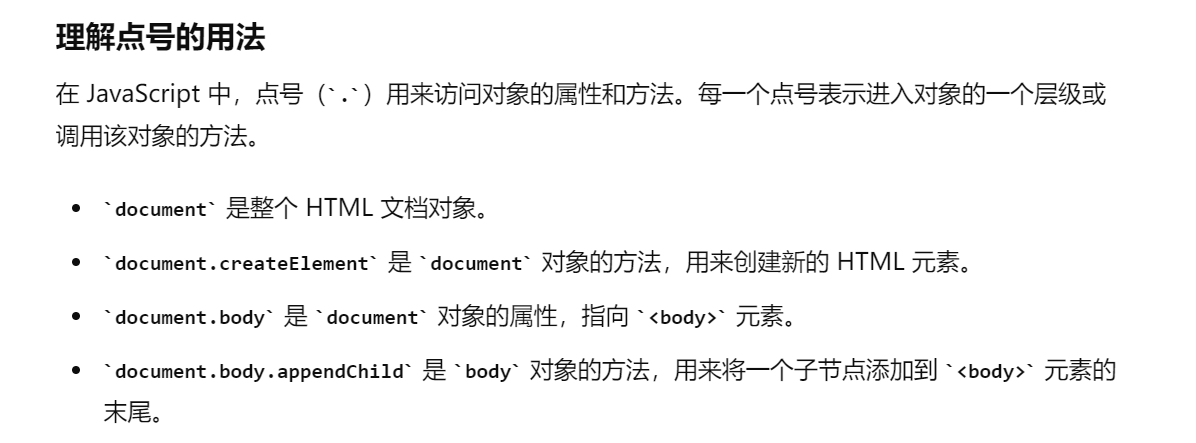
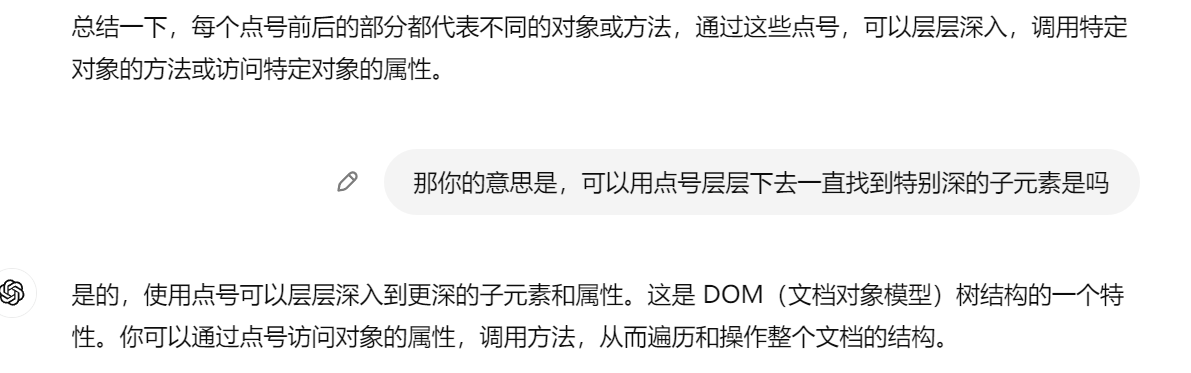
最重要的内容：理解js点号“.”的用法(.可以访问属性比如img的src属性)



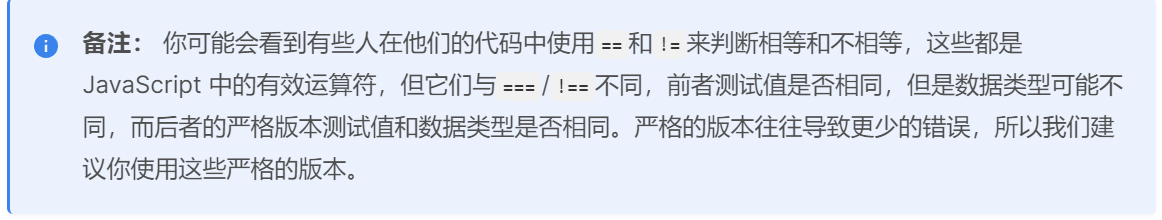


1. 变量用var还是let

一般都用let因为更像其他语言以及更加现代，具体区别可以问chatgpt



1. 关于=== == 与 !== !=



等号多的哪个版本更为严格，所以尽量都用这些等号多的版本

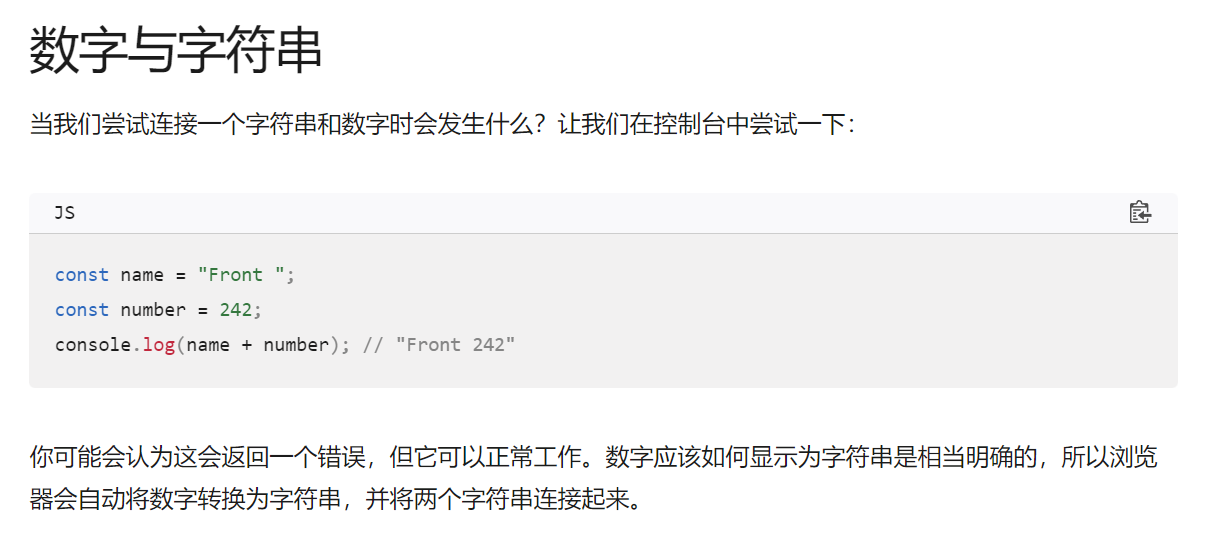
1. js字符串的嵌入（可以用${}，也可以用“+”，只是一个易读一个没那么易读，一个可以放表达式另一个不行）



这样子可以让一个字符串显示在另一个字符串的里面，还可以放表达式



1. js也存在自动类型转换



因为一个数字被迫跟一个字符串一起输出，那么为了不出错，会把数字给她变成字符串然后一起输出，这跟c++的是差不多的



Number() & String()可以把东西转换成数字和字符串

1. 字符串与数组互化

Split（），在字符串中找到括号内的元素并按该元素分割，装入数组

Join（），在数组的每一个元素之间加入括号内元素，连接成字符串（不写默认是逗号）



Tostring（）：将数组变成字符串，一步到位但是只能用“，”当做分割符

1. 在数组开头、末尾，增加或删除元素

[push()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/push) 和 [pop()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/pop)，都在末尾操作，分别是增加、删除

[unshift()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/unshift) 和 [shift()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/shift) 都在开头操作，分别是增加、删除

这两组用法完全相同，只是作用的地方不同

1. 匿名函数

说的通俗易懂就是没名字的函数但有用，几种写法如下

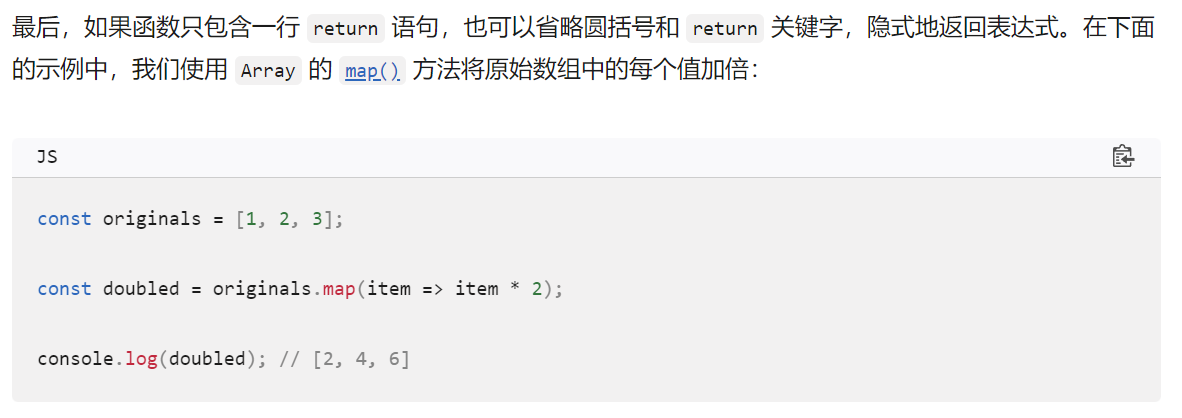
1. 用括号包起来 2.不用括号包起来，直接写到要用的地方

3.箭头函数写法，直接代替function就行，如果确定只有一个形参，可以不要形参边上的括号



注意，箭头函数就只有一行return的时候可以简化成如下形式



直接给出想要的值就可以，不要随意使用花括号，花括号代表代码块，需要返回return，如果要返回promise就会出错

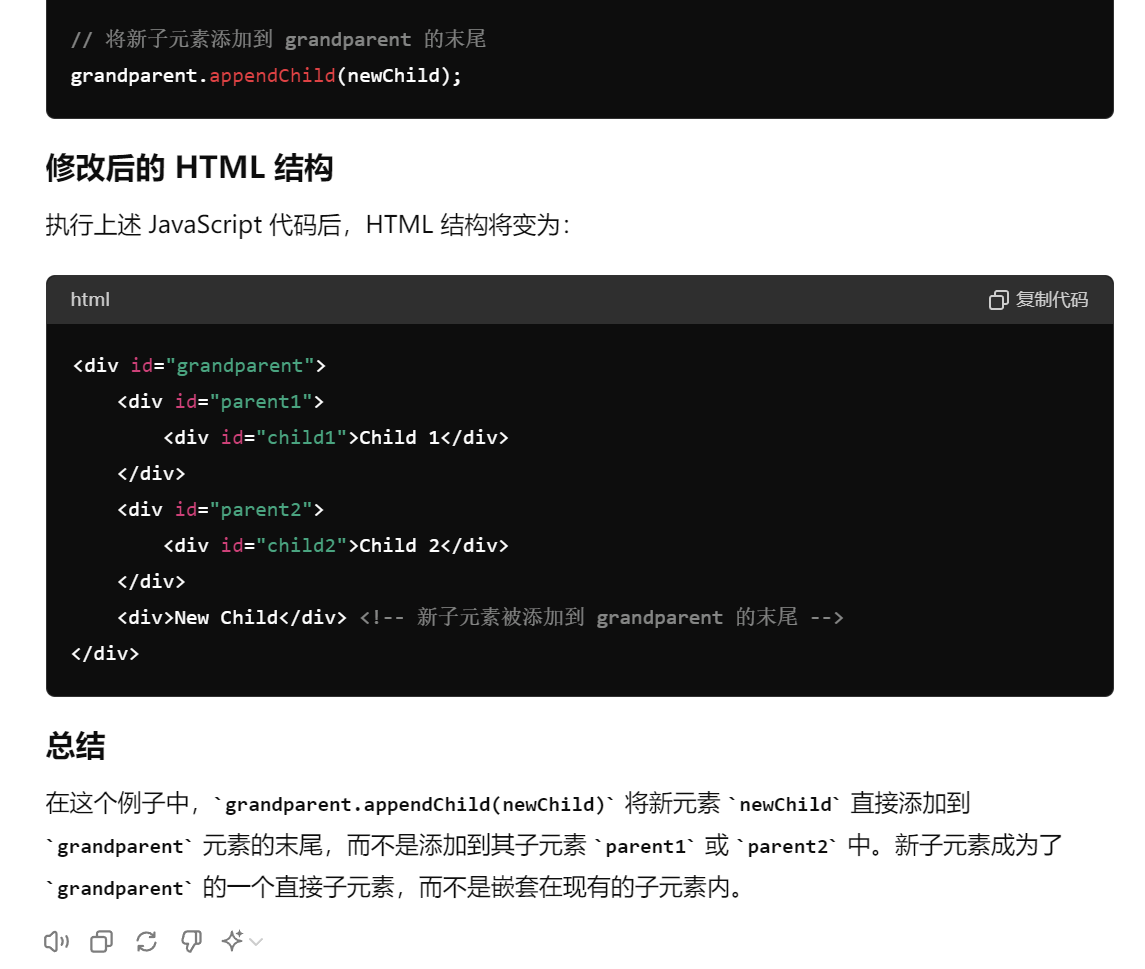


就像这样直接就可以输出想要的东西

1. node.appendchild



这么搞的话，会增加一个p元素，加到body的末尾作为其直接子元素



一定是直接子元素！！！

九、document.getElementById()，document.querySelector（），document.createElement

1.document.getElementById()就是找到这个id的元素用的



2. document.querySelector（）

一定是找到第一个标签e.g. document.querySelector（”p”）一定是去html里面找第一个<p>

3. document.querySelectorAll（）

跟上一个不一样，这个是找到所有的e.g. document.querySelectorAll（”p”）一定是去html里面找所有的<p>

4. document.createElement（）

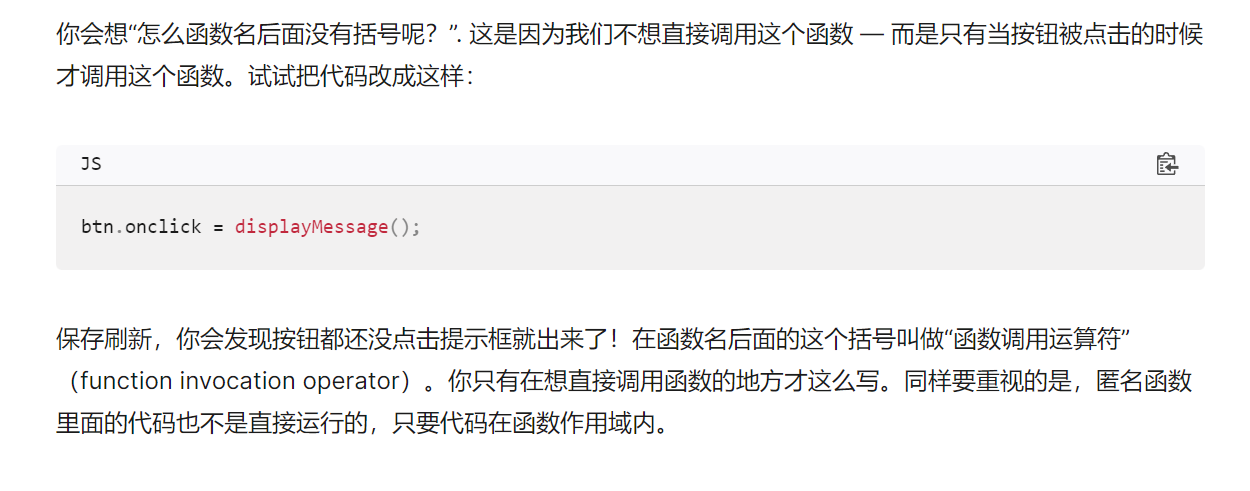
就是去创建一个HTML标签e.g. document.createElement("div")，这个是在html标签里面创建一个直接子元素<div>，但创建在哪里不知道，还要搭配上appendChild（）

e.g. let p = document.createElement("p");

document.body.appendChild(p);

这个东西就是创建一个<p>然后加到<html>的直接子元素<body>里面，让这个新的<p>成为<body>的最后一个直接子元素(但是这个是node的实例方法,node.appendChild())

十、不要莫名其妙的在调用函数时给它加括号（js的函数形参没有let，直接给名字就行）



因为括号又叫函数调用运算符，直接就会调用这个函数，前面的就失效了

十一、

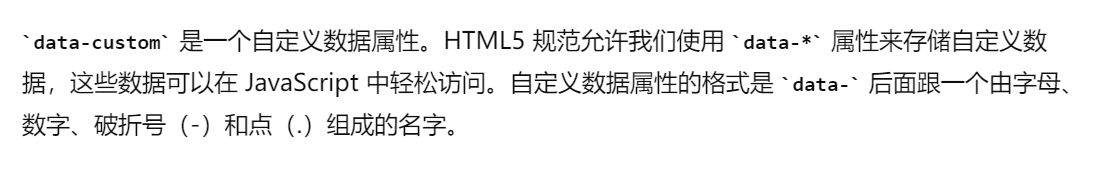
getAttribute & setAttribute







Html 5允许自定义属性名，具体格式如下面截图所示



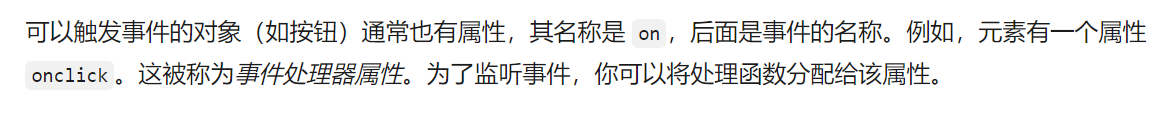
十二、事件监听器  
 1.最常用的就是[addEventListener](https://developer.mozilla.org/zh-CN/docs/Web/API/EventTarget/addEventListener)

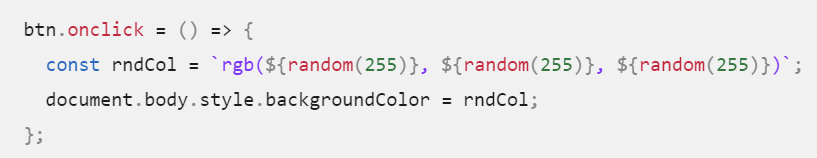
e.g. 当btn所绑定的html元素没点击（click）时，执行后面这个函数(changeBackground)



2.事件处理属性，多用“on”

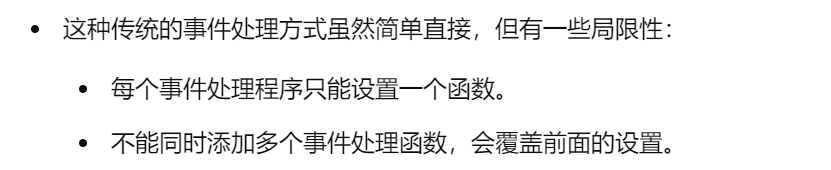
e.g.



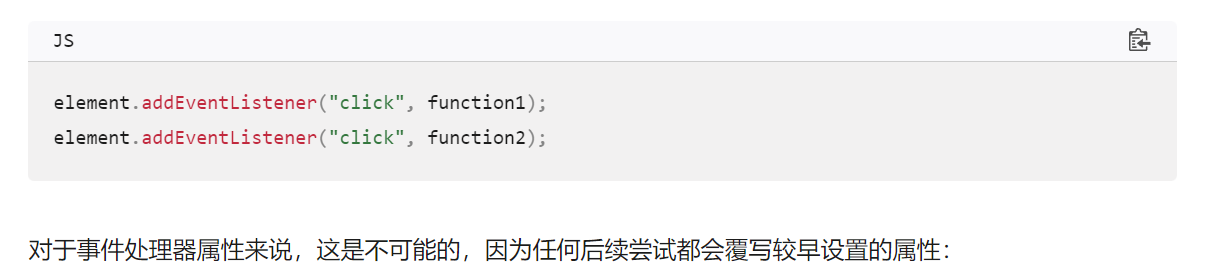




这两种方法都可以，一个是直接箭头函数，一个是比较规矩的写法，但是这种方法有点傻、



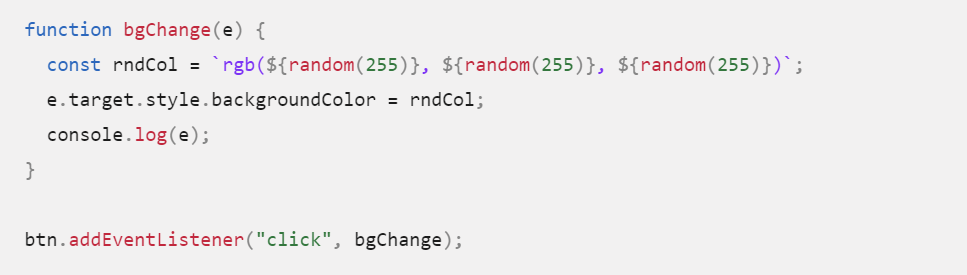
这种传统的方法，如果同时设置两个相同的事件，那么后面那个事件的操作会覆盖前面那个事件的操作



3.内联事件处理器（了解即可，不要使用！！！）

十三、事件对象（e,evt,event）

这种东西就是方便事件监听器在调用函数的时候，方便找到那个使用事件监听器的元素（或其他），然后对其进行一定的操作



如图所示，e就是一个事件对象，e.target就会定位到btn然后对其进行修改背景颜色的操作

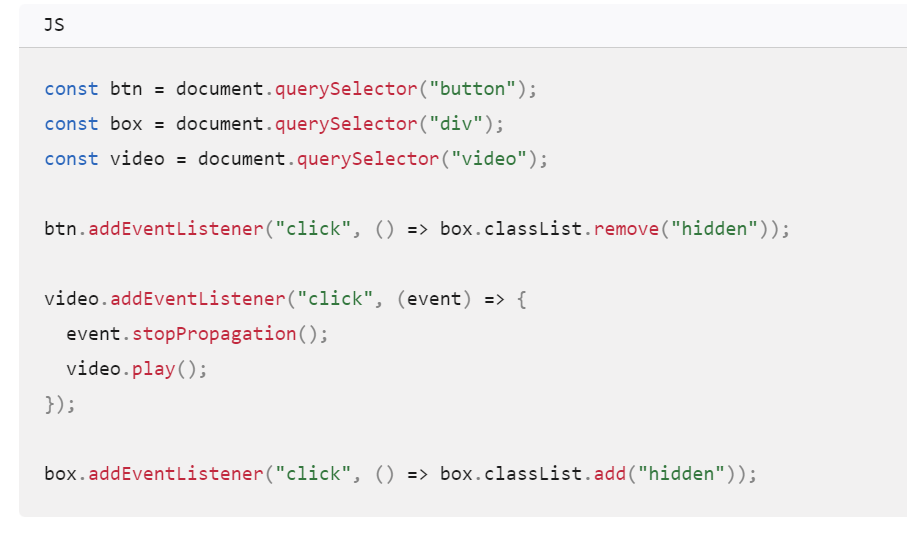
e.preventDefault()（不满足要求时，不提交表单） e. stopPropagation()

十四、事件冒泡、事件捕获

Js的事件对象会有点神奇，如果刚好在最内层，那么哪怕js的事件监视器是从父级元素开始写下来的，那么最先输出的也会是最内层的结果，示例如下：

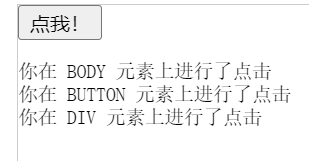
可见，哪怕是个最顶端的父级元素，也要遵守冒泡这一规则，但是可以用stopPropagation()来阻止事件冒泡，



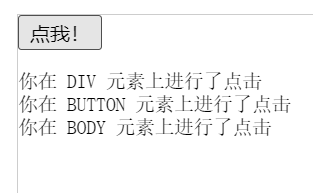
这样之后，视频就不会在被点击，开始播放后就隐藏起来，因为被阻止了

事件捕获：采用完全相反的顺序执行代码，代码啥顺序他就啥顺序，不会倒序



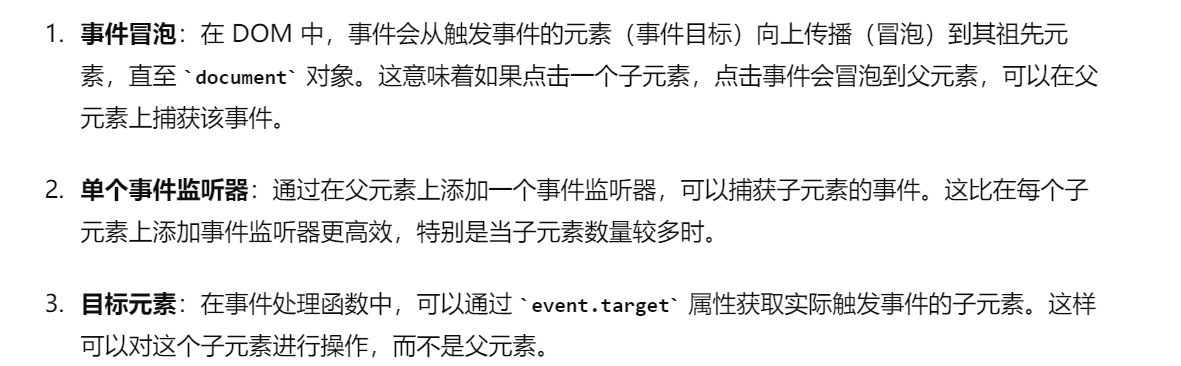
注意区分



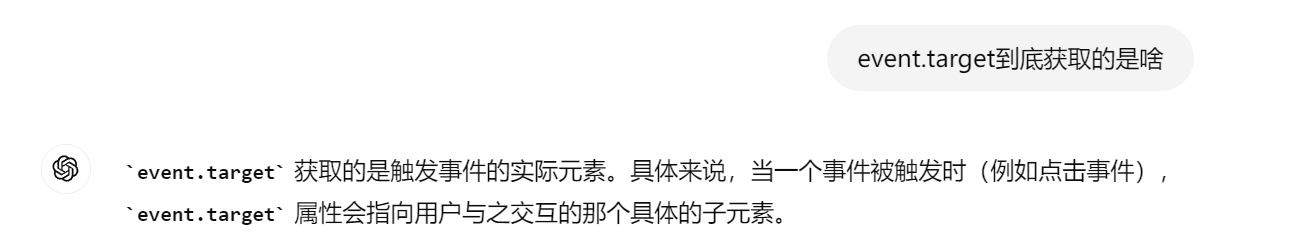


可以假想一个偷屁股的，从最后面开始扫，扫到capture: true就被捕获，然后这个元素就会按正常顺序出来

事件冒泡的理解：



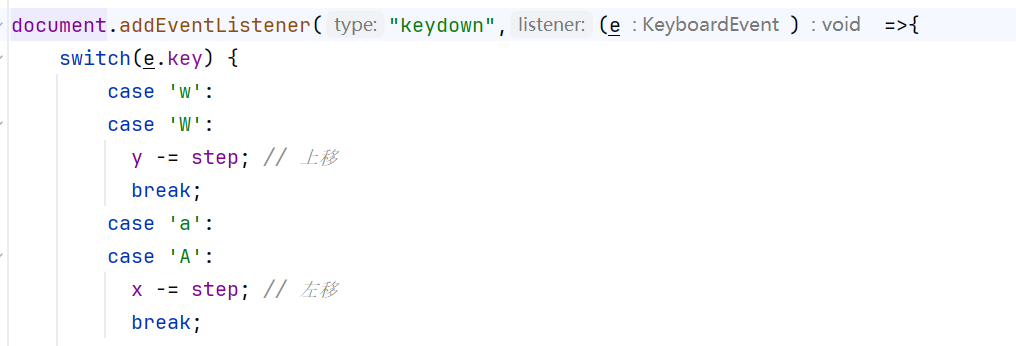
e.target的理解



十五、获取键盘输入的值

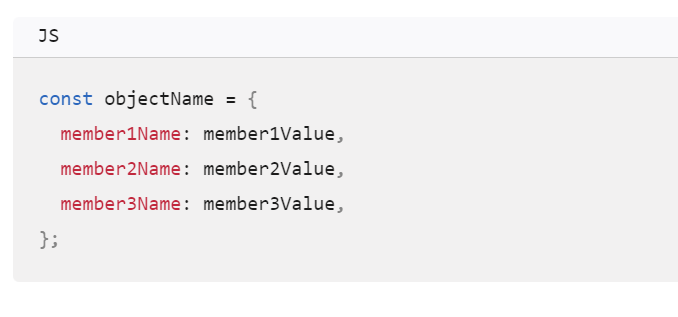


该代码是通过e.key来获取的，当然，获取的是该文本框内键盘输入的，如果要获取其他地方的，那么就是用其他地方的标签然后设置事件监听器 例如



直接获取整个html文档上的键盘输入

十六、js对象



但是当成员是函数的时候，就可以简单一点

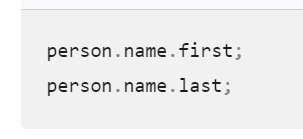


这是创建js对象的基础语法，也可以实现嵌套



访问方法：

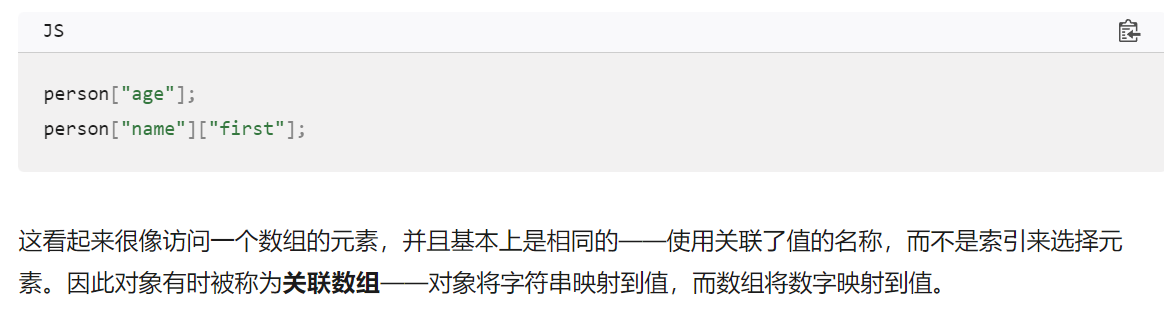
1.通过点运算符就能访问，第二个对象创建稍有不同



2.函数成员的访问如下

一定要有括号

3.还可以用类似数组的方式访问



Js对象的简写属性：如果键和值的值一样，那么可以直接写个键，就不用写值了

十七、js对象的成员不可用点运算符访问的五个情况（动态属性名，即保存在变量中）\*





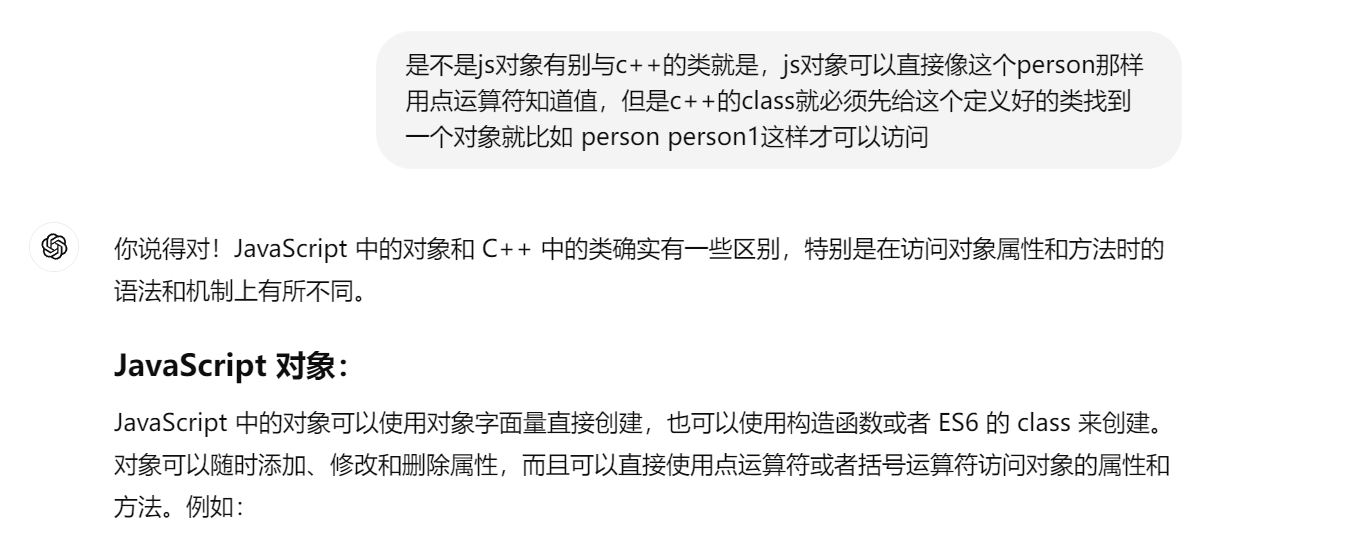




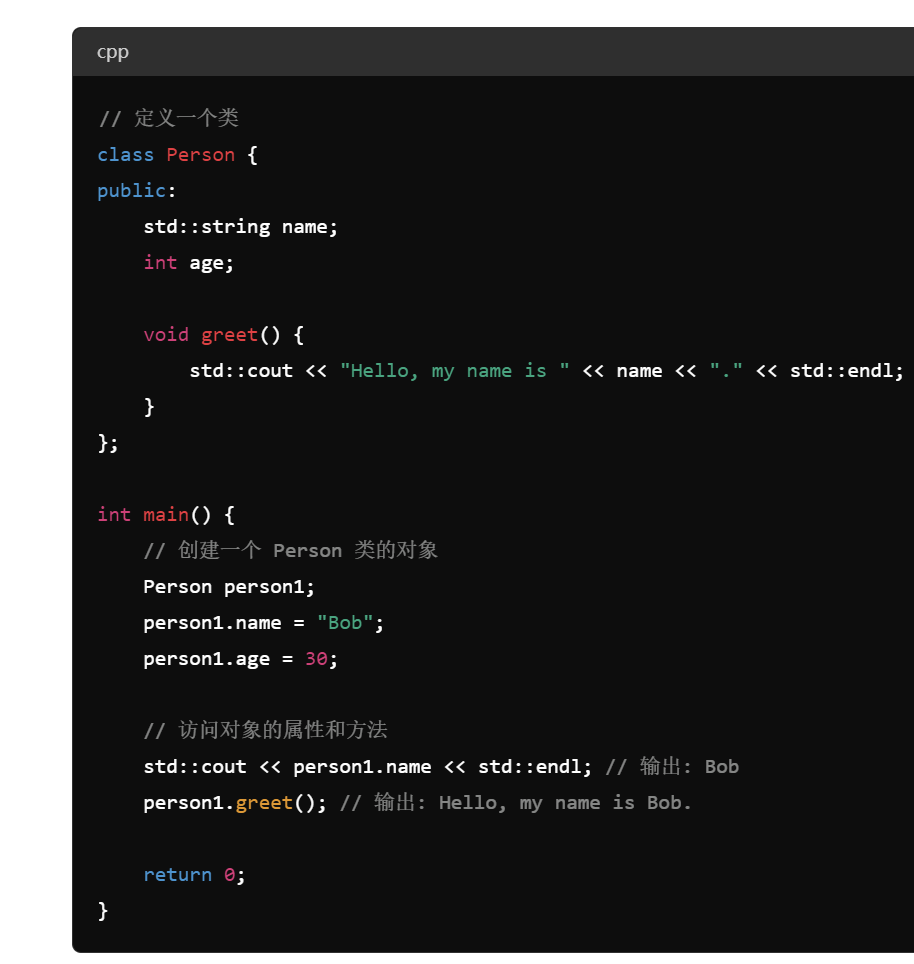


只有括号运算符可以做到，使用变量来访问js对象的内容，点运算符不行

十八、js对象对比c++类



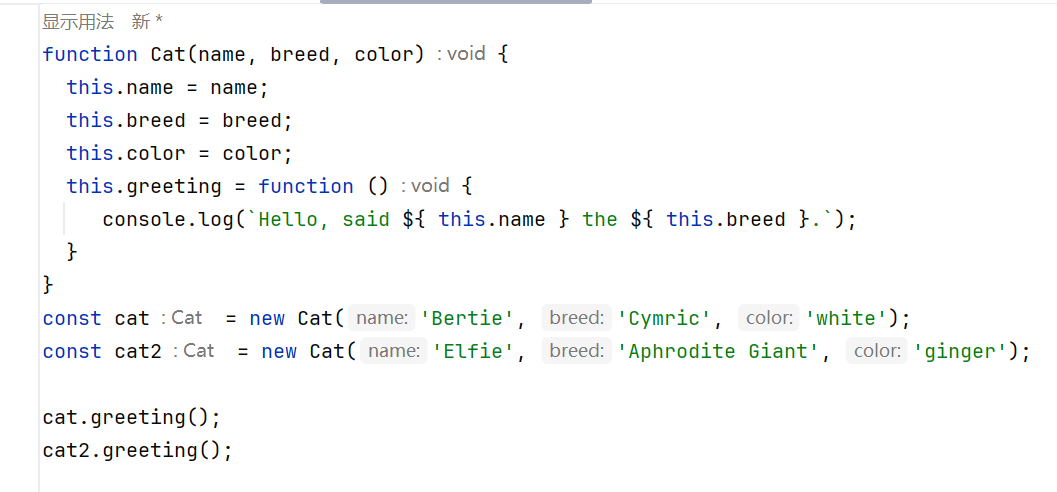




但是c++的类就必须先创建出这个类的对象才能访问这个类的成员，但是js就可以直接访问，直接用这个对象名

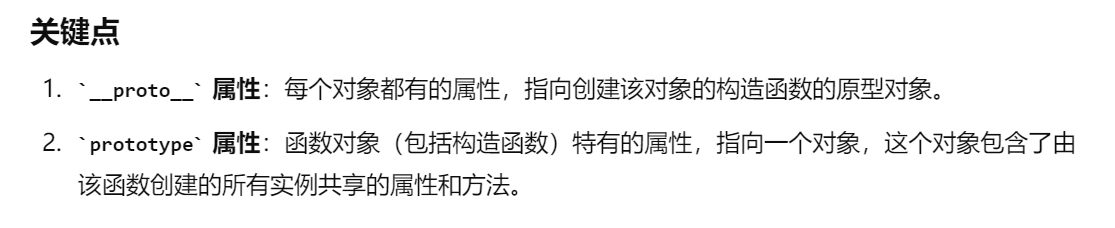
十九、当js对象要调用自己的某个元素时要用this

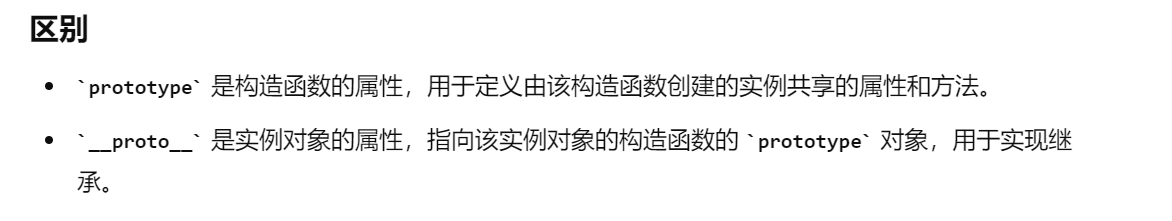
例子如下：



必须这个样子才是正确的调用自己。

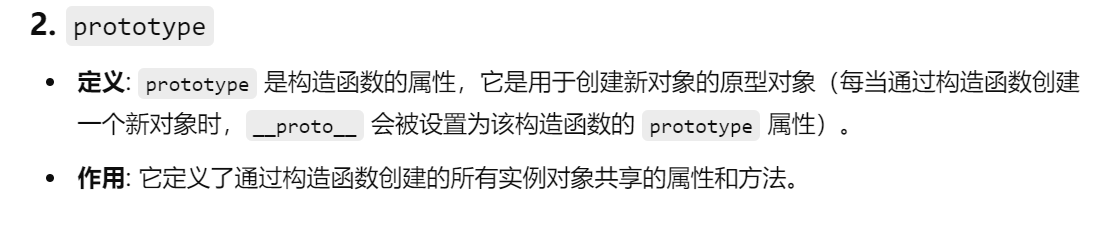
二十、\_\_proto\_\_和Prototype的区分（原型一定是一个对象）





任何js对象都是由object.prototype这个原型编出来的





1. alice.\_\_proto\_\_ === Person.prototype; 因为Prototype的定义如上图，所以由构造函数生成的js对象

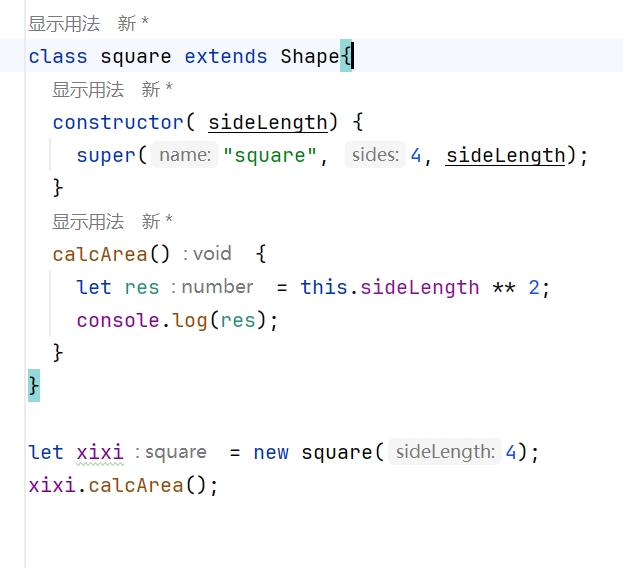
2.Person.prototype.\_\_proto\_\_ === Object.prototype;因为Person.prototype是创建新对象的原型对象，所以他是一个js对象，所以就有\_\_proto\_\_属性，又因为构造函数在往上就是Object,所以会跟Object.prototype相等

3.Person.prototype.constructor === Person，可以把这个constructor理解成建造者，那么person.prototype这个构造新对象的原型对象的建造者就是Person这个构造函数

4.alice.constuctor === Person，解释跟3一样

5.Object.prototype.\_\_proto\_\_ === null，因为Object以及是原型链的顶层了，所以再往上找一定是空

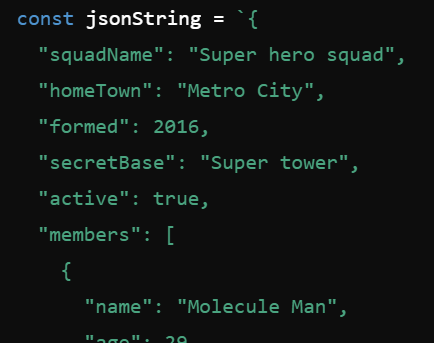
二十一、js派生类实例



如图，定义了一个类，一个派生类（extends），因为派生类里面用的sideLength是基类里面的，所以就不需要再次声明，所以在创造派生类的构造函数的时候，我们直接使用这个sideLength，然后一定要用这个this指针访问它，不然会出错。

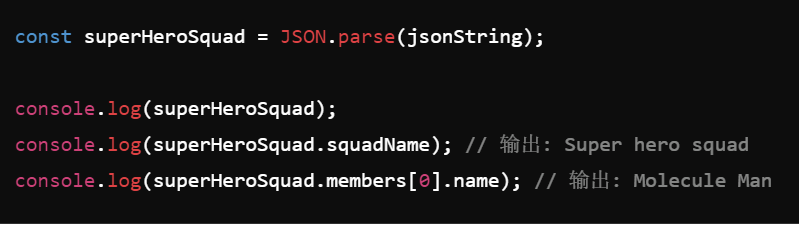
二十二、JSON是个字符串

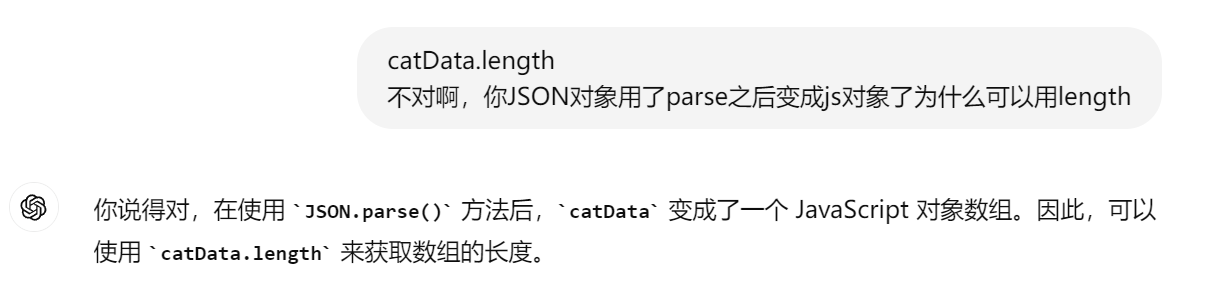
所以，在把JSON解析给一个变量的时候一定要加引号，如下图所示



通过JSON.parse()可以把JSON字符串解析为js对象,这个时候的js对象是一个数组

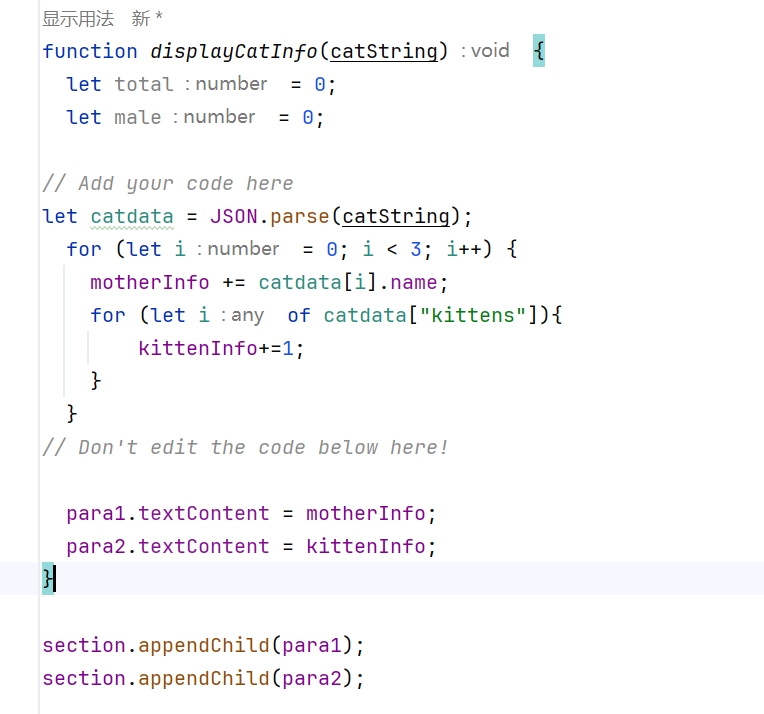
可以用.length来返回长度





二十三、fetch（）异步讲解

这玩意儿是一个异步操作，也就是说无论放在那里，他都会被放在最后执行，所以有些语句不能放在最后，以免执行完fetch（）之后执行不到最后的那几句话了，下图就是个很好的例子





二十四、快速交换值

、

利用数组解构，类似于python的那种快速交换

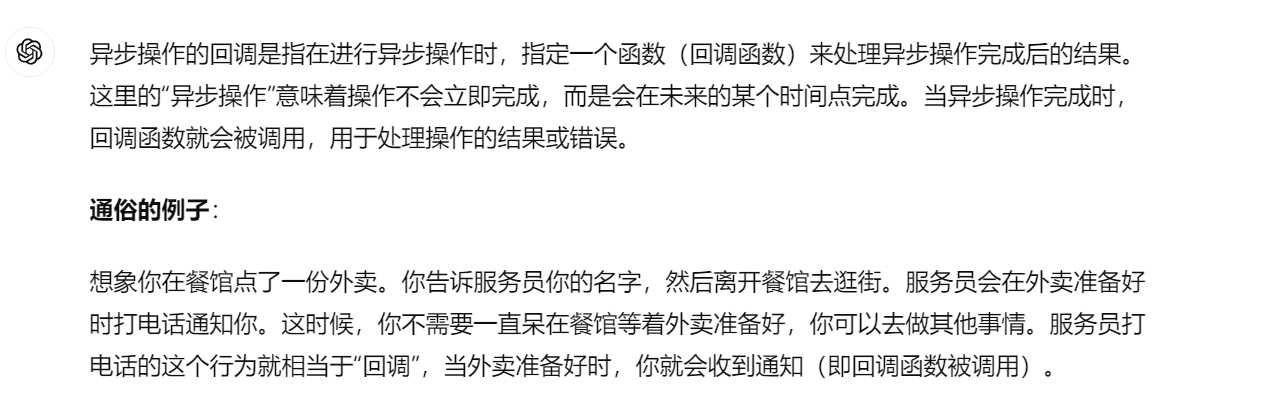
二十五、XMLHttpRequest（已经被fetch取代）

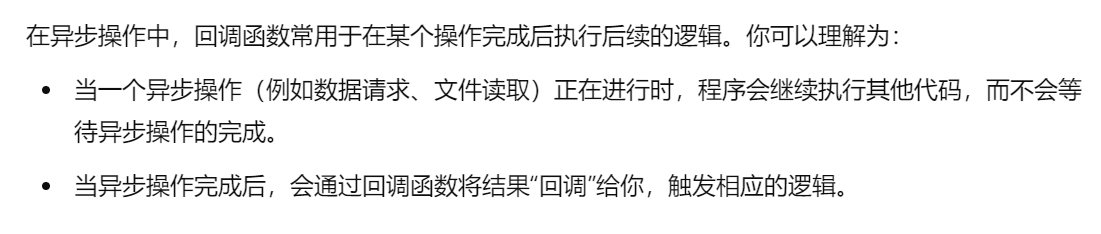
用于处理异步http请求，因为有些时候js代码执行某些东西的时候需要较长的时间，这就会导致相当长一段时间内，用户无法进行任何操作，类似于卡死，所以就要用到这个API。

创建XML对象



二十六、回调的理解





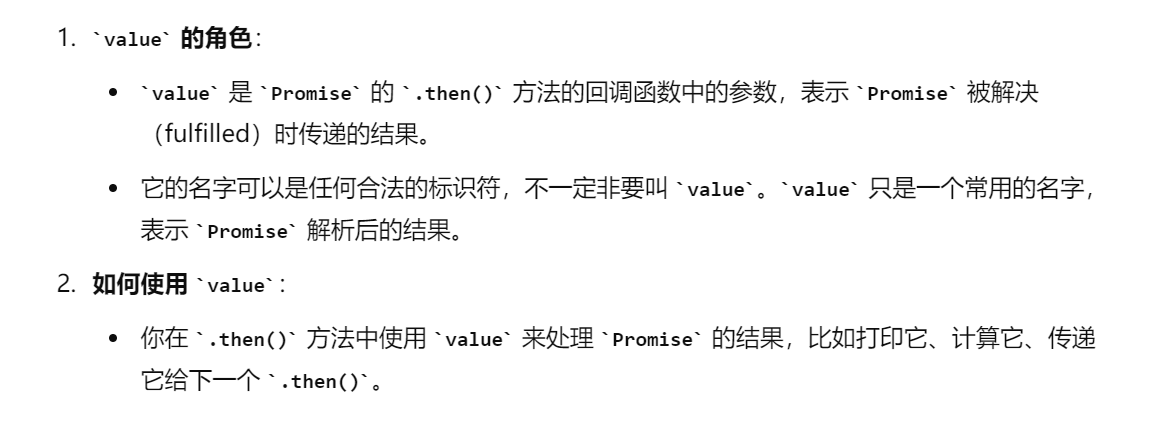
二十七、Promise的理解（也是一个异步操作）

Promise对象表示异步操作的结果值，就比如如下图的代码



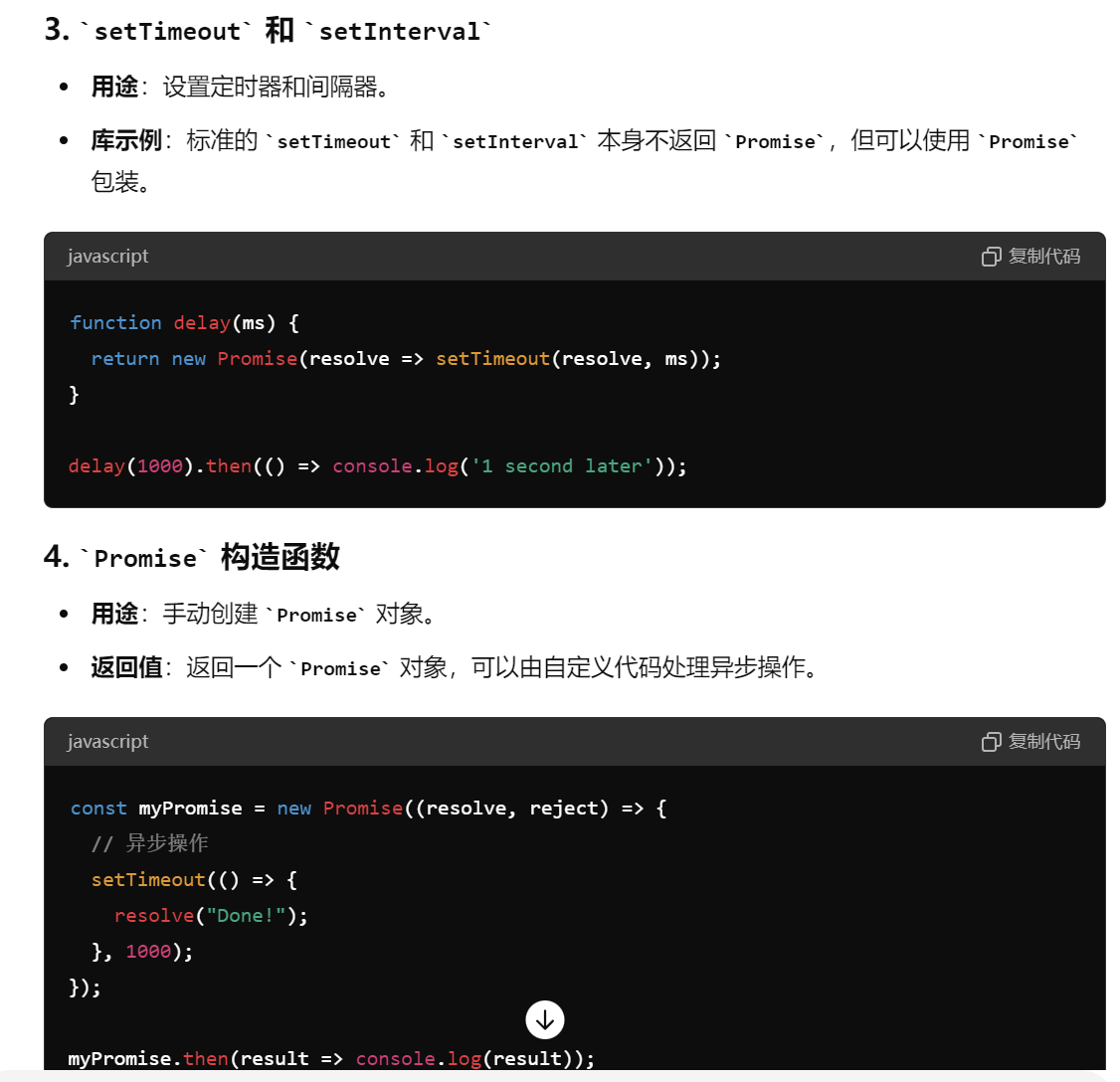
他把这个promise对象状态直接设置成已完成（就是resolve），那么这个promise对象就对保存这个操作的结果值，也就是“start”，那么在进行.then操作的时候这个“start”就会被拿出来。

注意.then里面用的都是异步操作的结果值，也就是说value这个词可以换成任何合法的词

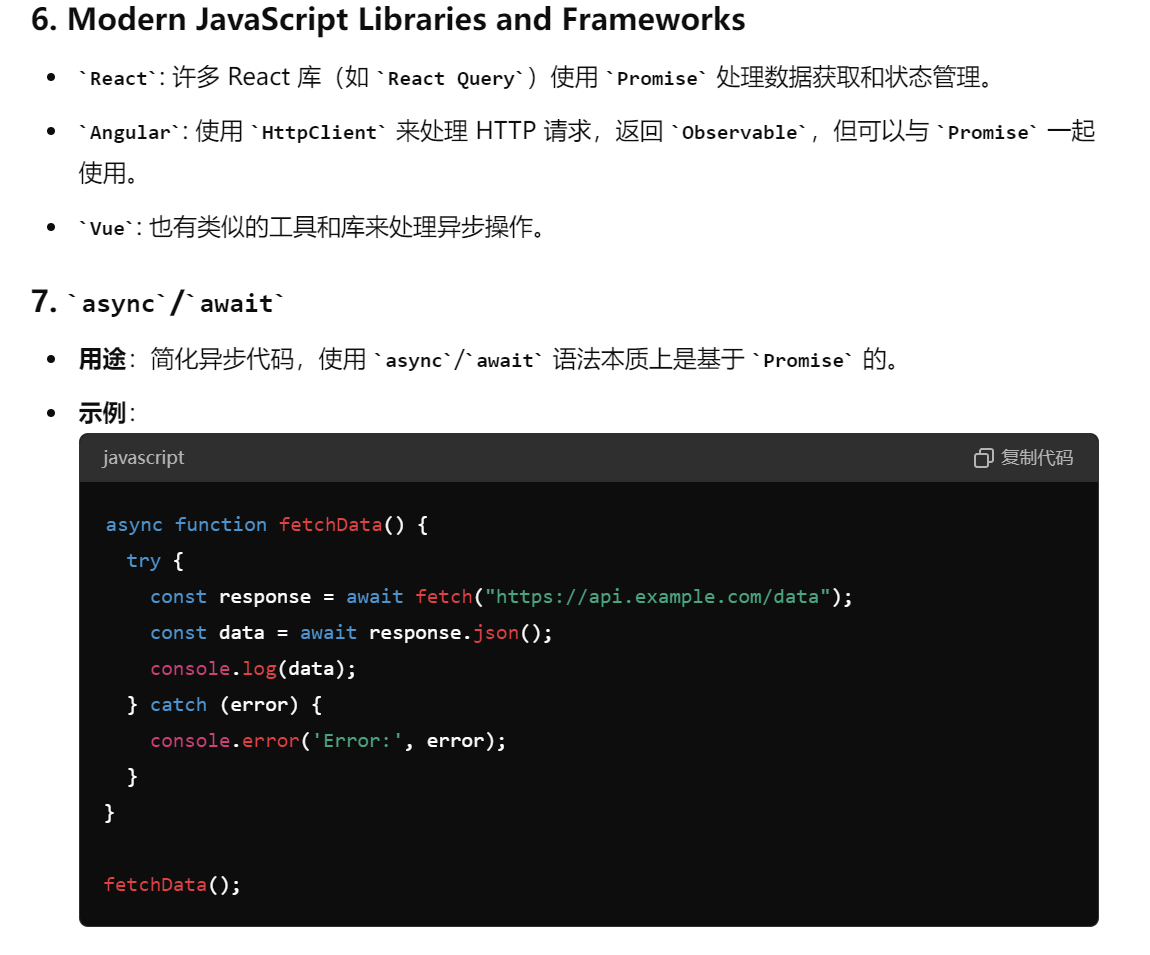


以下是会创建promise对象的几个API

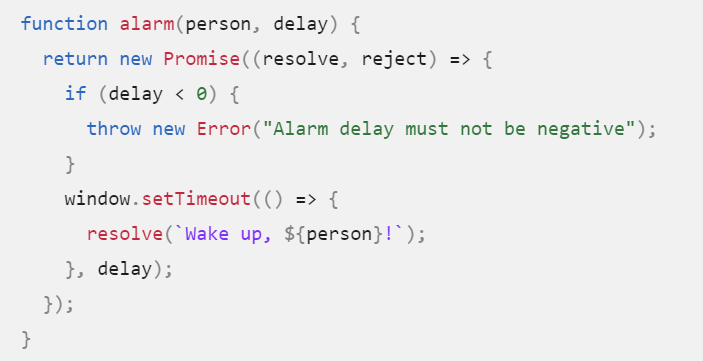
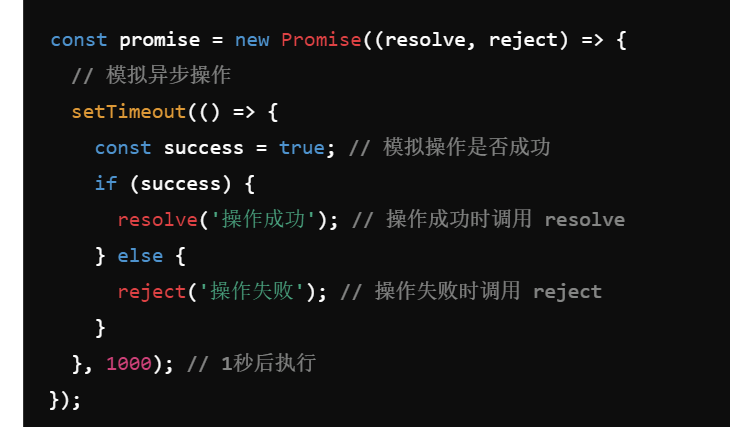








二十八、promise构造函数



例子如图，resolve跟reject互斥，条件成功就构造resolve，括号内的就是这个promise解析后的值，反之构造reject

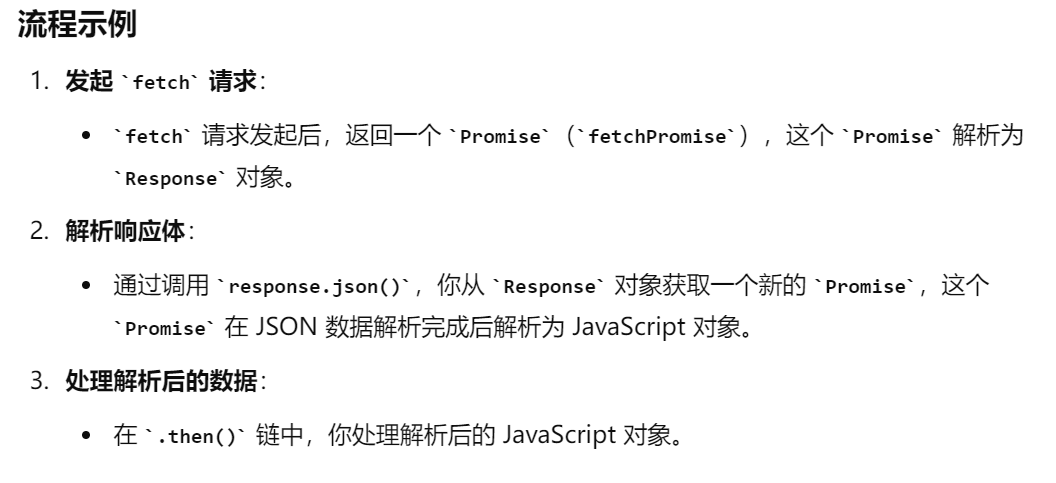
右边这个例子就是百分之百构造成功的例子，因为没有语句可能导致promise失败

二十九、.json()方法

将对象内容转为js对象（fetch()不会自动把JSON转换成js，必须要用这个）以下是一个例子



抓取了一个json到fetchPromise然后通过.then()使用了.json()方法



三十、promise.all promise.any

一个相当于逻辑与，一个相当于逻辑或

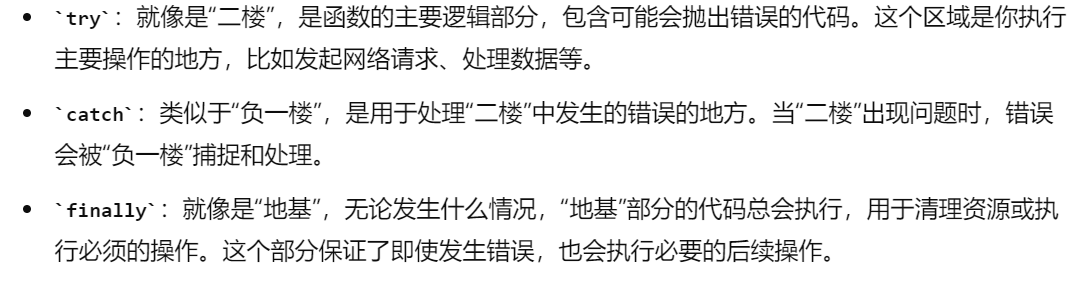
Promise.all只会在所有传入的Promise数组都成功时执行.then

Promise.any只要有一个成功都会执行.then，但是只会执行第一个成功的

三十一、async和await

Async这个关键字用于写异步函数（就是在普通函数的function关键字前面加上async，箭头函数用的时候是加async在括号前面）这个函数里面有try,catch,finally :

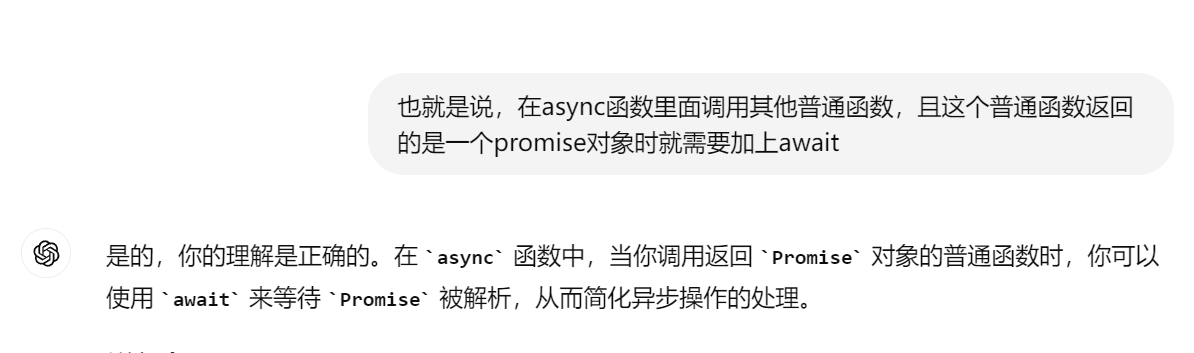




Finally是可选不是必选，try是主要部分，必须有且包括函数的功能

注意，不管你最后让这个异步函数返回什么东西，他都会变成promise，所以一定要在异步函数使用之后，把那个promise变成js对象在进行下面的操作

Await使用的地方



三十二、worker的详细解释

就是一个线程，可以与主线程隔绝防止干扰，他们主要有message事件互通，通过事件对象的.data属性获得彼此传过来的数据，截图如下

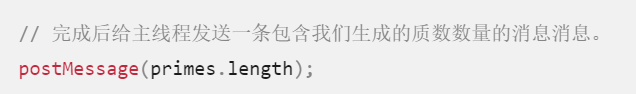


主线程调用worker.postMessage向worker发送一个JSON



Worker用同样的事件接收这个JSON并且用.data属性读取这个JSON

Worker的最后用



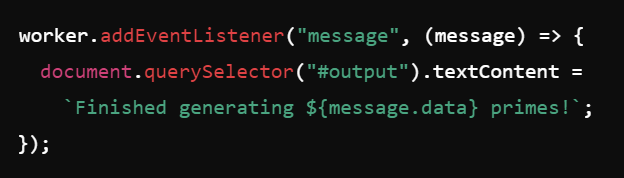
给主线程返回一条消息



主线程依旧用message这个事件来接受以及用.data使用

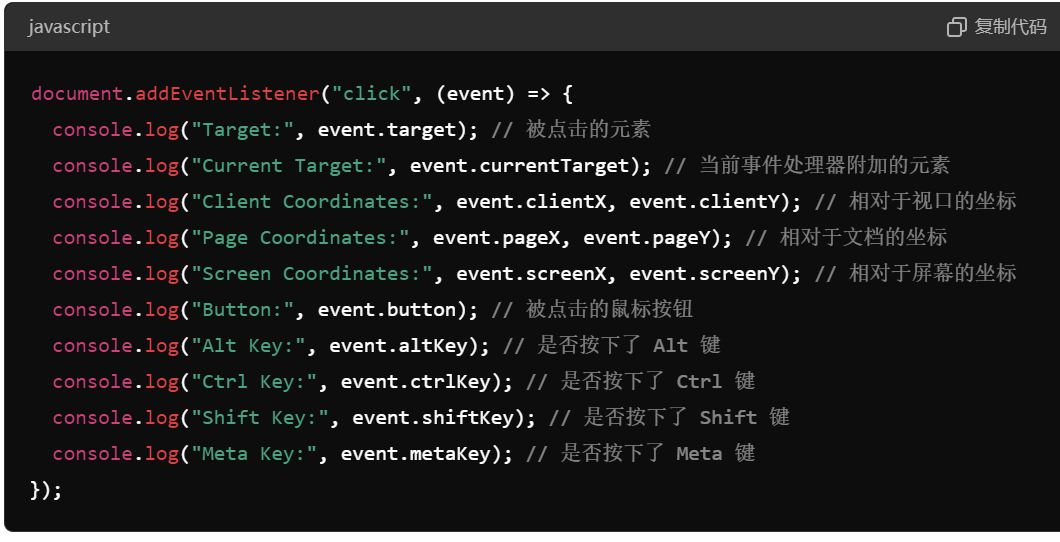
三十三、web workers常用的事件

1.“message”事件，用于在worker在向主线程发送消息的时候触发



三十四、各种事件的常用属性

1.”click”事件



2.”message”事件



三十五、操作文档

1. 创建并放置新的节点（前面还有几个最基本的就是那些query）

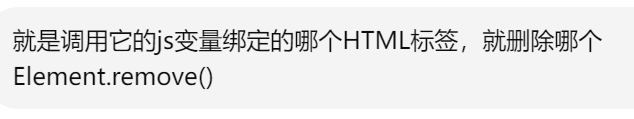
1.[Document.getElementById()](https://developer.mozilla.org/zh-CN/docs/Web/API/Document/getElementById) 选中那个id元素

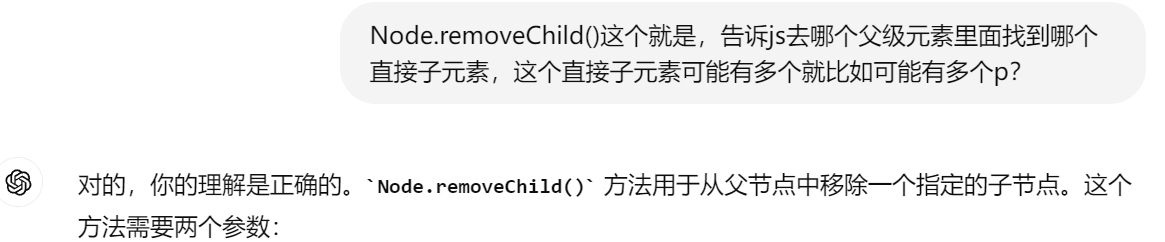
2.[Document.getElementsByTagName()](https://developer.mozilla.org/zh-CN/docs/Web/API/Document/getElementsByTagName) 返回页面中包含的所有已知类型元素的数组。（注意一定是所有，按照从上之下的顺序一次排成数组）

②移动和删除元素

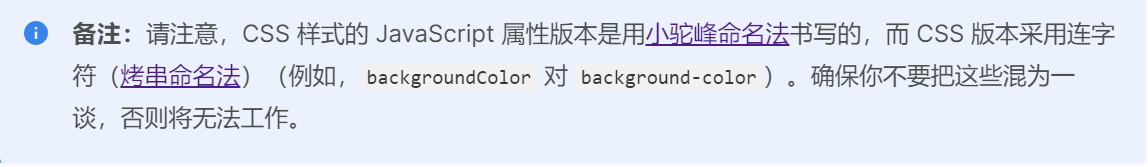
1. Node.cloneNode(deep)：这个方法会深度克隆节点，要给一个deep值，deep是true则会克隆所有后代节点，子节点，如果是false就只会克隆这个节点本身
2. appendChild()：会把括号里面的对象加到前面的对象的底部，作为一个直接子元素 
3. [Element.remove()](https://developer.mozilla.org/zh-CN/docs/Web/API/Element/remove)

[Node.removeChild()](https://developer.mozilla.org/zh-CN/docs/Web/API/Node/removeChild) ：这两个都是删除，但是有一点区别





1. 操作css样式



直接.style.(css样式名字比如color)，或者用setattribute（还有getattribute）

三十六、

1. .json



1. .text



1. .formData

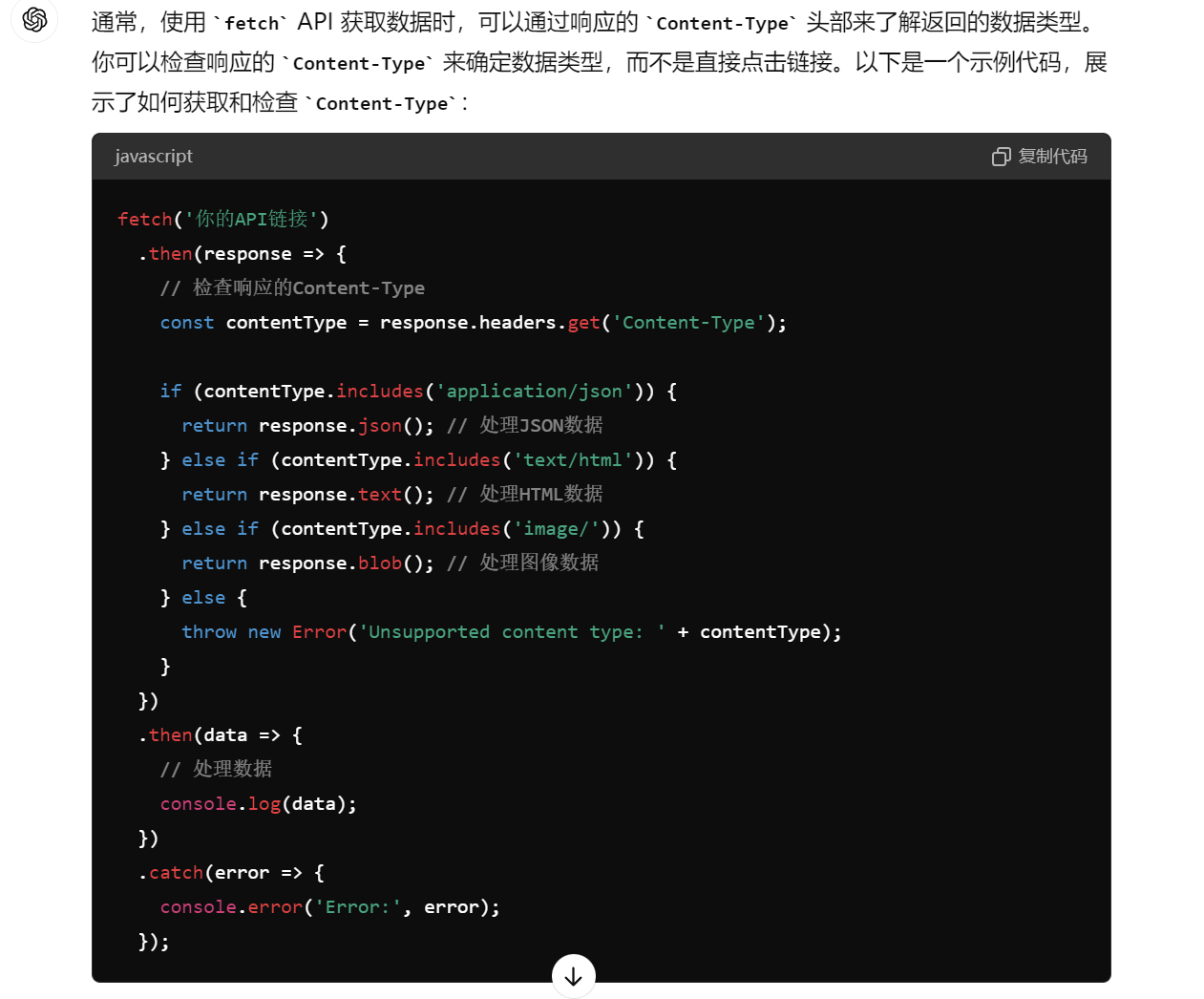


1. .arrayBuffer



1. .blob (图像或者音频)





如果不知道可能响应出来什么类型，可以直接使用Content-Type这个东西



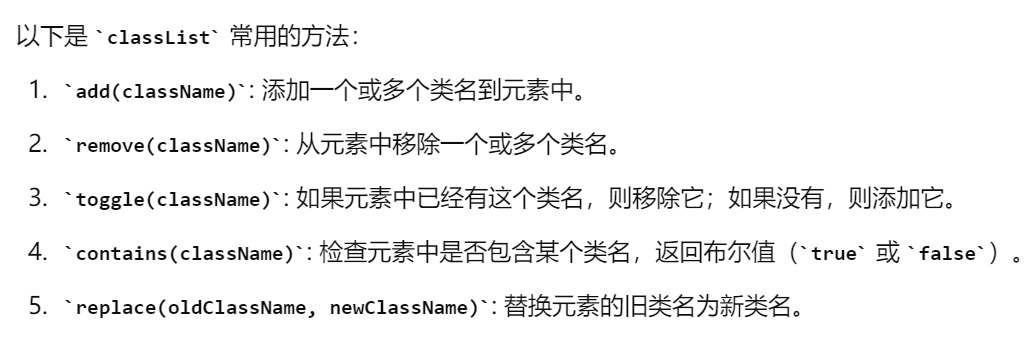
三十七、setInterval和clearInterval

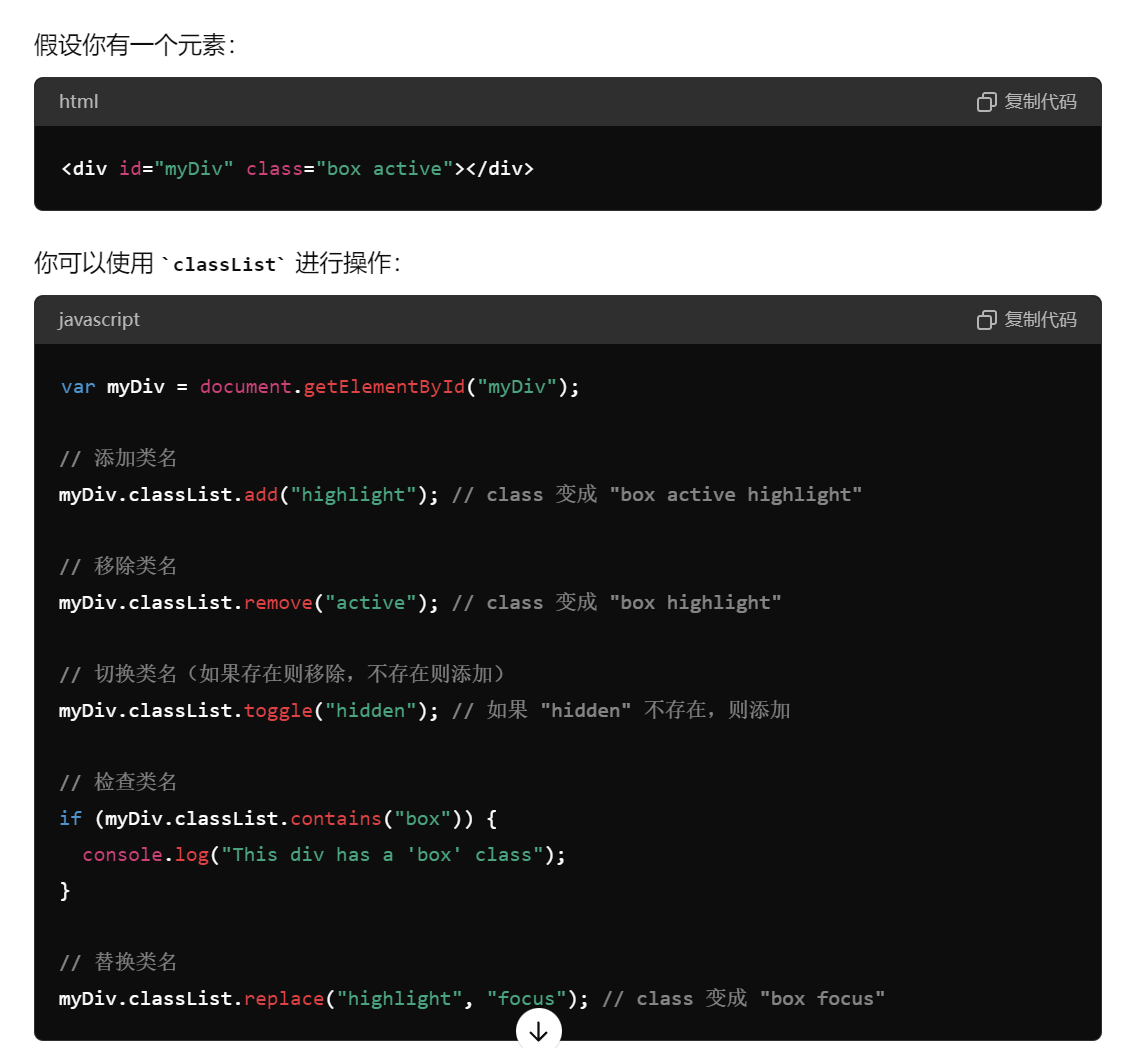
前者是设置定时器，后者是清除前者设置的定时器



三十八、classList（一个DOM属性）

可以直接操控class属性，切换（toggle）指的是切换有无，有就删，没有就添加





三十九、客户端存储

Cookie（太老了，设计安全性问题）

[Web Storage （包含[sessionStorage](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/sessionStorage" \o "sessionStorage) 和 [localStorage](https://developer.mozilla.org/zh-CN/docs/Web/API/Window/localStorage)）和 IndexedDB](https://developer.mozilla.org/zh-CN/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage#%E6%96%B0%E6%B5%81%E6%B4%BE%EF%BC%9Aweb_storage_%E5%92%8C_indexeddb)（当前）

Cache API（未来趋势，很多新浏览器支持）

1. Web Storage

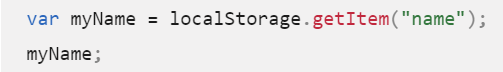
local[Storage.setItem()](https://developer.mozilla.org/zh-CN/docs/Web/API/Storage/setItem) local[Storage.getItem()](https://developer.mozilla.org/zh-CN/docs/Web/API/Storage/getItem)  local[Storage.removeItem()](https://developer.mozilla.org/zh-CN/docs/Web/API/Storage/removeItem)

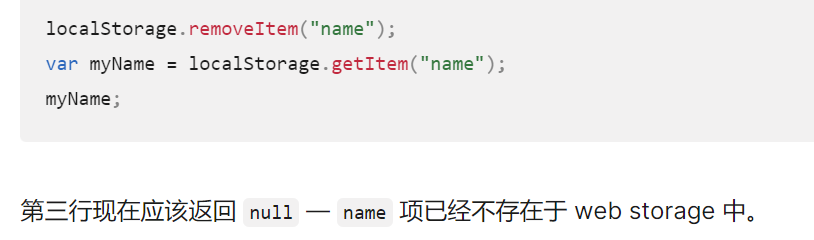
一个是设置数据项，一个是读取数据项，一个是删除数据项

如果是sessionStorage API只需要把前面的localStorage换成sessionStorage

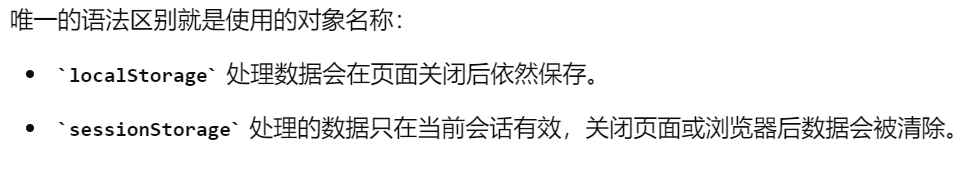
e.g.







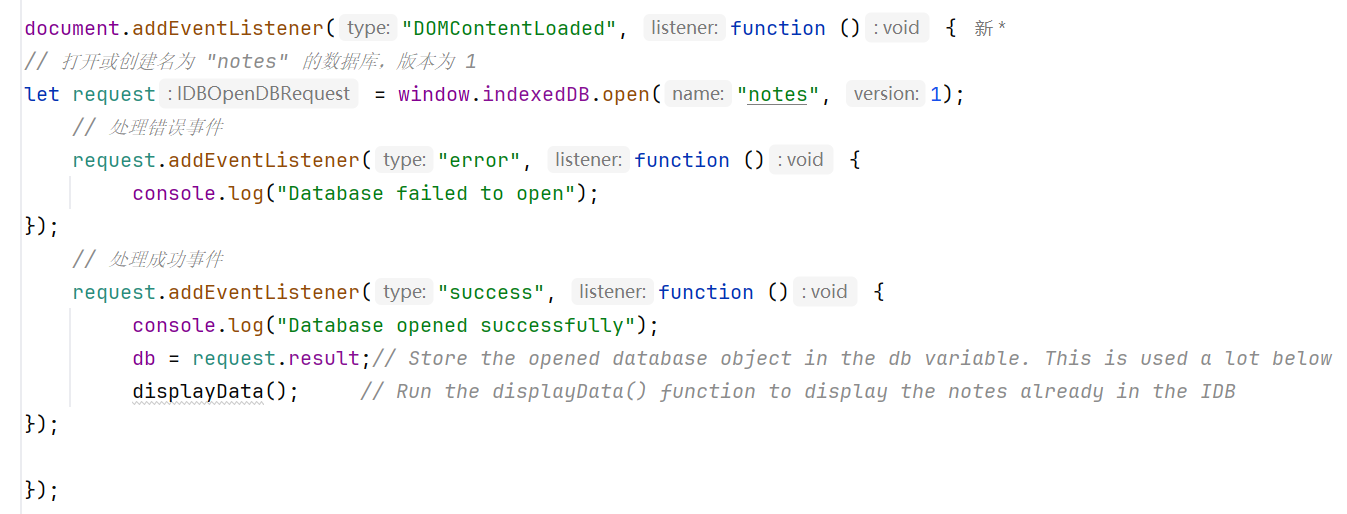
Local跟session的区别:



1. IndexedDB
   1. 创建数据库



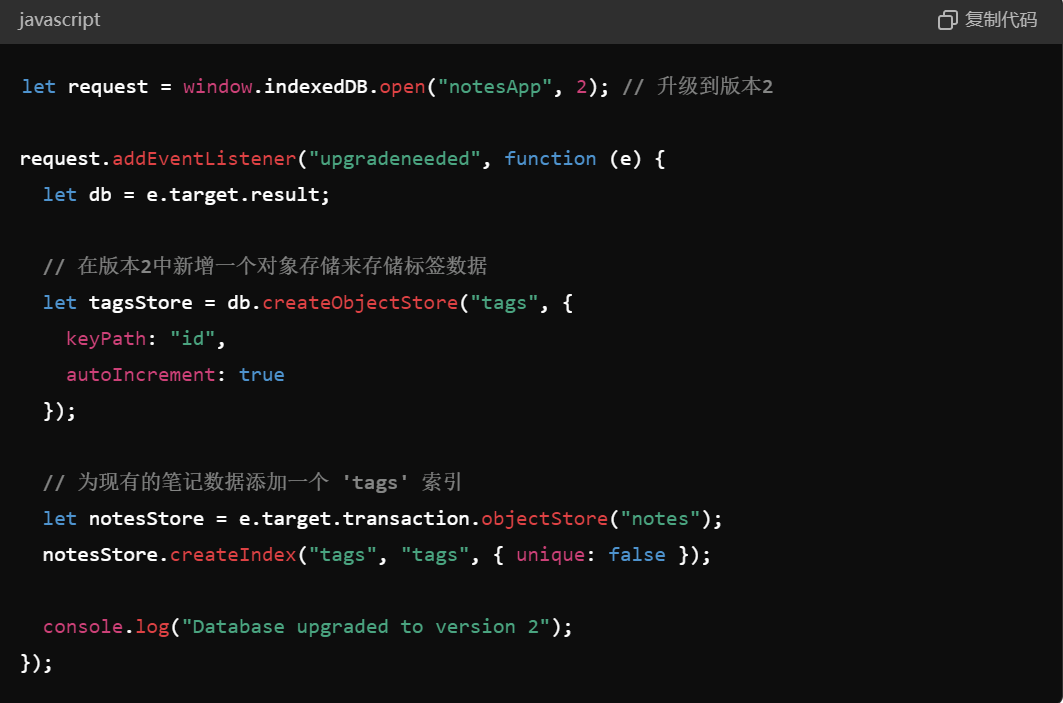
创建了数据库，名字叫notes，版本号是1，装在request变量里



网页被加载的时候会执行这些代码，确定数据库是否创建成功并且把他请

求的结果装到db这个变量里。（error & success）

* 1. 升级数据库（类似于把当前数据库的内容整合好，变得更大）



（为现有笔记创建索引就需要创建当前数据库的对象存储

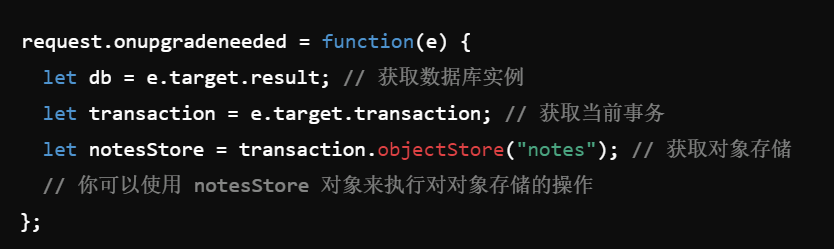
Let notesStore.createIndex("tags", "tags", { unique: false });）

如果要为新的笔记创建，直接用tagStore这个对象存储

let request = window.indexedDB.open("notesApp", 2); 这句话时触发升级事件的关键，因为要打开版本2，但是当前版本最高就是1，所以会触发升级事件

createObjectStore: 负责创建对象存储，存储每一个原数据库的内容，id是主键，其中autoIncrement代表是否允许主键自动递增（如果升级之前已经有id 那么升级之后会保留）

createIndex: 创建新数据库，对应某一个字段的索引（可以随便改，但最好跟老数据库的索引一致，新老数据库索引不互通）



注意区分这三个获取的东西，第一个获取数据库实例，对象存储需要根据这个来自己创建，但是第三个直接获取对象存储（直接获取的是当前数据库的对象存储）就不需要再创建

注意：主键不可重复创建，比如版本3要用到版本2的主键，继续延续下去（自动递增什么的），直接用版本2的对象存储即可（上面第三种代码）

* 1. 操作数据库

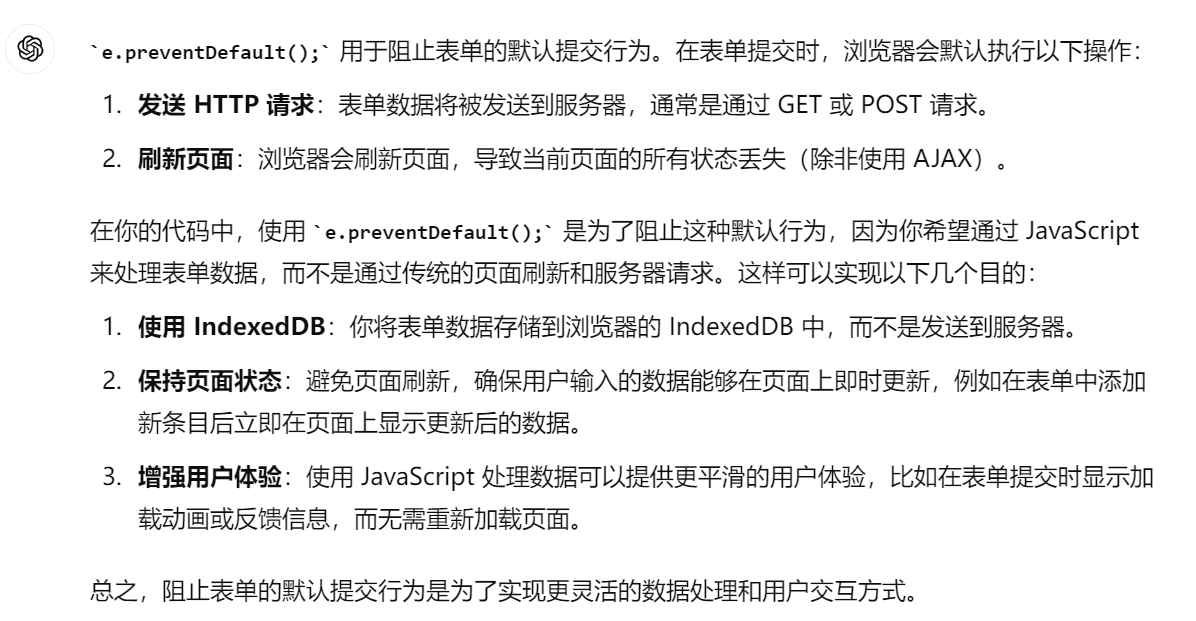
一定要创建事务来获取对象存储的引用，再来操作数据

let transaction = db.transaction(["notes"], "readwrite");这句话就是创建事务，

let objectStore = transaction.objectStore("notes"); 这句话是获取对象存储的引用

这样会更加安全,因为如果有任何一个数据库操作失败，都不会继续，会回滚，而上面直接获取对象存储的语句来操作数据库就不会这样

小贴士：阻止提交表单这一默认行为的使用条件



四十、IDBDatabase：transaction（）方法

这个东西用于创建事物对象，然后对对象存储进行操作（括号内填对象存储的名字，以及其他，详见网页）

<https://developer.mozilla.org/en-US/docs/Web/API/IDBDatabase/transaction>

transaction(["notes"], "readwrite") 以及 transaction.objectStore("notes")

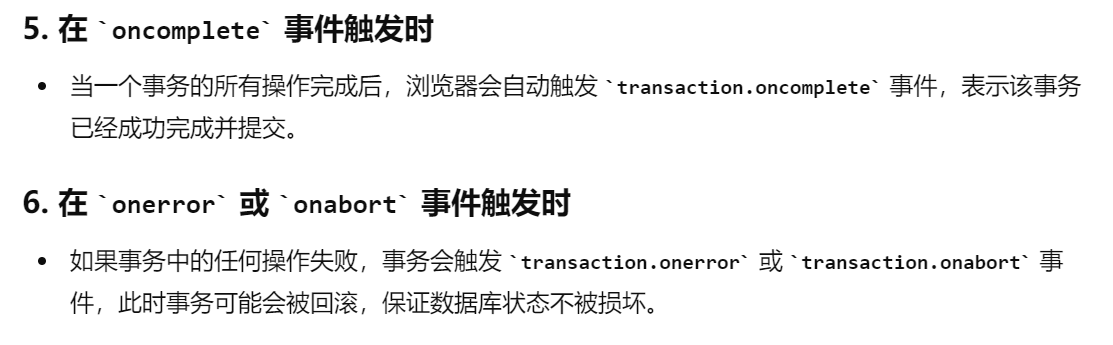


注意区分这两个

四十一、会创建事务的几个情况（第一个就是第四十条的东西）



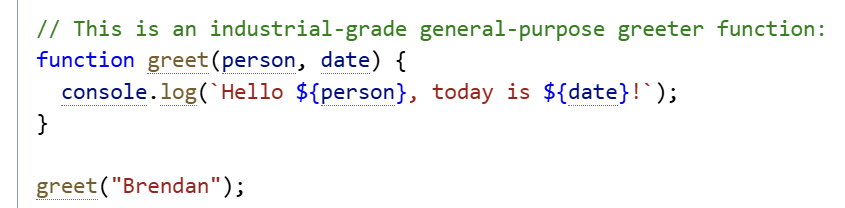




只有在事务存在的情况下才可以操作数据库，这就是为什么在上面的例子（添加数据库数据的时候，需要创建一个事物），这六条会自动创建事务，所以可以直接调用对象存储进行操作，无需自己创建

四十二、当未给函数传入形参的时候

比如有如下一个函数



我们没有给形参data传递任何值，那么js编译运行之后，这个函数输出的将会是  
**Hello Brendan, today is undefined!** 因为没有传入实际参数，形参默认值就是undefined