` 强制更新。

23.Vue中如何检测数组的变化?

`set` , `del`

`push` , `splice` , `shift`

Vue 框架是通过**遍历数组** 和**递归遍历对象**，从而达到利用 `Object.defineProperty()` 也能对对象和数组（部分方法的操作）进行监听。

24.Vue的组件data为什么必须是一个函数?

组件的 `data` 必须是一个函数，是为了防止两个组件的数据产生污染。 如果都是对象的话，会在合并的时候，指向同一个地址。 而如果是函数的时候，合并的时候调用，会产生两个空间。

25.请说明nextTick的原理。

`nextTick` 是一个微任务。

- `nextTick` 中的回调是在下次Dom更新循环结束之后执行的延迟回调
- 可以用于获取更新后的Dom
- Vue中的数据更新是异步的，使用 `nextTick` 可以保证用户定义的逻辑在更新之后执行

26.谈谈你对 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，避免重新渲染 ，其有以下特性：

- 一般结合路由和动态组件一起使用，用于缓存组件；
- 提供 include 和 exclude 属性，两者都支持字符串或正则表达式， include 表示只有名称匹配的组件会被缓存，exclude 表示任何名称匹配的组件都不会被缓存 ，其中 exclude 的优先级比 include 高；
- 对应两个钩子函数 activated 和 deactivated ，当组件被激活时，触发钩子函数 activated，当组件被移除时，触发钩子函数 deactivated。

27.Vuex、vue-router实现原理

`vuex` 是一个专门为vue.js应用程序开发的状态管理库。 核心概念：

- `state` (单一状态树) `getter/Mutation` 显示提交更改 `state`
- `Action类似Mutation` ， 提交 `Mutation` ， 可以包含任意异步操作。
- `module` (当应用变得庞大复杂，拆分 `store` 为具体的 `module` 模块)

28.Vue-router有几种钩子函数？执行流程如何?

全局守卫

- `beforeEach` 全局前置守卫，在路由跳转前触发，它在 **每次导航** 时都会触发。

- `beforeResolve` 全局解析守卫，在路由跳转前，所有 **组件内守卫** 和 **异步路由组件** 被解析之后触发，它同样在 **每次导航** 时都会触发。
- `afterEach` 全局后置钩子，它发生在路由跳转完成后， `beforeEach` 和 `beforeResolve` 之后， `beforeRouteEnter` （组件内守卫）之前。它同样在 **每次导航** 时都会触发。

路由守卫

- `beforeEnter` 需要在路由配置上定义 `beforeEnter` 守卫，此守卫只在进入路由时触发，在 `beforeEach` 之后紧随执行，不会在 `params` 、 `query` 或 `hash` 改变时触发。

组件守卫

- `beforeRouteEnter` 路由进入组件之前调用，该钩子在全局守卫 `beforeEach` 和路由守卫 `beforeEnter` 之后，全局 `beforeResolve` 和全局 `afterEach` 之前调用。该守卫内访问不到组件的实例，也就是 `this` 为 `undefined` ，也就是他在 `beforeCreate` 生命周期前触发。
- `beforeRouteUpdate`
- `beforeRouteLeave`

29.Vue中使用了哪些设计模式?

- 工场模式：传入参数就可以创建实例（虚拟节点的创建）
- 发布订阅模式：eventBus
- 观察者模式：watch和dep
- 代理模式：_data属性、proxy、防抖、节流
- 中介者模式：vuex

30.你都做过哪些Vue的性能优化?

kotlin 复制代码

```
编码阶段
尽量减少data中的数据，data中的数据都会增加getter和setter，会收集对应的watcher
v-if和v-for不能连用
如果需要使用v-for给每项元素绑定事件时使用事件代理
SPA 页面采用keep-alive缓存组件
在更多的情况下，使用v-if替代v-show
key保证唯一
使用路由懒加载、异步组件
防抖、节流
第三方模块按需导入
长列表滚动到可视区域动态加载
图片懒加载
```

SEO优化
预渲染
服务端渲染SSR
打包优化
压缩代码
使用cdn加载第三方模块
splitChunks抽离公共文件
骨架屏
还可以使用缓存(客户端缓存、服务端缓存)优化、服务端开启gzip压缩等。

31.你知道Vue3有哪些新特性吗？它们会带来什么影响?

- **性能提升**

更小巧、更快速 支持自定义渲染器 支持摇树优化：一种在打包时去除无用代码的优化手段 支持Fragments和跨组件渲染

- **API变动**

模板语法99%保持不变 原生支持基于class的组件，并且无需借助任何编译及各种stage阶段的特性 在设计时也考虑TypeScript的类型推断特性 重写虚拟DOM 可以期待更多的编译时提示来减少运行时的开销 优化插槽生成 可以单独渲染父组件和子组件 静态树提升 降低渲染成本 基于Proxy的观察者机制 节省内存开销

- **不兼容IE11**

检测机制 更加全面、精准、高效,更具可调试式的响应跟踪

32.实现双向绑定 Proxy 与 Object.defineProperty 相比优劣如何?

1. **Object.definedProperty**的作用是劫持一个对象的属性，劫持属性的getter和setter方法，在对象的属性发生变化时进行特定的操作。而 Proxy劫持的是整个对象。
2. **Proxy**会返回一个代理对象，我们只需要操作新对象即可，而Object.defineProperty只能遍历对象属性直接修改。
3. **Object.definedProperty**不支持数组，更准确的说是不支持数组的各种API，因为如果仅仅考虑arry[i] = value 这种情况，是可以劫持 的，但是这种劫持意义不大。而Proxy可以支持数组的各种API。
4. 尽管Object.defineProperty有诸多缺陷，但是其兼容性要好于Proxy。

33.浏览器从输入url到渲染页面，发生了什么?

构建请求

查找强缓存

DNS解析

建立TCP连接(三次握手)

发送HTTP请求(网络请求后网络响应)

解析html构建DOM树

解析css构建CSS树、样式计算

生成布局树(Layout Tree)

建立图层树(Layer Tree)

生成绘制列表

生成图块并栅格化

显示器显示内容

最后断开连接：TCP 四次挥手 (浏览器会将各层的信息发送给GPU,GPU会将各层合成,显示在屏幕上)

34.Http和Https区别（高频）

perl 复制代码

```
1.`HTTP` 的URL 以http:// 开头，而HTTPS 的URL 以https:// 开头
2.`HTTP` 是不安全的，而 HTTPS 是安全的
3.`HTTP` 标准端口是80 ，而 HTTPS 的标准端口是443
4.`在OSI` 网络模型中，HTTP工作于应用层，而HTTPS 的安全传输机制工作在传输层
5.`HTTP` 无法加密，而HTTPS 对传输的数据进行加密
6.`HTTP`无需证书，而HTTPS 需要CA机构wosign的颁发的SSL证书
```

35.GET和POST区别（高频）

vbscript 复制代码

```
1.GET在浏览器回退不会再次请求，POST会再次提交请求
2.GET请求会被浏览器主动缓存，POST不会，要手动设置
3.GET请求参数会被完整保留在浏览器历史记录里，POST中的参数不会
```

4.GET请求在URL中传送的参数是有长度限制的，而POST没有限制
5.GET参数通过URL传递，POST放在Request body中
6.GET参数暴露在地址栏不安全，POST放在报文内部更安全
7.GET一般用于查询信息，POST一般用于提交某种信息进行某些修改操作
8.GET产生一个TCP数据包；POST产生两个TCP数据包

36.理解xss，csrf，ddos攻击原理以及避免方式

XSS（Cross-Site Scripting，**跨站脚本攻击**)是一种代码注入攻击。攻击者在目标网站上注入恶意代码，当被攻击者登陆网站时就会执行这些恶意代码，这些脚本可以读取 cookie，session tokens，或者其它敏感的网站信息，对用户进行钓鱼欺诈，甚至发起蠕虫攻击等。

CSRF （Cross-site request forgery）**跨站请求伪造**：攻击者诱导受害者进入第三方网站，在第三方网站中，向被攻击网站发送跨站请求。利用受害者在被攻击网站已经获取的注册凭证，绕过后台的用户验证，达到冒充用户对被攻击的网站执行某项操作的目的。

DDoS 又叫**分布式拒绝服务**，全称 Distributed Denial of Service，其原理就是利用大量的请求造成资源过载，导致服务不可用。

XSS避免方式：

1. url 参数使用 encodeURIComponent 方法转义
2. 尽量不是有 InnerHtml 插入 HTML 内容
3. 使用特殊符号、标签转义符。

CSRF**避免方式：**

1. 添加验证码
2. 使用token
    - 服务端给用户生成一个token，加密后传递给用户
    - 用户在提交请求时，需要携带这个token
    - 服务端验证token是否正确

DDos避免方式：**

1. 限制单IP请求频率。
2. 防火墙等防护设置禁止 ICMP 包等
3. 检查特权端口的开放

37.前端性能优化的几种方式

markdown 复制代码

1. 浏览器缓存
2. 防抖、节流
3. 资源懒加载、预加载
4. 开启Nginx gzip压缩
5. 文件采用按需加载等等
6. 图片优化，采用svg图片或者字体图标

38.什么是同源策略

一个域下的js脚本未经允许的情况下，不能访问另一个域下的内容。通常判断跨域的依据是协议、域名、端口号是否相同，不同则跨域。同源策略是对js脚本的一种限制，并不是对浏览器的限制，像img,script脚本请求不会有跨域限制。

39.跨域通信的几种方式

解决方案：

1. `jsonp` (利用 `script` 标签没有跨域限制的漏洞实现。缺点：只支持 `GET` 请求)
2. `CORS` (设置 `Access-Control-Allow-Origin` ： 指定可访问资源的域名)
3. `Websocket` 是HTML5的一个持久化的协议，它实现了浏览器与服务器的全双工通信，同时也是跨域的一种解决方案。
4. `Node` 中间件代理
5. `Nginx` 反向代理
6. 各种嵌套 `iframe` 的方式，不常用。
7. 日常工作中用的最对的跨域方案是CORS和Nginx反向代理

40.能不能说一说浏览器的本地存储？各自优劣如何？

不同点：

1. `cookie` 数据始终在同源的 `http` 请求中携带（即使不需要），即 `cookie` 在浏览器和服务器间来回传递。 `cookie` 数据还有路径（ `path` ）的概念，可以限制 `cookie` 只属于某个路径下 `sessionStorage` 和 `localStorage` 不会自动把数据发送给服务器，仅在本地保存。
2. 存储大小限制也不同，

- `cookie` 数据不能超过4K， `sessionStorage和localStorage` 可以达到5M
- `sessionStorage` ： 仅在当前浏览器窗口关闭之前有效；

- `localStorage`：始终有效，窗口或浏览器关闭也一直保存，本地存储，因此用作持久数据；
- `cookie`： 只在设置的 `cookie` 过期时间之前有效，即使窗口关闭或浏览器关闭

1. 作用域不同

- `sessionStorage`： 不在不同的浏览器窗口中共享，即使是同一个页面；
- `cookie`：也是在所有同源窗口中都是共享的.也就是说只要浏览器不关闭，数据仍然存在

41.Promise是什么

`promise` 是异步编程的一种解决方案，是为解决异步操作函数里的嵌套回调。

`promise` 有三种状态： **`pending` (等待态)**， **`resolved` (成功态)**， **`rejected` (失败态)**；

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）。

有 `resolve`、 `reject`、 `then`、 `catch`、 `all`、 `race` 这些方法。

42.谈谈Promise.all方法，Promise.race方法以及使用

- Promise.all可以将多个Promise实例包装成一个新的Promise实例。同时，成功和失败的返回值是不同的，成功的时候返回的是一个结果数组，而失败的时候则返回最先被reject失败状态的值。（Promise.all 方法的参数不一定是数组，但是必须具有 iterator 接口，且返回的每个成员都是 Promise 实例。）

- 顾名思义，Promse.race就是赛跑的意思，意思就是说，Promise.race([p1, p2, p3])里面哪个结果获得的快，就返回那个结果，不管结果本身是成功状态还是失败状态。

43.Webpack构建流程简单说一下

- 初始化：启动构建，读取与合并配置参数，加载 Plugin，实例化 Compiler

- 编译：从 Entry 出发，针对每个 Module 串行调用对应的 Loader 去翻译文件的内容，再找到该 Module 依赖的 Module，递归地进行编译处理

- 输出：将编译后的 Module 组合成 Chunk，将 Chunk 转换成文件，输出到文件系统中

>>js 获取对象的长度

ini 复制代码

```
1. for in
2. var arr = Object.keys;
   arr.length
```

>>Set 中的特殊值

`Set` 对象存储的值总是唯一的，所以需要判断两个值是否恒等。有几个特殊值需要特殊对待：

- +0 与 -0 在存储判断唯一性的时候是恒等的，所以不重复
- `undefined` 与 `undefined` 是恒等的，所以不重复
- `NaN` 与 `NaN` 是不恒等的，但是在 `Set` 中认为 `NaN` 与 `NaN` 相等，所有只能存在一个，不重复。

Set 的属性:

- `size`：返回集合所包含元素的数量（不包含重复值）

Set 实例对象的方法

- `add(value)`：添加某个值，返回 `Set` 结构本身(可以链式调用)。
- `delete(value)`：删除某个值，删除成功返回 `true`，否则返回 `false`。
- `has(value)`：返回一个布尔值，表示该值是否为 `Set` 的成员。
- `clear()`：清除所有成员，没有返回值。

遍历方法

- `keys()`：返回键名的遍历器。
- `values()`：返回键值的遍历器。
- `entries()`：返回键值对的遍历器。
- `forEach()`：使用回调函数遍历每个成员。

`WeakSet` 结构与 `Set` 类似，也是不重复的值的集合。

- 成员都是数组和类似数组的对象，若调用 `add()` 方法时传入了非数组和类似数组的对象的参数，就会抛出错误。

const b = [1, 2, [1, 2]] new WeakSet(b) // Uncaught TypeError: Invalid value used in weak set 复制代码

- 成员都是弱引用，可以被垃圾回收机制回收，可以用来保存 DOM 节点，不容易造成内存泄漏。

- `WeakSet` 不可迭代，因此不能被用在 `for-of` 等循环中。

- `WeakSet` 没有 `size` 属性。

>>Map 对象的方法

- `set(key, val)`：向 `Map` 中添加新元素
- `get(key)`：通过键值查找特定的数值并返回
- `has(key)`：判断 `Map` 对象中是否有 `Key` 所对应的值，有返回 `true`，否则返回 `false`
- `delete(key)`：通过键值从 `Map` 中移除对应的数据
- `clear()`：将这个 `Map` 中的所有元素删除

Map 的属性

- size: 返回集合所包含元素的数量

遍历方法

- `keys()`：返回键名的遍历器
- `values()`：返回键值的遍历器
- `entries()`：返回键值对的遍历器
- `forEach()`：使用回调函数遍历每个成员

WeakMap

`WeakMap` 结构与 `Map` 结构类似，也是用于生成键值对的集合。

- 只接受对象作为键名（`null` 除外），不接受其他类型的值作为键名
- 键名是弱引用，键值可以是任意的，键名所指向的对象可以被垃圾回收，此时键名是无效的
- 不能遍历，方法有 `get`、`set`、`has`、`delete`

>>Set

- 是一种叫做集合的数据结构(ES6新增的)
- 成员唯一、无序且不重复
- `[value, value]`，键值与键名是一致的（或者说只有键值，没有键名）
- 允许储存任何类型的唯一值，无论是原始值或者是对象引用
- 可以遍历，方法有： `add` 、 `delete` 、 `has` 、 `clear`

>>WeakSet

- 成员都是对象
- 成员都是弱引用，可以被垃圾回收机制回收，可以用来保存 `DOM` 节点，不容易造成内存泄漏
- 不能遍历，方法有 `add` 、 `delete` 、 `has`

>>Map

- 是一种类似于字典的数据结构，本质上是键值对的集合
- 可以遍历，可以跟各种数据格式转换
- 操作方法有: `set` 、 `get` 、 `has` 、 `delete` 、 `clear`

>>WeakMap

- 只接受对象作为键名（ `null` 除外），不接受其他类型的值作为键名
- 键名是弱引用，键值可以是任意的，键名所指向的对象可以被垃圾回收，此时键名是无效的
- 不能遍历，方法有 `get` 、 `set` 、 `has` 、 `delete`