1. **浏览器从输入网址到页面渲染过程**

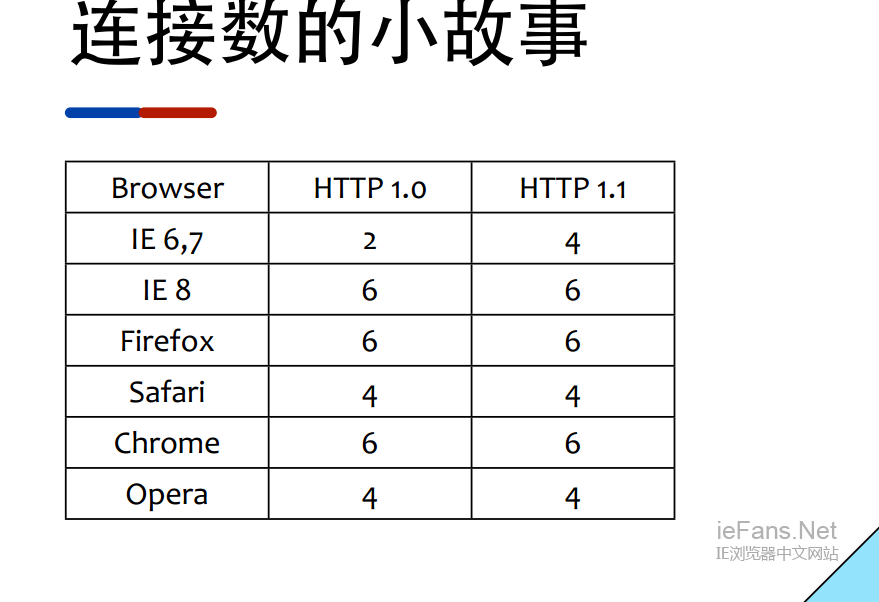
**https://www.cnblogs.com/qing-5/p/11126524.html，https://blog.csdn.net/qq\_31965515/article/details/81295151）**

1、DNS解析URL的过程

2、浏览器发送请求与服务器交互的过程

3、浏览器对接收到的html页面渲染的过程

**2.浏览器一次最大请求资源数**



1. **foreach 和 map的区别**

**forEach():** 除了抛出异常以外，没有办法中止或跳出 forEach() 循环。

**map()**:map 方法会给原数组中的每个元素都按顺序调用一次 callback 函数。callback 每次执行后的返回值（包括 undefined）组合起来形成一个新数组。 callback 函数只会在有值的索引上被调用；那些从来没被赋过值或者使用 delete 删除的索引则不会被调用。

forEach()方法不会返回执行结果，而是undefined。也就是说，forEach()会修改原来的数组。

map()方法会分配内存空间存储新数组并返回

1. **防抖和节流**

**防抖：**防抖的含义就是让某个时间期限（如上面的1000毫秒）内，事件处理函数只执行一次。

效果：如果短时间内大量触发同一事件，只会执行一次函数。

实现：那实现的关键就在于setTimeout这个函数，由于还需要一个变量来保存计时，考虑维护全局纯净，可以借助闭包来实现

**节流：**函数执行一次之后，该函数在指定的时间期限内不再工作，直至过了这段时间才重新生效。

**5、基于JS实现Ajax并发请求的控制**

promise对象的all方法可以等待所有接口皆resolve之后调用回调函数

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。

**6、垃圾回收机制**

js代码想要运行，需要操作系统或者运行时提供内存空间，来存储变量及它的值。在某些变量（例如局部变量）在不参与运行时，就需要系统回收被占用的内存空间，称为垃圾回收

**标记清除：**垃圾收集器在运行的时候会给存储在内存中的所有变量都加上标记。然后，它会去掉环境中的变量以及被环境中的变量引用的标记。而在此之后再被加上标记的变量将被视为准备删除的变量，原因是环境中的变量已经无法访问到这些变量了。最后。垃圾收集器完成内存清除工作，销毁那些带标记的值，并回收他们所占用的内存空间。

**引用计数：**引用计数根据变量被引用的次数，决定是否回收，如果被引用次数为0，则该回收了

**避免垃圾回收：**数组array优化、对象尽量复用、循环优化

1. **什么是外边距合并**

css外边距合并是指当两个垂直外边距相遇时，它们将形成一个外边距，并且合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者；但是只有普通文档流中块框的垂直外边距才会发生外边距合并，而行内框、浮动框或绝对定位之间的外边距不会合并

1. **介绍下px， pt，em，rem，vh**

**px**:基于像素的单位（屏幕上显示数据的最基本的点）。用像素定义的的文字、图片等会随屏幕的分辨率变化而变化。如果在定义字体大小时，使用px作为单位，分辨率调高，实际看到的文字就变“小”。

**Pt**:是一种固定长度的度量单位,大小为1/72英寸,如果在web上使用pt做单位的文字，字体的大小在不同屏幕下一样（DPI精度一样），但在Word中使用pt相当方便。因为使用Word主要目的都不是为了屏幕浏览，而是输出打印。当打印到实体时，pt作为一个自然长度单位就方便实用了。

**em**:相对长度单位,当用于指定字体大小时，em单位是指父元素的字体大小

**Rem**:rem是CSS3新增的一个相对长度单位，只相对根目录即HTML元素。所以可以在html标签上设置字体大小为标准，文档中的字体大小都会以此为参照。

**Vh**:vw 相对于视区的宽度：视区宽度是100vw。  
vh 相对于视区的高度：视区宽度是100vh

1. **web运用从服务器主动推送data到客户端有哪些方法？**

Javascript数据推送

Commet：基于HTTP长连接的服务器推送技术

基于WebSocket的推送方案

SSE（Server-Send Event）：服务器推送数据新方式

1. **说说你对amd和commonjs的了解？**

**（https://www.jianshu.com/p/c5cfc6063344）**

**Common**.js:CommonJS是服务器端模块的规范，Node.js采用了这个规范。CommonJS规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。

**Amd**:AMD规范则是非同步加载模块，允许指定回调函数  
AMD推荐的风格通过返回一个对象做为模块对象，CommonJS的风格通过对module.exports或exports的属性赋值来达到暴露模块对象的目的

1. **谈谈你对promise的理解？**

promise是为解决异步处理回调金字塔问题而产生的

1）pending 初始状态

2）fulfilled 成功状态

3）rejected 失败状态

Promise的状态一旦改变，就不会再变，任何时候都可以得到这个结果，状态不可以逆

Resolve：传出成功；rejected：传出失败

Promise实例生成以后，可以用then方法分别指定resolved状态和rejected状态的回调函数

promise.then(function(value) {

// success

}, function(error) {

// failure

});

1. **谈谈你对event loop的理解？**

(https://www.cnblogs.com/wancheng7/p/10137988.html)

js语言的一大特点就是单线程，单线程之所以能实现异步是因为在处理异步任务时并不是马上运行的，而是通过一个事件循环(event loop)机制来管理任务的执行

a先执行宏任务中的同步代码

b执行栈清空之后查询任务队列

c如果任务队列中有微任务，先执行

d执行完了微任务之后开始下一轮event loop，执行队列中宏任务的异步代码

1. **谈谈你对this的理解**

(https://blog.csdn.net/lxcao/article/details/52750575)

this表示当前对象，this的指向是根据调用的上下文来决定的，默认指向window对象，指向window对象时可以省略不写

****全局环境：this****始终指向的是window对象

****局部环境：a****在全局作用域下直接调用函数，this指向window

b对象函数调用，哪个对象调用就指向哪个对象

c使用 new 实例化对象，在构造函数中的this指向实例化对象

d使用call或apply改变this的指向

**使用this**:

 用于区分全局变量和局部变量，需要使用this

.返回函数当前的对象

将当前的对象传递到下一个函数

1. **谈谈你对闭包的理解**

“闭包是有权访问其他函数作用域的局部变量的一个函数”

闭包两个作用，一是读取函数内部的变量，二是让这些变量的值始终存在内存之中。

函数内部可以访问到外部的变量，而外部无法访问到内部作用域的变量。所以在某些情况下，我们不想某个变量直接暴露，就将这个变量保存在函数中，使其变为局部变量，再使用闭包将其“间接暴露”

1. **继承的几种方法**

a借助构造函数实现继承

b借助原型链实现继承

c组合继承

# 深拷贝的几种方法

1. JSON.parse(JSON.string(obj)) （ 只能拷贝对象和数组，无法拷贝函数，无法拷贝原型链上的属性和方法(Date, RegExp, Error等)）
2. for...in 循坏递归

# get和post的区别

|  |  |  |
| --- | --- | --- |
|  | Get | post |
| 缓存 | 能被缓存 | 不能被缓存 |
| 编码类型 | application/x-www-form-urlencoded | application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。 |
| 历史 | 参数保留在浏览器历史中。 | 参数不会保存在浏览器历史中。 |
| 长度限制 | 是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。 | 无限制 |
| 类型限制 | 只允许 ASCII 字符。 | 没有限制。也允许二进制数据。 |
| 安全性 | 与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。在发送密码或其他敏感信息时绝不要使用 GET ！ | POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中 |
| Tcp包 | 1个 | 2个 |

# 使用css实现一个三角形

width: 0;

height: 0;

border-width: 40px;

border-style: solid;

border-color: red transparent transparent transparent;

# 前端的性能优化

（https://zhuanlan.zhihu.com/p/113864878?from\_voters\_page=true）

**优化dom**

\* 删除不必要的代码和注释包括空格，尽量做到最小化文件。

\* 可以利用 GZIP 压缩文件。

\* 结合 HTTP 缓存文件。

**优化cssdom**

\* 减少关键 CSS 元素数量

\* 当我们声明样式表时，请密切关注媒体查询的类型，它们极大地影响了 CRP 的性能 。

**优化js**

\* async: 当我们在 script 标记添加 async 属性以后，浏览器遇到这个 script 标记时会继续解析 DOM，同时脚本也不会被 CSSOM 阻止，即不会阻止 CRP。

\* defer: 与 async 的区别在于，脚本需要等到文档解析后（ DOMContentLoaded 事件前）执行，而 async 允许脚本在文档解析时位于后台运行（两者下载的过程不会阻塞 DOM，但执行会）。

\* 当我们的脚本不会修改 DOM 或 CSSOM 时，推荐使用 async 。

\* 预加载 —— preload & prefetch 。

\* DNS 预解析 —— dns-prefetch

图片懒加载

防抖和节流

**Vue**

# vue中的计算属性和watch之间的区别

**computed**计算属性是用来声明式的描述一个值依赖了其它的值，当所依赖的值或者变量改变时，计算属性也会跟着改变；

根据一个现有数据去生成一个新数据，并且这两个数据会永久的建立关系，还会建立缓存，当无关数据改变的时候，不会重新计算而是直接使用缓存中的值

get//回调函数 当需要读取当前属性值是执行，根据相关数据计算并返回当前属性的值

set//监视当前属性值的变化，当属性值发生变化时执行，更新相关的属性数据

**watch** 监听的是已经在 data 中定义的变量，当该变量变化时，会触发 watch 中的方法；

**区别**

所以区别来源于用法，只是需要动态值，那就用计算属性；需要知道值的改变后执行业务逻辑，才用 watch

1.computed 是一个对象时，它有哪些选项？

有get和se两个选项

2.computed 和 methods 有什么区别？

methods是一个方法，它可以接受参数，而computed不能，computed是可以缓存的，methods不会。

3.computed 是否能依赖其它组件的数据？

computed可以依赖其他computed，甚至是其他组件的data

4.watch是一个对象时，它有哪些选项？

watch配置

handler

deep 是否深度

immediate 是否立即执行

# vue------provide & inject 依赖注入

（https://www.jianshu.com/p/6651dccd282c）

  通常情况下，父组件向孙组件传递数据，可以采用父子props层层传递，也可以使用bus和Vuex直接交互。在Vue2.2.0之后，Vue还提供了provide/inject选项

这对选项允许一个祖先组件向其所有子孙后代组件注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。

也就是说，在父组件只要声明了provide，在其子组件，孙组件，曾孙组件等能形成上下游关系的组件中交互，无论多深都能通过inject来访问provider中的数据。而不是局限于只能从当前父组件的prop属性来获取。注意他只做祖先通后代的单向传递的一个办法。有人这么形容：

provide就相当于加强版父组件prop，可以跨越中间组件，inject就相当于加强版子组件的props

官网不建议在应用中直接使用该办法

#### action为什么是异步的 vuex

在 mutation 中混合异步调用会导致你的程序很难调试。例如，当你能调用了两个包含异步回调的 mutation  
来改变状态，你怎么知道什么时候回调和哪个先回调呢？这就是为什么我们要区分这两个概念。在 Vuex 中，mutation 都是同步事务：

正是因为不可以在mutation中进行异步操作才要action，但是他本质上就是普通的函数，其实你不写actions也行的，只不过把常用的操作封装到actions里边用起来比较方便，也能在全局进行复用。（尤雨溪原话）

#### vue-cli2 和3的区别

（https://www.jianshu.com/p/c178e1e37b7f）

vue-cli 3.0的项目摈弃了 config 、 build 、 static 目录，新增了 public 目录，将根目录下的 index.html 放置在 public 目录下。

新增 webpack 的配置文件 vue.config.js ，可以在该文件中进行webpack的相关配置，例如 loader、开发环境等等。

新增 .browserslistrc 文件，指定了项目的目标浏览器的范围，用来确定需要转译的 JavaScript 特性和需要添加的 CSS 浏览器前缀，可以理解为浏览器兼容。

新增 babel.config.js 替代原先的.babelrc，具备和原先.babelrc一样的作用。

src文件夹中多了 views 文件夹，相比2.0，在 index.js 变为了 router.js

2.0版本相比3.0版本 有build和config文件夹等，src文件夹中有router文件夹，里面有index.js

#### 作用域插槽

（https://www.jianshu.com/p/e10baeff888d）

插槽模板是slot，它是一个空壳子，因为它的显示与隐藏以及最后用什么样的html模板显示由父组件控制。但是插槽显示的位置却由子组件自身决定，slot写在组件template的什么位置，父组件传过来的模板将来就显示在什么位置。

单个插槽

具名插槽

作用域插槽

作用域插槽内，父组件可以拿到子组件的数据。子组件可以在slot标签上绑定属性值

# 路由守卫

（https://www.jianshu.com/p/12261c3d439d）

路由跳转前做一些验证

to：即将要进入的目标路由对象

from：当前导航正要离开的路由

next：执行下一步

**全局前置守卫beforEach**

**全局后置钩子afterEach（少用）**

全局后置钩子与全局前置守卫类似，然而和守卫不同的是，这些钩子不会接受 next 函数也不会改变导航本身

**路由独享的守卫beforeEnter**

使用方法与全局守卫相同，不同的是：全局守卫可以作用于全局，路由独享守卫只作用于被设置守卫的路由

**组件内的守卫**

可以在路由组件内直接定义以下路由导航守卫：

（1）beforeRouteEnter

在渲染该组件的对应路由被 confirm 前调用

不！能！获取组件实例 this，因为当守卫执行前，组件实例还没被创建

可以通过 next 获取data中的数据

1. **beforeRouteUpdate**

如果一个在两个子路由之间跳转，是不触发beforeRouteLeave的。这会导致某些重置操作，没地方触发。在之前，我们都是用watch 的。但是通过这个勾子，我们有了更好的方式。

# 数据响应式的原理

VUE实现双向数据绑定的原理就是利用了 Object.defineProperty() 这个方法重新定义了对象获取属性值(get)和设置属性值(set)的操作来实现的。

双向数据绑定, 数据层和视图层中的数据同步, 在写入数据时视图层实时的跟着更新

1. 数据变化会同步更新视图 （热更新），我们使用Vue的数据双向绑定一般是操纵data来更新view

2. 视图变化也可以更新数据，通过事件监听实现

3. 不用再去操作DOM对象，更多精力投入到业务逻辑上

当你把一个普通的 JavaScript 对象传入 Vue 实例作为 data 选项，Vue 将遍历此对象所有的属性，并使用 Object.defineProperty 把这些属性全部转为 getter/setter。

这些 getter/setter 对用户来说是不可见的，但是在内部它们让 Vue 能够追踪依赖，在属性被访问和修改时通知变更。

每个组件实例都对应一个 watcher 实例，它会在组件渲染的过程中把 "接触" 过的数据属性记录为依赖。之后当依赖项的 setter 触发时，会通知 watcher，从而使它关联的组件重新渲染

1. 数据劫持（Object.defineproPerty）:感知数据变化  
   先劫持  
   （2）观察者模式 （实时跟新）

# 虚拟dom和抽象语法树的区别

抽象语法树的终点是渲染函数（h函数）。

渲染函数（h函数）,它既是AST的产物，也是vnode(虚拟节点)的起源。h函数里面是不含指令的。

抽象语法树不会进行diff算法的并且抽象语法树不会直接生成虚拟节点，抽象语法树最终生成的是渲染函数的

**传统dom**：浏览器收到第一个DOM请求后并不知道还有9次更新操作，因此会马上执行流程，最终执行10次。

**虚拟dom:**一次操作中有10次更新DOM的动作，虚拟DOM不会立即操作DOM，而是将这10次更新的diff内容保存到本地一个JS对象中，最终将这个JS对象一次性attch到DOM树上，再进行后续操作，避免大量无谓的计算量。

用JS对象模拟DOM节点的好处是，页面的更新可以先全部反映在JS对象(虚拟DOM)上，操作内存中的JS对象的速度显然要更快，等更新完成后，再将最终的JS对象映射成真实的DOM，交由浏览器去绘制。

虚拟DOM是用Object来代表一颗节点，这个Object叫做VNode，然后使用两个VNode进行对比，根据对比后的结果修改真实DOM。

为什么是两个VNode？因为每次渲染都会生成一个新的VNode，然后和上一次渲染时用的VNode进行对比。然后将这一次新生成的VNode缓存，用来进行下一次对比。

# 父子组件的生命周期先后执行顺序

**加载渲染过程**：父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount->子mounted->父mounted

子组件挂载完成后，父组件还未挂载。所以组件数据回显的时候，在父组件mounted中获取api的数据，子组件的mounted是拿不到的。

**更新过程：**父beforeUpdate->子beforeUpdate->子updated->父updated

**销毁过程：**父beforeDestroy->子beforeDestroy->子destroyed->父destroyed

# v-if 和 v-show 有什么区别

相同点：v-if与v-show都可以动态控制dom元素显示隐藏

不同点：v-if显示隐藏是将dom元素整个添加或删除，而v-show隐藏则是为该元素添加css--display:none，dom元素还在。



# Vue中给data中的对象属性添加一个新的属性时会发生什么，如何解决

数据改变，视图未更新

原因在于****在Vue实例创建时， 新的属性并未声明，因此就没有被Vue转换为响应式的属性，自然就不会触发视图的更新****，这时就需要使用Vue的全局api—— ****$set()****：

三个参数：this.$set("改变的对象"，"改变的对象属性"，"值")