目录

[介绍下项目 2](#_Toc35606849)

[js有哪些数据类型 2](#_Toc35606850)

[JSON是什么？ 2](#_Toc35606851)

[原型 3](#_Toc35606852)

[this 3](#_Toc35606853)

[变量提升 6](#_Toc35606854)

[前端模块化 7](#_Toc35606855)

[commonjs规范（nodejs） 7](#_Toc35606856)

[AMD规范（require.js） 7](#_Toc35606857)

[CMD规范(sea.js) 7](#_Toc35606858)

[Es6规范 7](#_Toc35606859)

[js实现跨域 8](#_Toc35606860)

[jsonp 8](#_Toc35606861)

[nginx的反向代理 8](#_Toc35606862)

[webpack 8](#_Toc35606863)

[闭包 8](#_Toc35606864)

[js实现继承 9](#_Toc35606865)

[原型链实现继承 9](#_Toc35606866)

[借助构造函数实现继承 9](#_Toc35606867)

[组合继承 9](#_Toc35606868)

[es6的class和extends关键字 9](#_Toc35606869)

[js异步 9](#_Toc35606870)

[有那些是异步操作 9](#_Toc35606871)

[异步编程 9](#_Toc35606872)

[Vue 10](#_Toc35606873)

[理解生命周期 10](#_Toc35606874)

[生命周期执行顺序 10](#_Toc35606875)

[父子组件通信 11](#_Toc35606876)

[修改第三方ui库样式 12](#_Toc35606877)

[Es6 12](#_Toc35606878)

[let/const 12](#_Toc35606879)

[箭头函数 13](#_Toc35606880)

[解构赋值 13](#_Toc35606881)

[promise和async/await 13](#_Toc35606882)

[模块化：export/import 13](#_Toc35606883)

[防抖节流 13](#_Toc35606884)

[防抖 14](#_Toc35606885)

[节流 14](#_Toc35606886)

[html5 15](#_Toc35606887)

[声明方式 15](#_Toc35606888)

[新特性 15](#_Toc35606889)

[视频和音频 16](#_Toc35606890)

[Web storage本地存储 16](#_Toc35606891)

[浏览器渲染页面 17](#_Toc35606892)

# 介绍下项目

Vue 个人博客 2019-09-2019-11

1，简介：一个可以发布文章以及评论的个人博客

前端：NuxtJs、ElementUI、Vuex、Axios

后台：vue-element-template

后端：node的express框架搭建的简单服务提供api接口，用sequelize操作mysql数据库

# js有哪些数据类型

基本类型和引用类型

基本类型：string字符串、number数字、boolean布尔、Null空、Undefined未定义、symbo(es6)

引用类型：Array数组、Object对象，Date日期，Function函数，RegExp

区别：引用类型值可添加属性和方法，而基本类型值则不可以。

# JSON是什么？

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它是基于JavaScript的一个子集。数据格式简单, 易于读写, 占用带宽小。

格式：采用键值对，例如：{'age':'12', 'name':'back'}

json.stringify可以将对象转换为 JSON 字符串

JSON.parse可以将 JSON 字符串转换为对象

# ajax实现步骤

# 原型

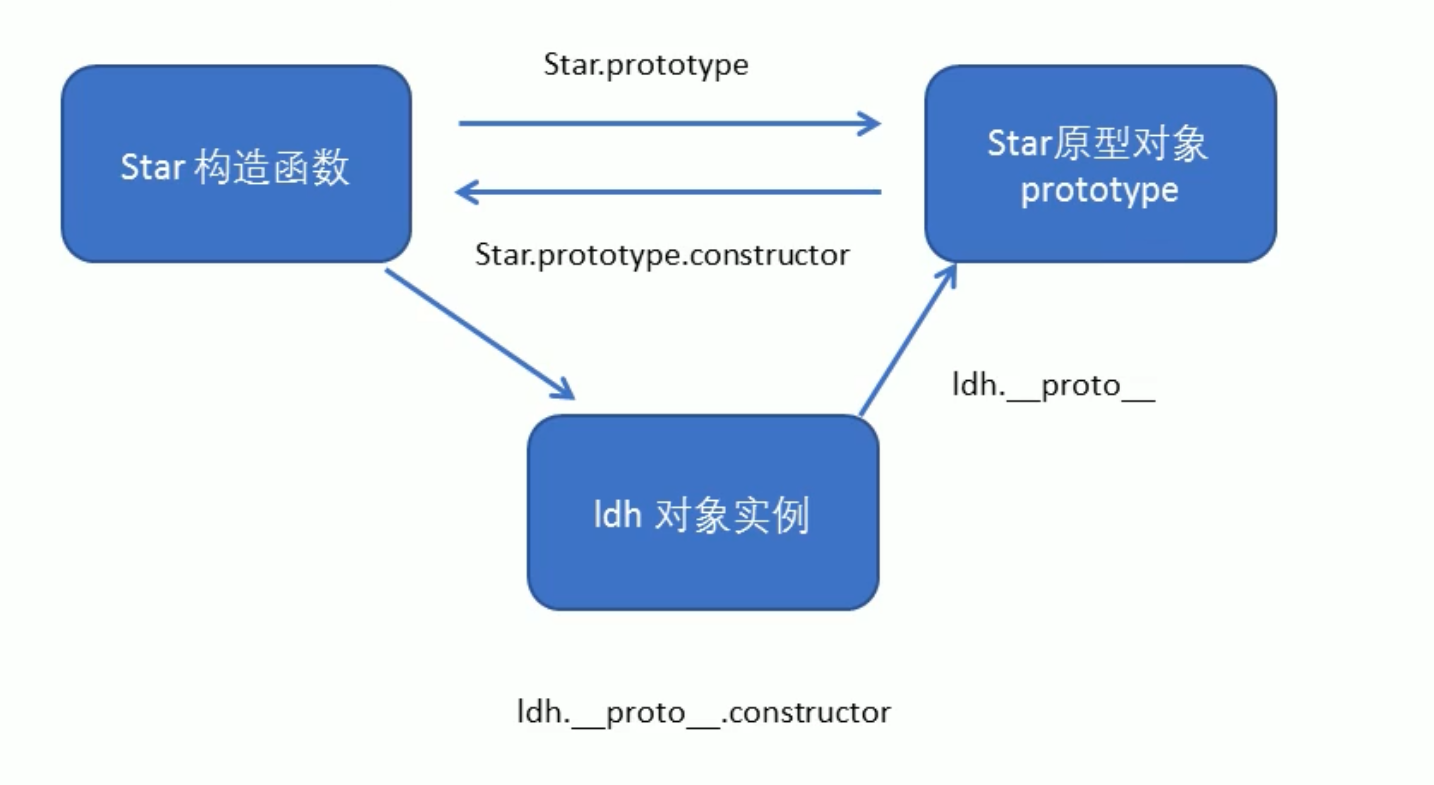
prototype原型对象，每一个构造函数都有一个prototype原型对象指向一个对象

\_\_proto\_\_对象原型，每一个对象都有一个\_\_proto\_\_对象原型指向原型对象

constructor构造函数，对象原型（\_\_proto\_\_）和原型对象（prototype）都存在一个constructor，它指向构造函数本身。

构造函数，对象实例，原型对象（prototype）三者之间的关系：

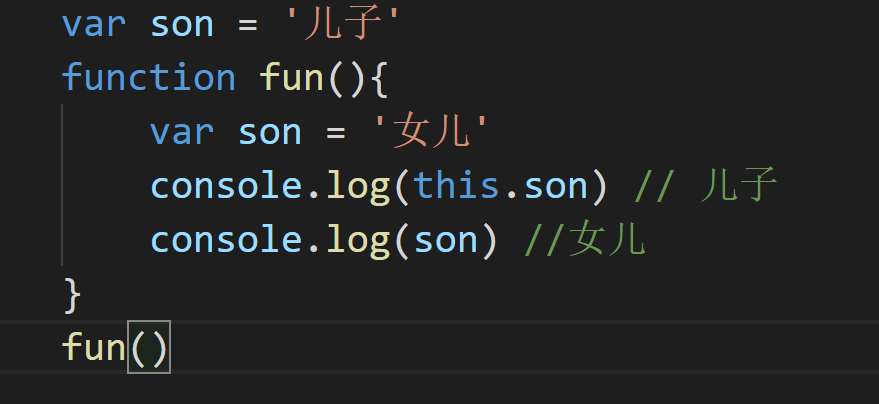
构造函数通过prototype指向原型对象（prototype），原型对象（prototype）通过constructor指回构造函数，而通过构造函数创建出来对象实例又通过\_\_proto\_\_指向原型对象（prototype）



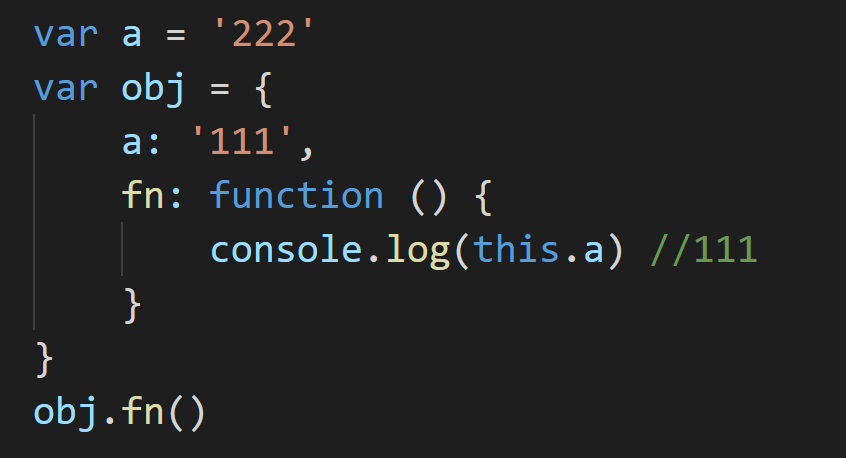
# this

this指向：一般来说，**哪个对象调用函数，函数里面的this指向哪个对象**

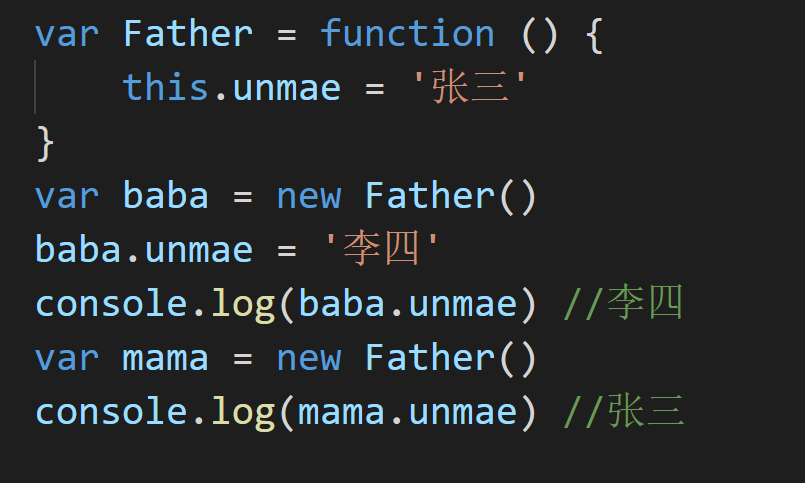
**普通函数调用：指向全局对象window**



对象函数调用：那个对象调用，this就指向那个对象



构造函数调用：this指向构造函数的实例化对象



call,apply,bind:都能改变this指向，

call和apply都会调用函数， call和apply的第一个参数都是this指向的对象，不同点在后面的参数，call的参数是直接放进去用逗号分隔，而apply则是传一个数组。bind和call一样，只不过bind不会调用函数。

箭头函数：箭头函数里的this会继承外面的环境



var that = this;在函数外面添加var that = this，函数内部使用that获取外面的this数据

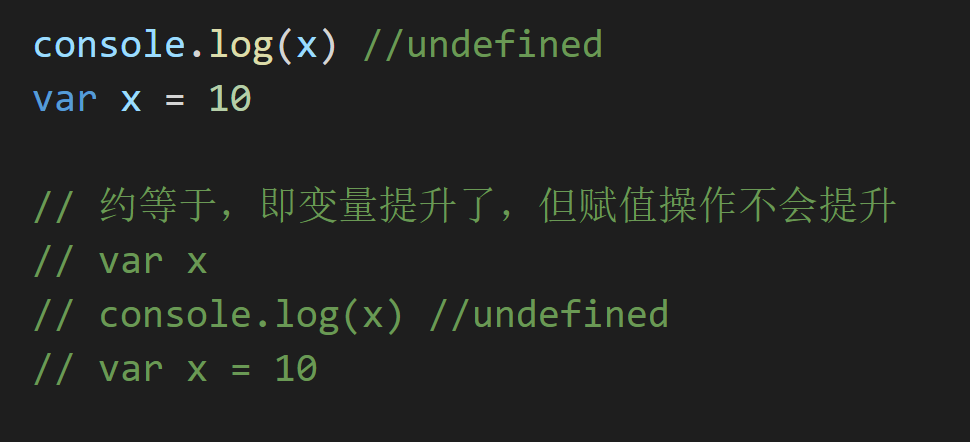
# 变量提升

JavaScript 只有声明的变量会提升，初始化的不会。

只有var会提升，let不会提升，const(定义常量)也不会提升

使用函数表达式（var fun = function(){}）定义函数不会提升

初始化变量（var str =”str”）不会提升



# 前端模块化

## commonjs规范（nodejs）

module.exports=value或者直接exports.xxx=value暴露模块

require引入模块

本质：每个模块内部，module变量代表当前模块，这个变量是一个对象，他的exports属性是对外的接口，加载某个模块其实是加载该模块的modules.exports属性

## AMD规范（require.js）

## CMD规范(sea.js)

## Es6规范

通过export/import

export default默认导出 export default { }

export 按需导出 export var name = ‘aaa’; export function(){ console.log(‘aaa’)}

import from导入

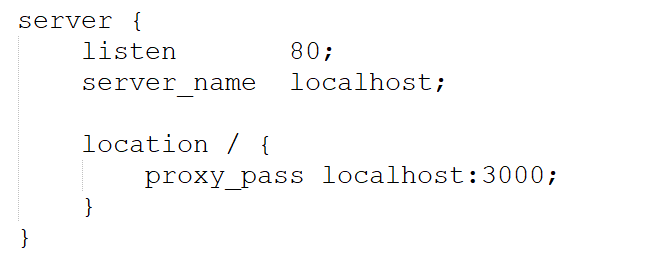
# js实现跨域

## jsonp

由于浏览器的同源策略，要从不同的域（网站）访问数据会产生跨域问题，img的src(获取图片)，link的herf（获取css），script的src(获取JavaScript)，这三个属性不属于同源策略，都可以跨域获取数据，因此，jsonp应运而生

JSONP实现跨域的原理简单的说，就是动态创建script标签，然后利用script的src 不受同源策略约束来跨域获取数据。步骤：前端发送请求，后端接收到请求后，把要返回的数据填充进js语句里并返回给前端js语句，然后前端执行返回的js语句，这样前端就可以获取后端的数据

## nginx的反向代理



## webpack

安装webpack-dev-server插件，添加proxy的属性

# 闭包

变量作用域，全局变量和局部变量，函数内部可以使用全局变量，函数外部不能使用函数内部的局部变量

简单的说就是一个作用域可以访问另一个函数内部的局部变量

一个函数里嵌套一个函数，这里面的函数可以访问外面的函数，因此，外面的函数是闭包函数。

# js实现继承

## 原型链实现继承

## 借助构造函数实现继承

在子构造函数里通过使用call()或者apply()调用父构造函数，将父构造函数里的this修改为子构造函数的this

## 组合继承

将原型链和借用构造函数的技术组合到一起

## es6的class和extends关键字

class 函数名{}，里面写一个constructor()函数，如果要继承父类的constructor，要在constructor里面首先调用一下super()

# js异步

## 有那些是异步操作

1. 定时器
2. 事件绑定，鼠标事件：onclick点击，键盘事件：onkeydown键盘按下，onkeyup键盘被松开
3. ajax请求

## 异步编程

1. 回调函数， 就是将B函数被作为参数传递到A函数里，在A函数执行完后再执行B
2. Promise，new Promise，它接收一个函数作为参数，这个函数有resolve，reject两个参数，在执行完异步操作后调用resolve方法
3. async/await，es6语法糖

# Vue

## 理解生命周期

什么是生命周期：Vue实例从创建到销毁的过程，就是生命周期

什么是钩子函数：Vue实例从创建到销毁的过程中也会运行一些叫做生命周期钩子的函

首先new Vue()新建实例，然后初始化事件和生命周期，

然后执行beforeCreate，在执行这个钩子时，data和methods中的数据都还没有被初始化，

然后执行created，在执行这个钩子时，data和methods中的数据已经被初始化好了，如何要操作data或者调用methods方法，最早要在created中操作。

然后判断实例里是否有el选项，如果有el选项，则继续向下编译，如果没有el选项，则停止编译，也就意味着停止了生命周期，直到在该vue实例上手动挂载，即调用vm.$mount(“#app”)，才会继续向下执行。

然后判断有没有template选项，如果有，则将template解析成抽象语法树，然后再通过render函数编译，如果没有template选项，则将el的外部html作为template编译

然后执行beforeMount，此时模板已经在内存中渲染好了，但还没把模板挂在到页面中去

执行完beforeMount后，内存中的渲染好的模板会真实的替换到页面中去。

然后执行mounted，此时代表实例已经完全被创建好了。如果需要操作dom,至少要在mounted中执行。

当实例的data发生改变时，会立即执行beforeUpdate，此时，data的数据是新的，但页面中的数据还是旧的，

执行完beforeUpdate后，虚拟dom重新渲染更新

当执行到updated时，页面和data数据都是最新的

beforeDestroy

destroyed

## 生命周期执行顺序

beforeEach: 路由全局前置守卫，可用于登录验证、加载进度条等。

beforeEnter: 路由独享守卫，和全局前置守卫差不多

beforeRouteEnter: 进入组件前的钩子，可导航完成前获取数据

beforeCreate:组件生命周期，不能访问this。

created:组件生命周期，可以访问this，不能访问dom。

beforeMount:组件生命周期

mounted:访问/操作dom。

beforeRouteLeave:路由组件的组件离开路由前钩子，可取消路由离开。一般用在还没保存就突然离开，如发布文章写到一半突然离开，没有保存草稿时，可用next(false)取消离开

父子组件加载渲染过程

父组件beforeCreated ->父组件created ->父组件beforeMounted ->子组件beforeCreated ->子组件created ->子组件beforeMounted ->子组件mounted -> 父组件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

多个子组件的created执行顺序为父组件内子组件DOM顺序

多个子组件的mounted顺序无法保证，跟子组件本身复杂程度有关

父组件一定在所有子组件结束mounted钩子之后，才会进入mounted钩子

## 父子组件通信

父组件通过 prop 给子组件下发数据，子组件通过事件给父组件发送消息

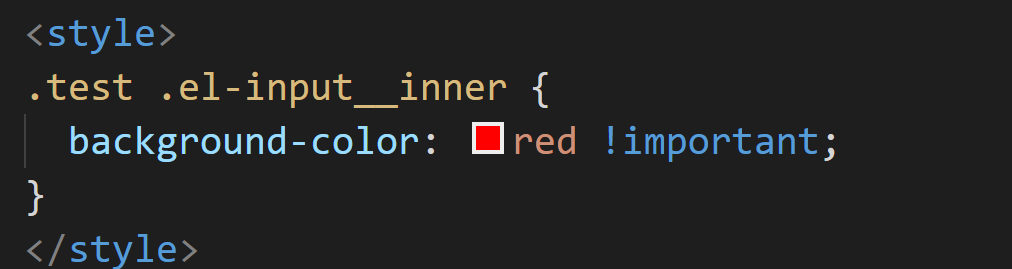
**父传子的实现方式就是通过props属性，子组件通过props属性接收从父组件传过来的值，而父组件 使用v-bind 子组件中预留的变量名进行传值**

**子传父的实现方式是通过 this.$emit ，在子组件里定义$emit，它接收两个参数，第一个是为父组件定义的方法名，第二个是需要传递的参数。在父组件中引用子组件处添加v-on定义好的方法名=“要执行的方法”**

## **修改第三方ui库样式**

vue的style标签中，会添加一个scoped属性，防止样式的污染。

增加一个不带scoped的style标签，但在里面写的话会污染全局，可以在组件添加一个父class，然后选择器定位到该父class下要修改的样式进行修改



在不带scoped的style标签里使用深度选择器/deep/



# Es6

## let/const

var定义的变量，可以重复声明、要么是全局作用域要么是局部作用域。没有块的概念，可以跨块访问, 不能跨函数访问。

let定义的变量，同一个作用域不可以重复声明、拥有块级作用域。只能在块作用域里访问，不能跨块访问，也不能跨函数访问。

const用来定义常量，使用时必须初始化(即必须赋值)，只能在块作用域里访问，如果定义的是基本类型（string，number等），修改会报错，如果定义的是引用类型（object，array等）可以修改里面的属性和方法

## 箭头函数

1，语法上比函数表达式简洁

2，箭头函数里面的this是继承外部环境的this

3，不能使用argument对象，但能使用扩展运算符...arg获取参数

1. 不能使用new构造函数，箭头函数没有this和prototype

## 解构赋值

ES6中允许按照一定模式，从数组和对象中提取值，对变量进行赋值

## promise和async/await

promise有几种状态，什么时候会进入catch/then

三个状态：pending、fulfilled、reject

两个过程：pedding->fulfilled、pedding->rejected

当pending为fulfilled时，会进入then

当pending为rejected时，会进入catch

## 模块化：export/import

通过export/import

export default默认导出

export 按需导出

import from导入

# 防抖节流

函数防抖和节流是优化高频率执行js代码的一种手段，如scroll（监听滚动条），resize（监听浏览器窗口），input的keyup（键盘按键松开），会不断地调用绑定在事件上的回调函数，极大地浪费资源，降低前端性能。为了优化体验，需要对这类事件进行调用次数的限制。

## 防抖

在规定时间里只让最后一次执行，前面的不生效。

举例：用户一直点击登录按钮，只让最后一次点击的生效。

利用定时器和闭包实现，首先定义一个防抖函数，防抖函数里面定义一个保存定时器的变量timer，然后return一个函数，函数里面首先清除定时器timer，然后再定义timer定时器，定时器里面执行要执行的操作



## 节流

一个函数执行一次后，只有大于设定的周期才会执行第二次

举例：监听浏览器滚动条，窗口，以及input输入框里的keyup事件

利用定时器和闭包，首先定义一个节流函数，节流函数里定义一个booler变量为true，然后return一个函数，里面if判断booler是否为true，如果为true则重新复制为false，然后执行要执行的操作，然后写一个定时器，讲booler修改为true



# html5

## 声明方式

html4：严格模式，过渡模式，框架集模式

html5：只需要<!DOCTYPE html>

## 新特性

更好的语义化标签：常用的：header，footer，nav（导航），article（内容区块），aside（侧边栏），section（一组或者一节内容），details（描述文档细节），dialog（对话框）

优点：对搜索引擎友好，有利于seo，具有可读性，便于团队开发和维护，页面结构比较清晰

音频视频，注意兼容性问题

表单控件，常用的：email，tel，color，number，主要就有利于移动端做表单，如tel，number，email用这些标签虚拟键盘会切换到相应模式

5个本地存储api，常用的localstorage和sessionstorage

画布/canvas

5个地理定位api，常用getCurrentPosition()。一般用第三方的ip库，如用新浪，搜狐api获取客户端的ip地址，邮编，城市名；

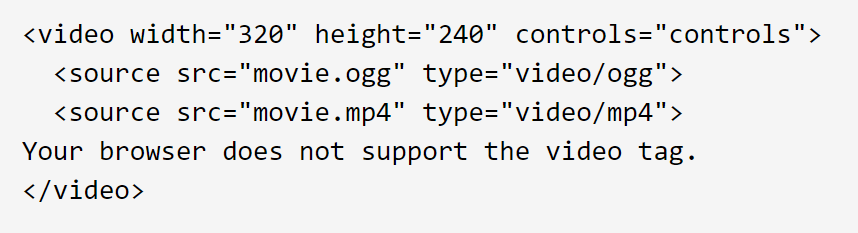
5个拖拽释放api

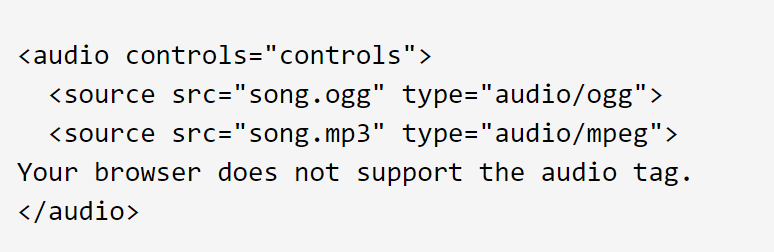
Web Workers

## 视频和音频

视频和音频都允许多个 source 元素。source 元素可以链接不同的文件。浏览器将使用第一个可识别的格式

视频：添加video标签设置视频，获取video标签可以调用play()播放，pause()暂停这两个方法，paused属性检测是否暂停





## Web storage本地存储

会话存储session storage

本地存储local storage

保存数据：localStorage.setItem(key,value);

读取数据：localStorage.getItem(key);

删除单个数据：localStorage.removeItem(key);

删除所有数据：localStorage.clear();

得到某个索引的key：localStorage.key(index);

本地存储和会话存储，两种都会把存储的值隐式转换为字符串，都遵循同源策略，两者区别在于本地存储没有过期时间设置，除非手动删除或者程序删除，而会话存储刷新不会失效（一个网页设置了sessionstorage，打开另一个网页不能访问，只能在当前页访问），关闭浏览器窗口才会失效。

# 浏览器渲染页面

1. 用户输入url，浏览器发送请求，请求对应的url资源
2. 解析html生成dom树，渲染引擎首先解析html文档，将标签转化为dom树的node节点(包括js生成的标签)，生成dom树
3. 构建render树，解析对应的css样式文件，包括js生成的css和外部引入的css
4. 布局render树，对渲染树的每个节点进行布局处理，计算节点的在屏幕的显示的形状位置
5. 绘制render树，遍历渲染树，在屏幕画出每一个节点

影响页面渲染速度主要有：**重排（重构/回流/reflow）和重绘（repaint或redraw），实际开发中应该避免频繁操作dom，尽量少使用jquery**