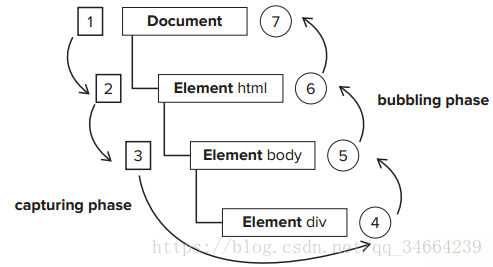
事件绑定，事件冒泡，事件捕获，事件绑定与解绑

DOM事件被触发时会经历三个不同的阶段，

捕获阶段：先由文档的根节点document往事件触发对象，从外向内捕获事件对象；

目标阶段：到达目标事件位置（事发地），触发事件；

冒泡阶段：再从目标事件位置往文档的根节点方向回溯，从内向外冒泡事件对象。



jquery中，无论是on添加事件，还是直接加事件函数，同一个事件都可以多次绑定

函数中

retrun true； 返回正确的处理结果。

return false；返回错误的处理结果，终止处理；阻止提交表单；阻止执行默认的行为。

return；把控制权返回给页面。

e.stopPropagation()阻止冒泡

事件代理

事件代理也叫事件委托，就跟快递点代收一样，不是所有人都去等快递，而是委托快递点先接收然后判断接收人是谁，再转交给他，委托快递点的人就是dom节点

第一，现在委托前台的同事是可以代为签收的，即程序中的现有的dom节点是有事件的；

第二，新员工也是可以被前台MM代为签收的，即程序中新添加的dom节点也是有事件的。

减少与dom的交互次数，提高性能。

原理：

利用事件冒泡的特性，将里层的事件委托给外层事件，根据event对象的属性进行事件委托，改善性能。

使用事件委托能够避免对特定的每个节点添加事件监听器；事件监听器是被添加到它们的父元素上。

事件监听器会分析从子元素冒泡上来的事件，找到是哪个子元素的事件。

利用事件的冒泡原理来实现的，从最深层节点开始逐步向上传播事件，当用事件委托的时候，根本就不需要去遍历元素的子节点，只需要给父级元素添加事件就好了，其他的都是在js里面的执行，这样可以大大的减少dom操作，这才是事件委托的精髓所在。focus，blur之类的，本身就没用冒泡的特性

JS几种继承方式

通过模仿实现

（1）属性拷贝：将对象的成员复制一份给需要继承的对象

如果继承过来的成员是引用类型的话,

那么这个引用类型的成员在父对象和子对象之间是共享的,

也就是说修改了之后, 父子对象都会受到影响.

（2）原型式继承：借用构造函数的原型对象实现继承

父构造函数的原型对象和子构造函数的原型对象上的成员有共享问题

只能继承父构造函数的原型对象上的成员, 不能继承父构造函数的实例对象的成员

（3）原型链继承：子构造函数.prototype = new 父构造函数()

不能给父构造函数传递参数，父子构造函数的原型对象之间有共享问题

（4）借用构造函数：使用call和apply借用其他构造函数的成员,

可以解决给父构造函数传递参数的问题, 但是获取不到父构造函数原型上的成员.也不存在共享问题

（5）借用构造函数 + 原型式继承（3+4）：

上面代码解决了 父构造函数的属性继承到了子构造函数的实例对象上了,

并且继承了父构造函数原型对象上的成员

解决了给父构造函数传递参数问题

（6）借用构造函数 + 深拷贝：

这样就将Person的原型对象上的成员拷贝到了Student的原型上了, 这种方式没有属性共享的问题.

（7）ES6新增继承方式，Class 可以通过extends关键字实现继承

new 操作符具体干了什么

创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型

属性和方法被加入到 this 引用的对象中

新创建的对象由 this 所引用，并且最后隐式的返回 this

闭包是什么（与匿名函数的区别）

匿名函数就是没有名字的函数

闭包就是有权访问另一个函数作用域中变量的函数，常见方式：一个函数内部创建另一个函数，通过另一个函数访问这个函数的局部变量，优点也是缺点就是可以把局部变量驻留在内存中，可以避免使用全局变量。

由于闭包里作用域返回的局部变量资源不会被立刻销毁回收，所以可能会占用更 多的内存。过度使用闭包会导致性能下降，建议在非常有必要的时候才使用闭包。

深浅拷贝

浅拷贝

拷贝原对象引用

可以使用Array.prototype.slice()也可以完成对一个数组或者对象的浅拷贝

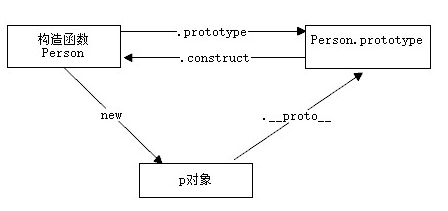
Object.assign()方法

深拷贝

最常用的方式就是 JSON.parse(JSON.stringify(目标对象)，缺点就是只能拷贝符合JSON数据标准类型的对象

JavaScript 原型，原型链，它们有何特点

JavaScript是基于原型的，每个构造函数(constructor)都有一个原型对象(prototype),原型对象都包含一个指向构造函数的指针,而实例(instance)都包含一个指向原型对象的内部指针.



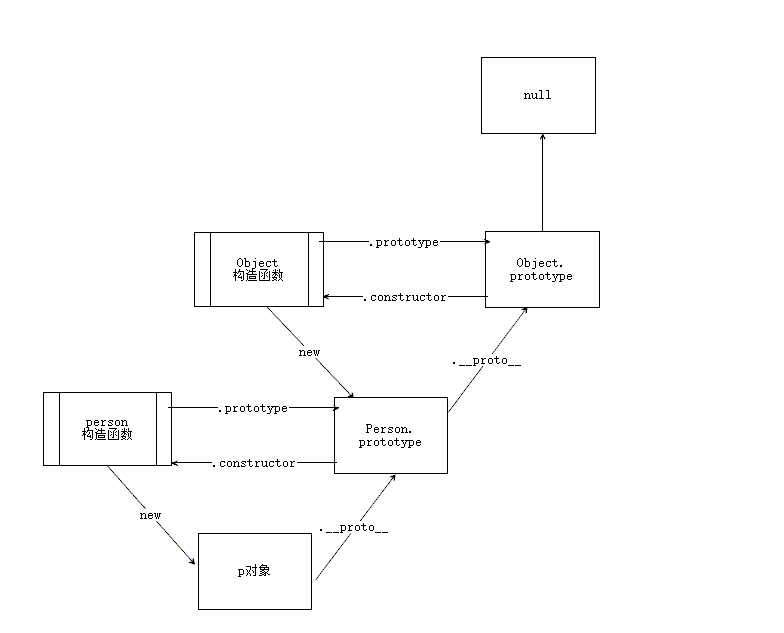
图解：

每一个构造函数都拥有一个prototype属性，这个属性指向一个对象，也就是原型对象

原型对象默认拥有一个constructor属性，指向指向它的那个构造函数

每个对象都拥有一个隐藏的属性[[prototype]]，指向它的原型对象

JavaScript中所有的对象都是由它的原型对象继承而来。而原型对象自身也是一个对象，它也有自己的原型对象，这样层层上溯，就形成了一个类似链表的结构，这就是原型链。所有原型链的终点都是Object函数的prototype属性。Objec.prototype指向的原型对象同样拥有原型，不过它的原型是null，而null则没有原型。



同步和异步的区别，怎么异步加载 JavaScript

同步模式又称阻塞模式。javascript 在默认情况下是会阻塞加载的。当前面的 javascript 请求没有处理和执行完时，会阻止浏览器的后续处理，js调用放在head里头就会阻塞。

异步模式又叫非阻塞，浏览器在下载执行 js 同时，还会继续进行后续页面的处理

异步加载 JavaScript

动态添加 script 标签

defer

async

defer属性和async都是属于 script 标签上面的属性，两者都能实现 JavaScript 的异步加载。不同之处在于：async 在异步加载完成的时候就马上开始执行了，defer 会等到 html 加载完毕之后再执行

跨域问题的产生，怎么解决它

解决跨域问题有很多种方式，常用的就是以下几种：

jsonp 跨域：动态创建script，再请求一个带参网址实现跨域通信.缺点就是只能实现 get 一种请求

document.domain + iframe跨域：两个页面都通过js强制设置document.domain为基础主域，就实现了同域.但是仅限主域相同，子域不同的跨域应用场景

跨域资源共享（CORS）：只服务端设置Access-Control-Allow-Origin即可，前端无须设置，若要带cookie请求：前后端都需要设置

nginx反向代理接口跨域：同源策略是浏览器的安全策略，不是HTTP协议的一部分。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，也就不存在跨越问题

WebSocket协议跨域

对 this 的理解

this 永远指向最终调用它的那个对象，除非改变指向或箭头函数

apply()、call()和 bind() 是做什么的，它们有什么区别

相同点：三者都可以改变 this 的指向

不同点：

apply 方法传入两个参数：一个是作为函数上下文的对象，另外一个是作为函数参数所组成的数组

call 方法第一个参数也是作为函数上下文的对象，但是后面传入的是一个参数列表，而不是单个数组

bind 接受的参数有两部分，第一个参数是是作为函数上下文的对象，第二部分参数是个列表，可以接受多个参数

apply、call 方法都会使函数立即执行，因此它们也可以用来调用函数

bind 方法不会立即执行，而是返回一个改变了上下文 this 后的函数。而原函数 func 中的 this 并没有被改变，依旧指向全局对象 window

bind 在传递参数的时候会将自己带过去的参数排在原函数参数之前

什么是内存泄漏，哪些操作会造成内存泄漏（垃圾回收机制后面补充）

是指一块被分配的内存既不能使用，又不能回收，直到浏览器进程结束。

可能造成内存泄漏的操作：

意外的全局变量

闭包

循环引用

被遗忘的定时器或者回调函数

对AMD和CMD的理解，它们有什么区别

AMD和CMD都是为了解决浏览器端模块化问题而产生的，AMD规范对应的库函数有 Require.js，CMD规范是在国内发展起来的，对应的库函数有Sea.js

AMD和CMD最大的区别是对依赖模块的执行时机处理不同

1、AMD推崇依赖前置，在定义模块的时候就要声明其依赖的模块

2、CMD推崇就近依赖，只有在用到某个模块的时候再去require

ES6你不知道的事（十大新特性）

1. 块级作用域变量

1、let定义块级作用域变量

没有变量的提升，必须先声明后使用

let声明的变量，不能与前面的let，var，conset声明的变量重名

const声明变量的同时必须赋值，const声明的变量必须初始化，一旦初始化完毕就不允许修改

2、const声明变量也是一个块级作用域变量

const声明的变量没有“变量的提升”，必须先声明后使用

const声明的变量不能与前面的let， var ， const声明的变量重名

1. 关于函数

可以给形参函数设置默认值，箭头函数，

同时复习一下this的认识

1.在函数体外，this指的就是window对象

2.在函数替内，谁调用函数this就指向谁

3.在构造函数中，this指的是新创建的对象

4.在html标签中，this指的是当前的这个标签元素

5.在ES6中，对于箭头函数，要看它在哪里创建的，和当前函数的作用域一致。

1. 关于数组的展开运算

在数组之前加上三个点（...），可以根据数组的展开运算用数组给函数批量的传参

1. 关于apply和call

apply和call，都是对象本身没有某个属性或者方法，去引用其他对象的属性或方法，也就是说两者都可以改变this的属性

不同之处

apply(this的指向，数组/arguments) apply括号里的是谁，调用的前面的函数里面的this就是谁

call(this的指向，参数1，参数2，参数3)

1. 关于解构赋值

以前我们互换两个变量的值，需要借助第三个变量，利用解构赋值，就方便很多了

Json数据也可以用结构来操作

1. string中加入include方法

includes("字符"); 用于判断字符串中是否包含某个字符，可以大大简化条件语句

1. 创建对象

class 感觉跟java里的很像了

1. 继承

//ES6的继承，有两条继承链，构造函数 和 原型函数条

console.log( Dog.prototype)//Animal {}

console.log( Dog.constructor==Animal.constructor)//true

console.log( Dog.prototype.\_\_proto\_\_==Animal.prototype)//true

1. Set（）和Map（）

set（）有序列表集合（没有重复）包含add()、has()、delete()、clear()等

Map() 用来存放键值对的集合 key/value，get(key)根据key值取得value，has() 判断是否存在某个键值对，clear() 清空集合

Proxy

async函数 await命令

Class类

Module 体系 模块的加载和输出方式

Promise对象及和setTimeout有什么区别

简单说就是一个容器，里面保存着某个未来才会结束的事件（通常是一个异步操作）的结果

有三种状态：pending（进行中）、fulfilled（已成功）和 rejected（已失败）。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。

特点：

对象的状态不受外界影响

一旦状态改变，就不会再变，任何时候都可以得到这个结果

Promise 新建后就会立即执行，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，返回一个新的Promise实例

Promise.prototype.catch 用于指定发生错误时的回调函数,具有“冒泡”性质，会一直向后传递，直到被捕获为止。也就是说，错误总是会被下一个catch语句捕获，返回的还是一个Promise对象，因此还可以用then方法

vue中的nextTick异步更新dom操作是先判断是否支持Promise，如果不支持就判断是否支持MutationObserval,如果也不支持的话，最后才是setTimeout

console.log(1);

setTimeout(()=>{

console.log(2)

},100)

Promise.resolve().then(()=>{

console.log(3);

})

console.log(4);

结果是1432，Promise是一个micro task,我们的主线程是一个task。micro task会在task后面执行，然后才会接着执行下一个task。而setTimeout的返回函数是一个新的task,所以这里Promise的执行会先于新task执行。根据HTML标准，一个task执行完后，UI会重渲染，所以micro task更新完数据后再渲染dom的操作要比setTimout的性能要好。如果使用setTimeout的话，会有两次ui重渲染。

async 函数以及 awit 命令

async 函数是Generator函数的语法糖，Generator 函数是 ES6 提供的一种异步编程解决方案，具体再看吧

async 函数返回一个 Promise 对象，可以使用 then 方法添加回调函数。当函数执行的时候，一旦遇到 await 就会先返回，等到异步操作完成，再接着执行函数体内后面的语句

async 函数内部 return 语句返回的值，会成为 then 方法回调函数的参数

async 函数返回的 Promise 对象，必须等到内部所有 await 命令后面的 Promise 对象执行完，才会发生状态改变，除非遇到 return 语句或者抛出错误

async 函数内部抛出错误，会导致返回的 Promise 对象变为 reject 状态。抛出的错误对象会被 catch 方法回调函数接收到

awit使用注意点：

await 命令后面的 Promise 对象，运行结果可能是 rejected，所以最好把 await 命令放在 try...catch 代码块中

多个 await 命令后面的异步操作，如果不存在继发关系，最好让它们同时触发

await 命令只能用在 async 函数之中，如果用在普通函数，就会报错

export 与 export default有什么区别

export 与 export default 均可用于导出常量、函数、文件、模块等

在一个文件或模块中，export、import 可以有多个，export default 仅有一个

通过 export 方式导出，在导入时要加 { }，export default 则不需要

使用 export default命令，为模块指定默认输出，这样就不需要知道所要加载模块的变量名; export 加载的时候需要知道加载模块的变量名

export default 命令的本质是将后面的值，赋给 default 变量，所以可以直接将一个值写在 export default 之后

前端性能优化

1. 尽可能的减少 HTTP 的请求数 content

2. 使用 CDN（Content Delivery Network） server

3. 添加 Expires 头(或者 Cache-control ) server

4. Gzip 组件 server

5. 将 CSS 样式放在页面的上方 css

6. 将脚本移动到底部（包括内联的） javascript

7. 避免使用 CSS 中的 Expressions css

8. 将 JavaScript 和 CSS 独立成外部文件 javascript css

9. 减少 DNS 查询 content

10. 压缩 JavaScript 和 CSS (包括内联的) javascript css

11. 避免重定向 server

12. 移除重复的脚本 javascript

13. 配置实体标签（ETags） css

14. 使 AJAX 缓存

对JS引擎执行机制的理解

JavaScript 是单线程语言

JavaScript 的 Event Loop 是 JS 的执行机制, 也就是事件循环

JavaScript 将任务分为同步任务和异步任务，执行机制就是先执行同步任务，将同步任务加入到主线程，遇到异步任务就先加入到 event table ，当所有的同步任务执行完毕，如果有可执行的异步任务，再将其加入到主线程中执行

console.log(1)

setTimeout(function(){

console.log(2)

},0)

console.log(3)

// 1 3 2