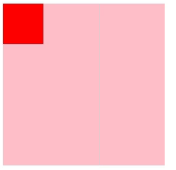
### JS事件冒泡

1. 冒泡顺序：div - body - html - document - window
2. 不冒泡的事件：blur focus load unload
3. onmouseout事件有两个问题：

给粉色父盒子添加onmouseout事件

当鼠标从红色子盒子移入粉盒子或者从粉盒子移入红盒子都会触发onmouseout事件，这是不合理的。jQ中的mouseleave不存在这个问题。

1. 阻止冒泡：event.stopPropagation / event.cancelBubble = true; (IE678)

### 事件委托

1. 定义：委托父元素代为执行子元素的事件。

三个同事签收快递，有两种办法：一是三个人都到公司门口等快递；二是委托前台MM代为签收。如果公司里来了新员工，前台MM也会在收到寄给新员工的快递后核实并代为签收。

列举两种用到委托的情形：

1. 比如有100个li，每个li都要注册点击事件，我们会用for循环给它们添加事件；
2. 页面加载的时候无法获取后来动态生成的元素，也就无法为这些元素注册事件。
3. 事件委托的原理：事件冒泡。
4. 如何实现？
5. 事件目标：通过event.target可以找到事件源。
6. on (要委托事件的父元素，事件类型，事件要注册在哪些元素上，发生事件后要执行的函数); on要自己封装，jQ有现成的。

### 事件绑定

1. 绑定事件的两种方式：on
2. 追加事件用添加事件监听器：addEventListener / removeEventListener
3. 类似的，window.onload 和 addLoadEvent

|  |  |
| --- | --- |
| function addLoadEvent(fn) {  var old = window.onload;  if (typeof old === "function") {  old();  fn();  } else {  fn();  }  } | function addLoadEvent(fn) {  var old = window.onload;  if (typeof old === "function") {  window.onload = function() {  old();  fn();  }  } else {  window.onload = function() {  fn();  }  }  } |

比较这两个打印结果，可以对onload事件的执行时机有更深的理解。

### 事件委托 / 代理

on

### 事件解绑

1. null
2. removeEventListener(“click”, fn)

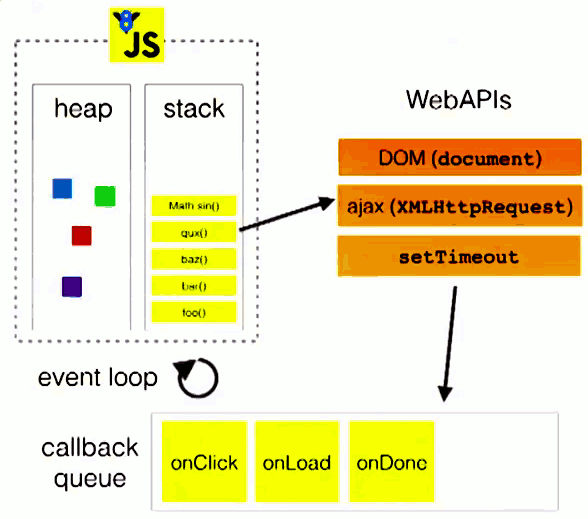
第二个参数必须传函数名，直接传匿名函数的话不起作用。

### 单线程 + 事件队列

JavaScript中的线程包括函数调用、Ajax、定时器、用户操作的事件（click、keyup、scroll等）。

事件队列中的任务执行的条件：

1. 主线程已空闲；
2. 任务满足触发条件：① 定时函数的触发条件是延时时间已经达到；② 事件函数的触发条件是特定事件被触发；③ Ajax的回调函数的触发条件是服务器端有数据响应。



for (var i = 0; i < 5; i++) {

console.log(i);

}

A：0 1 2 3 4

for (var i = 0; i < 5; i++) {

setTimeout(function() {

console.log(i);

}, 1000 \* i);

}

A： 隔秒打印5个5

for (var i = 0; i < 5; i++) {

(function(i) {

setTimeout(function() {

console.log(i);

}, i \* 1000);

})(i);

}

A：0 1 2 3 4

for (var i = 0; i < 5; i++) {

(function() {

setTimeout(function() {

console.log(i);

}, i \* 1000);

})(i);}

A：隔秒打印5个5

for (var i = 0; i < 5; i++) {

setTimeout((function(i) {

console.log(i);

})(i), i \* 1000);

}

A：瞬间打印0 1 2 3 4

setTimeout(function() {

console.log(1)}, 0);

new Promise(function executor(resolve) {

console.log(2);

for( var i=0 ; i<10000 ; i++ ) {

i == 9999 && resolve();

}

console.log(3);

})

.then(function() {

console.log(4);

});

console.log(5);

A：2 3 5 4 1

首先先碰到一个 setTimeout，于是会先设置一个定时，在定时结束后将传递这个函数放到任务队列里面，因此开始肯定不会输出 1 。

然后是一个 Promise，里面的函数是直接执行的，因此应该直接输出 2 3 。

然后，Promise 的 then 应当会放到当前 tick 的最后，但是还是在当前 tick 中。

因此，应当先输出 5，然后再输出 4 。

最后在到下一个 tick，就是 1 。

“2 3 5 4 1”