**1怎么理解闭包**

就是定义在一个函数内部的函数，可以使用其他函数的变量。

闭包是一种保护私有变量的机制，函数执行时形成私有作用域，保护私有变量不受外界影响。

内部函数可以使用外部函数的参数和变量，函数和变量不会被垃圾回收机制回收。

1. **回调函数**

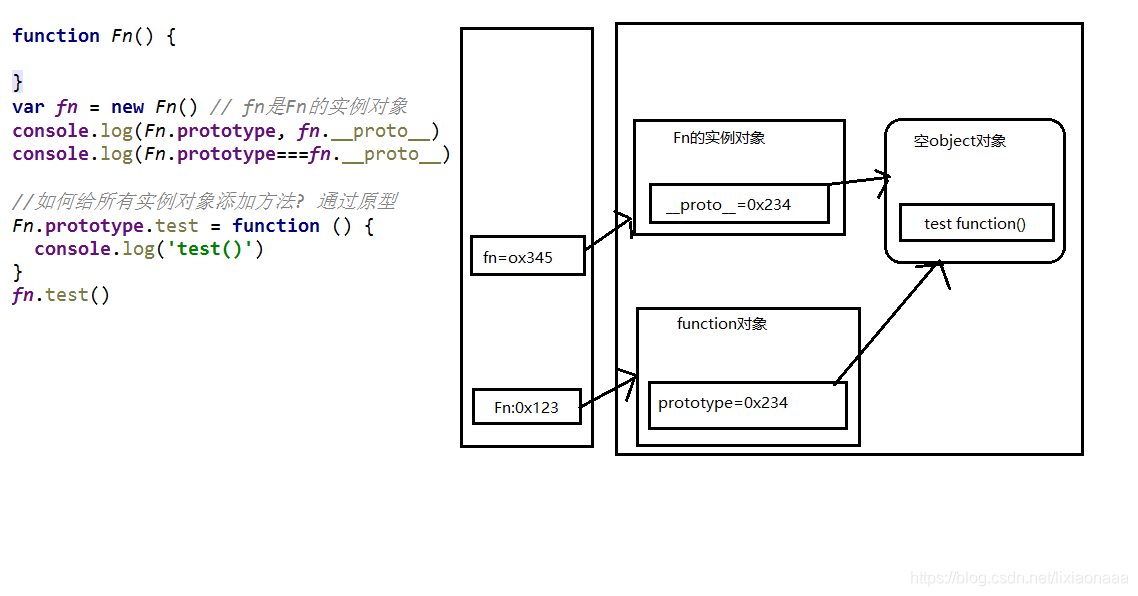
回调函数是一段可执行的代码段，它****作为一个参数****传递给其他的代码，其作用是****在需要的时候方便调用****这段（回调函数）代码。（作为[参数传递](https://so.csdn.net/so/search?q=%E5%8F%82%E6%95%B0%E4%BC%A0%E9%80%92&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_47075145/article/details/_blank)到另外一个函数中，这个作为参数的函数就是回调函数）

1. **常见的回调函数**

forEach ,setTimeout ,[ajax](https://so.csdn.net/so/search?q=ajax&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_44404444/article/details/_blank)请求回调函数,[生命周期](https://so.csdn.net/so/search?q=%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_44404444/article/details/_blank)回调函数,

1. **原型**

Fn是一个[构造函数](https://so.csdn.net/so/search?q=%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/lixiaonaaa/article/details/_blank)，每个构造函数都会自动生成一个prototype属性，指向一个空对象，这个空对象就是原型。每一个实例对象都会从原型继承属性和方法。



1. **原型链**

其实就是javascript的继承机制，是指获取JavaScript对象的属性会顺着其\_proto\_的指向寻找，直至找到Object.prototype上

1. **作用域,作用域链**

**作用域**就是变量的可用性的代码范围，就叫做这个变量的作用域。简单理解，就是在这个范围内，变量是可以使用的，超过这个范围，变量就无法使用，这个范围就是作用域。

作用域分为三种：全局作用域、局部作用域、块级作用域。

当你要访问一个变量时，首先会在当前作用域下查找，如果当前作用域下没有查找到，则返回上一级作用域进行查找，直到找到全局作用域，这个查找过程形成的链条叫做作用域链。

1. **执行上下文**

**全局执行上下文**

在执行全局代码前将window确定为全局执行上下文

对全局数据进行预处理

var定义的全局变量 ===> undefined，添加为window属性

function声明的全局函数 ===> 赋值（fun），添加为window的方法

this ===> 赋值为window

**函数执行上下文**

在调用函数，准备执行函数体之前，创建对应的函数执行上下文对象

对局部数据进行预处理：

形参变量====>赋值(实参数据)==>添加为执行上下文的属性

arguments===>赋值(实参列表)，添加为执行上下文的属性

var定义的局部变量===>undefined,添加为执行上下文的属性

function声明的函数===>赋值（fun），添加为执行上下文的属性

this===>赋值（调用函数的对象）

**执行上下文栈**

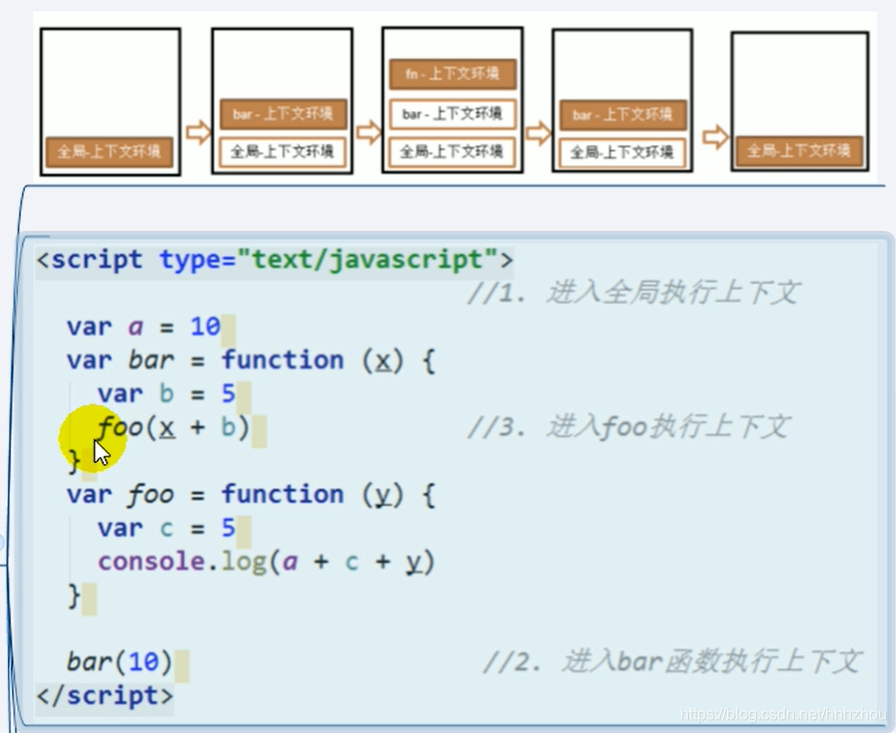
在全局代码执行前，JS引擎就会创建一个栈来存储管理所有的执行上下文对象

在全局执行上下文(window)确定后，将其添加到栈中(压栈)

在函数执行上下文创建后，将其添加到栈中(压栈)

在当函数执行完成后，将栈顶的对象移除（出栈）

当所有的代码执行完后，栈中只剩下window



**对象的创建模式**

工厂模式（Factory Method)

构造函数模式

原型模式

组合使用构造函数模式和原型模式

动态原型模式

寄生构造函数模式

稳妥构造函数模式

通过字面量创建对象模式

1. **继承模式**

****原型链继承****

****构造函数继承****

****组合继承****

****原型式继承****

****寄生式继承****

****寄生组合继承****

**ES6 Class extends**

1. **事件机制**

**Js事件机制描述的是事件在DOM里面传递的顺序，以及我们对这些事件做出如何的响应**

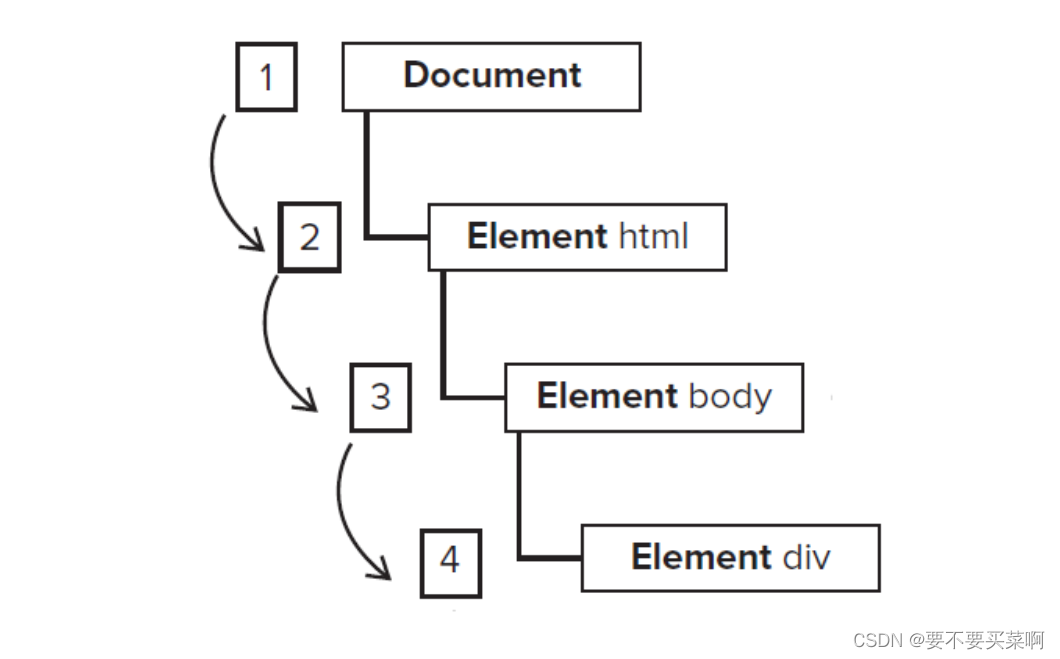
事件流(Event Flow)指的就是「****网页元素接收事件的顺序****」。****事件流可以分成两种机制****：

* 事件捕获(Event Capturing)
* 事件冒泡(Event Bubbling)

当一个事件发生后，会在子元素和父元素之间传播（propagation）。这种传播分成三个阶段：

1. 捕获阶段：事件从window对象****自上而下向目标节点****传播的阶段；
2. 目标阶段：****真正的目标节点正在处理事件****的阶段；
3. 冒泡阶段：事件从目标节点****自下而上向window对象传播****的阶段。

事件捕获指的是「从启动事件的元素节点开始，逐层往下传递」，直到最下层节点，也就是div。

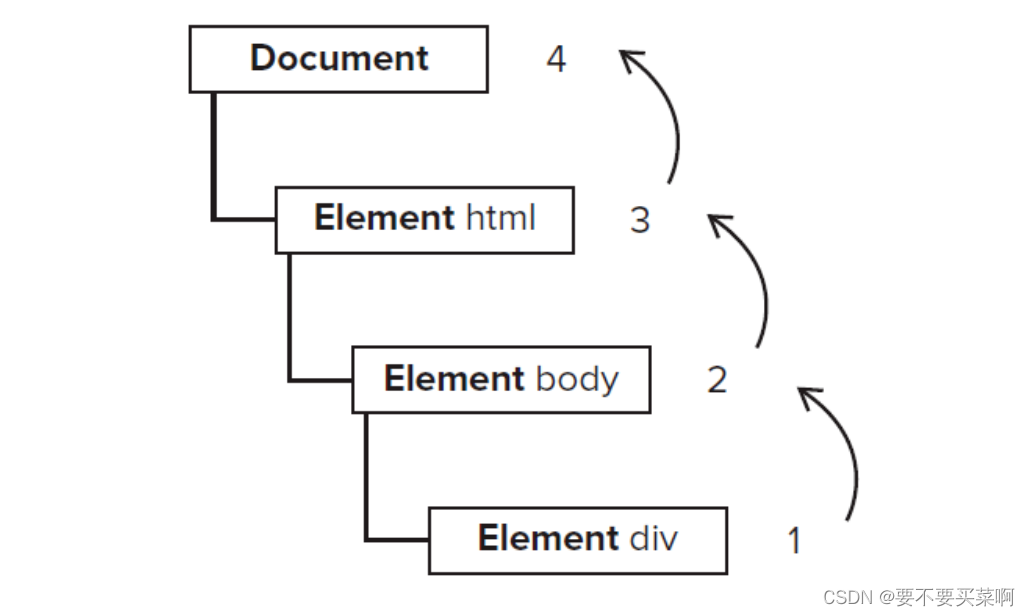


假设我们点击(click)了<div>点我</div>元素，那么在「事件捕获」的机制下，触发事件的顺序会是：

1. document
2. <html>
3. <body>
4. <div>点我</div>

像这样click****事件由上往下依序被触发，就是「事件捕获」机制****。

刚刚说过「事件捕获」机制是由上往下来传递，那么「事件冒泡」(Event Bubbling) 机制则正好相反。



假设我们点击(click)了<div>点我</div>元素，那么在「事件冒泡」的机制下，触发事件的顺序会是：

1. <div>点我</div>
2. <body>
3. <html>
4. document

像这样click事件****逐层向上依序被触发，就是「事件冒泡」机制****。

## ****事件监听 ( EventTarget.addEventListener() )****

 addEventListener()基本上有三个参数，分别是「事件名称」、「事件的处理程序」(事件触发时执行的function)，以及一个「Boolean」值，由这个Boolean决定事件是以「捕获」还是「冒泡」机制执行，若不指定则预设为「冒泡」。

1. **定时器的执行**

js 定时器有以下两个方法：

* setInterval() ：按照指定的周期（以毫秒计）来调用函数或计算表达式。方法会不停地调用函数，直到 clearInterval() 被调用或窗口被关闭。
* setTimeout() ：在指定的毫秒数后调用函数或计算表达式。

setInterval(code,millisec,lang)

|  |  |
| --- | --- |
| **参数** | **描述** |
| code | 必需。要调用的函数或要执行的代码串。 |
| millisec | 必须。周期性执行或调用 code 之间的时间间隔，以毫秒计。 |
| lang | 可选。 JScript | VBScript | JavaScript |

setTimeout(code,millisec,lang)

|  |  |
| --- | --- |
| **参数** | **描述** |
| code | 必需。要调用的函数后要执行的 JavaScript 代码串。 |
| millisec | 必需。在执行代码前需等待的毫秒数。 |
| lang | 可选。脚本语言可以是：JScript | VBScript | JavaScript |

1. **Web Workers**

web worker 是运行在后台的 JavaScript，不会影响页面的性能。

1. **js下载json对象**

有时候后端接口返回了一个JSON对象，为了方便查看和核对数据，可能想将其下载到文件中，那么只需要将下载文本的方法稍微改造一下即可，具体代码如下：

function downloadJson(fileName, json) {

const jsonStr = (json instanceof Object) ? JSON.stringify(json) : json;

const url = window.URL || window.webkitURL || window;

const blob = new Blob([jsonStr]);

const saveLink = document.createElementNS('http://www.w3.org/1999/xhtml', 'a');

saveLink.href = url.createObjectURL(blob);

saveLink.download = fileName;

saveLink.click();

}

有的时候可能希望保存到文件中的 json 数据是格式化的，这样方便查看，那么只需要稍微调整一下 JSON.stringify() 方法即可，先来看看 JSON.stringify() 方法的定义

JSON.stringify(value[, replacer [, space]])

value

* 将要序列化成 一个 JSON 字符串的值。

replacer 可选

* 如果该参数是一个函数，则在序列化过程中，被序列化的值的每个属性都会经过该函数的转换和处理；如果该参数是一个数组，则只有包含在这个数组中的属性名才会被序列化到最终的 JSON 字符串中；如果该参数为 null 或者未提供，则对象所有的属性都会被序列化。

space 可选

* 指定缩进用的空白字符串，用于美化输出（pretty-print）；如果参数是个数字，它代表有多少的空格；上限为 10。该值若小于 1，则意味着没有空格；如果该参数为字符串（当字符串长度超过 10 个字母，取其前 10 个字母），该字符串将被作为空格；如果该参数没有提供（或者为 null），将没有空格。

如果想要格式化 JSON，那么可以传入一个 space 参数，具体如下：

JSON.stringify(json, null, 4);

除了使用4个空格外，还可以制表符（\t）来缩进，如下：

JSON.stringify(json, null, '\t');

下载格式化JSON方法修改如下：

function downloadJson(fileName, json) {

const jsonStr = (json instanceof Object) ? JSON.stringify(json, null, 4) : json;

const url = window.URL || window.webkitURL || window;

const blob = new Blob([jsonStr]);

const saveLink = document.createElementNS('http://www.w3.org/1999/xhtml', 'a');

saveLink.href = url.createObjectURL(blob);

saveLink.download = fileName;

saveLink.click();

}