JavaScript的EventLoop

1.区分进程和线程

官方描述：一个进程是一个工厂，不同的工厂之间相互独立，一个工厂有它的独立资源，线程是工厂中的工人，工厂内至少有一个或多个工人，人协作完成任务，工人之间共享空间。

官方术语：

- 进程是cpu资源分配的最小单位（是能拥有资源和独立运行的最小单位）。
- 线程是cpu调度的最小单位（线程是建立在进程的基础上的一次程序运行单位，线程是进程中的一个实体）。
- 一个进程中可以有多个线程，不同进程之间也可以通信，不过代价较大。
- 单线程与多线程，都是指在一个进程内的单和多（所以核心还是得属于一个进程才行）。

2.浏览器的进程和线程

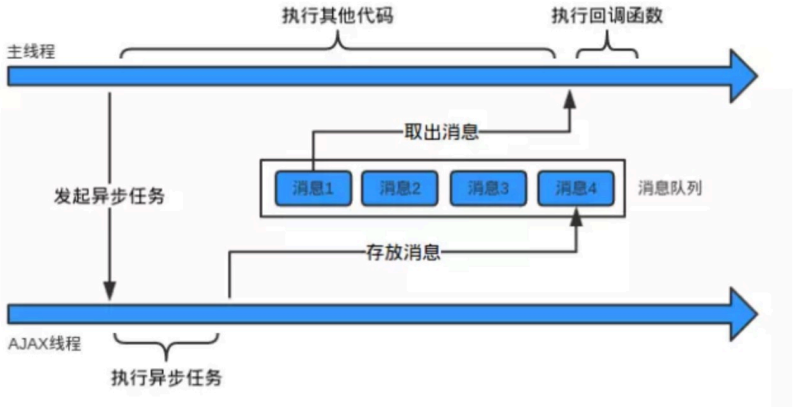
- 浏览器是多进程的（每开启一个网页至少会开一个进程，开启多个空白网页，默认为是一个进程）
- 浏览器之所以能够运行，是因为系统给它的进程分配了资源（cpu、内存）
- 简单点理解，每打开一个Tab页，就相当于创建了一个独立的浏览器进程。
- 每个Tab页进程中又有多个线程：（1）javascript引擎线程（2）界面渲染线程（3）浏览器事件触发线程 （4）HTTP请求线程

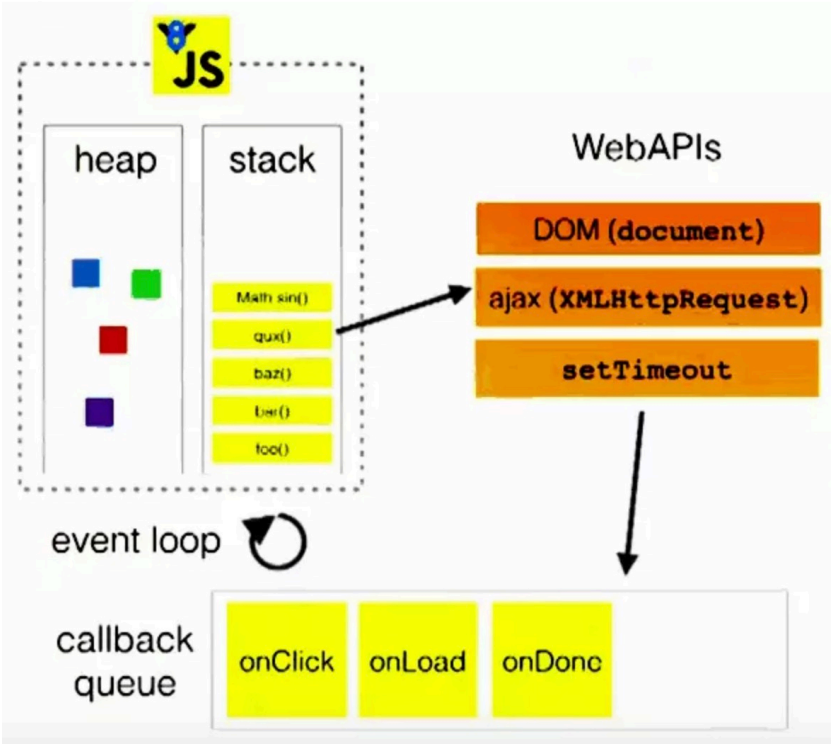
3.Javascript 为什么是单线程?

- 浏览器是用户进行操作触发的**事件驱动（Event driven）**，浏览器中很多行为是**异步（Async）的**，会创建事件并放入执行队列中。
- 假设JS有两个线程，一个线程在一个节点上添加内容，另一个线程又在这个节点上删除内容，这是浏览器应该以哪个线程为准？浏览器必须给发生事件的时间点排序。
- JS引擎是单线程处理它的任务队列，你可以理解成就是普通函数和回调函数构成的队列。当异步事件发生时，如mouse click, a timer fir XMLHttpRequest completing，将他们放入执行队列，等待当前代码执行完成。
- 单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。如果前一个任务耗时很长，后一个任务就不得不一直等

4.EventLoop（事件循环）

当一个异步事件发生的时候，它就进入事件队列。**浏览器有一个内部大消息循环，Event Loop（事件循环），会轮询大的事件队列并处理事件**如，浏览器当前正在忙于处理onclick事件，这时另外一个事件onSize发生了，这个异步事件就被放入事件队列等待处理，只有前面的处理完毕了才会执行这个事件。setTimeout也是一样，当调用的时候，JS引擎会启动定时器timer，大约xxms以后执行xxx，当定时器时间到，就把该事事件队列等待处理（浏览器不忙的时候才会真正执行）。





- **stack**为自动分配的内存空间，它由系统自动释放；而**heap**则是动态分配的内存，大小不定也不会自动释放。
- **基本类型**：存放在栈内存中的简单数据段，数据大小确定，内存空间大小可以分配。5种基本数据类型有**Undefined**、**Null**、**Boolean**、**Number** 和 **String**，它们是直接按值存放的，所以可以直接访问。
- **引用类型**：存放在堆内存中的对象，变量实际保存的是一个指针，这个指针指向另一个位置。每个空间大小不一样，要根据情况开进行
配。

1. 执行栈执行主线程任务，当有操作dom，ajax交互，使用定时器异步操作的时候，这些任务会被移入到 callback queue 任务队列中
2. 当主线程任务执行完毕为空时，会读取callback queue队列中的函数，进入主线程执行
3. 上述过程会不断重复，也就是常说的Event Loop(事件循环)。

5.macro task与micro task

在一个事件循环中，异步事件返回结果后会被放到一个任务队列中。然而，根据这个异步事件的类型，这个事件实际上会被对应的宏任务队列
务队列中去，当执行栈为空的时候，主线程会首先查看微任务中的事件，如果微任务不是空的那么执行微任务中的事件，如果没有在宏任务中
面的一个事件。把对应的回调加入当前执行栈...如此反复，进入循环。

- macro-task(宏任务)：（1）setTimeout（2）setInterval（3）setImmediate
- micro-task(微任务)：（1）Promise（2）process.nextTick

代码一

```
function test1() {
  console.log(1);
}

// T1-1
setTimeout(test1, 1000);
// (立即) 无输出
// (1s ->) 1

```

```
// T1-2
// setTimeout(test1(), 1000);
// (立即) 1
// (1s ->) 无输出


// T1-3
// setTimeout(console.log(1.1), 1000);
// (立即) 1.1
// (1s ->) 无输出


// ADD
// T1-4
// setTimeout('console.log(1.2)', 1000);
// (立即) 无输出
// (1s ->) 1.2
```

代码二

```
function test2(value) {
    value = value || 'default';
    console.log(value);
}

// T2-1
setTimeout(test2, 1000, 2.1);
// (立即) 无输出
// (1s ->) 2.1


// T2-2
// setTimeout(test2(), 1000, 2.2);
// (立即) default
// (1s ->) 无输出


// T2-3
// setTimeout(test2(2.3), 1000, 2.31);
// (立即) 2.3
// (1s ->) 无输出
```

代码三

```
function test3(value) {
    value = value || 'default';
    console.log(value);
    return test3;
}

// T3-1
setTimeout(test3, 1000, 3.1);
// (立即) 无输出
// (1s ->) 3.1


```

```
// T3-2
// setTimeout(test3(), 1000, 3.2);
// (立即) default
// (1s ->) 3.2


// T3-3
// setTimeout(test3(3.3), 1000, 3.31);
// (立即) 3.3
// (1s ->) 3.31
```

代码四

```
// T4-1
for (var i = 0; i < 5; i++) {
    console.log(i);
}
// (立即) 0 1 2 3 4


// T4-2
// for (var i = 0; i < 5; i++) {
//     setTimeout(function () {
//         console.log(i);
//     }, 1000 * i);
// }
// (立即) 无输出
// (0s ->) 5
// (1s ->) 5
// (2s ->) 5
// (3s ->) 5
// (4s ->) 5


// T4-3
// for (var i = 0; i < 5; i++) {
//     setTimeout(function (i) {
//         console.log(i);
//     }, 1000 * i);
// }
// (立即) 无输出
// (0s ->) undefined
// (1s ->) undefined
// (2s ->) undefined
// (3s ->) undefined
// (4s ->) undefined


// ADD
// T4-3-2
// for (var i = 0; i < 5; i++) {
//     setTimeout(function (i) {
//         console.log(i);
//     }, 1000 * i, 999);
// }
```

```
// (立即) 无输出
// (0s ->) 0
// (1s ->) 0
// (2s ->) 0
// (3s ->) 0
// (4s ->) 0


// T4-4
// for (var i = 0; i < 5; i++) {
//     (function (i) {
//         setTimeout(function () {
//             console.log(i);
//         }, i * 1000);
//     })(i);
// }
// (立即) 无输出
// (0s ->) 0
// (1s ->) 1
// (2s ->) 2
// (3s ->) 3
// (4s ->) 4


// T4-5
// for (var i = 0; i < 5; i++) {
//     (function () {
//         setTimeout(function () {
//             console.log(i);
//         }, i * 1000);
//     })(i);
// }
// (立即) 无输出
// (0s ->) 5
// (1s ->) 5
// (2s ->) 5
// (3s ->) 5
// (4s ->) 5


// ADD
// T4-5-2
// for (var i = 0; i < 5; i++) {
//     (function () {
//         setTimeout(function () {
//             console.log(i);
//         }, i * 1000);
//     })();
// }
// (立即) 无输出
// (0s ->) 5
// (1s ->) 5
// (2s ->) 5
// (3s ->) 5
```

```
// (4s ->) 5


// T4-6
// for (var i = 0; i < 5; i++) {
//     setTimeout((function (i) {
//         console.log(i);
//     })(i), i * 1000);
// }
// (立即) 0 1 2 3 4
// (0s ->) 无输出
// (1s ->) 无输出
// (2s ->) 无输出
// (3s ->) 无输出
// (4s ->) 无输出


// ADD
// T4-6-2
// for (var i = 0; i < 5; i++) {
//     setTimeout((function () {
//         console.log(i);
//     })(i), i * 1000);
// }
// (立即) 0 1 2 3 4
// (0s ->) 无输出
// (1s ->) 无输出
// (2s ->) 无输出
// (3s ->) 无输出
// (4s ->) 无输出


// ADD
// T4-6-3
// for (var i = 0; i < 5; i++) {
//     setTimeout((function () {
//         console.log(i);
//     })(), i * 1000);
// }
// (立即) 0 1 2 3 4
// (0s ->) 无输出
// (1s ->) 无输出
// (2s ->) 无输出
// (3s ->) 无输出
// (4s ->) 无输出
```

代码五

```
function someLoop() {
  var tag = true;
  var temp = 0;

  // 异步
  setTimeout(function () {
    tag = !tag;
```

```
    console.log(tag)
  }, 1000);

  // 同步
  while (tag) {
    temp++;
  }
}

someLoop();
```

代码六

```
const times = 10000;// for阻塞耗时 < 200ms
// const times = 50000;// 200ms < for阻塞耗时 < 300ms:
// const times = 100000;// for阻塞耗时 > 300ms

console.log(1);

setTimeout(function () {
    console.log(2);
}, 300);

setTimeout(function () {
    console.log(3)
}, 400);

var start = new Date();
for (var i = 0; i < times; i++) {
    console.log(4);
}
var end = new Date();
console.log('阻塞耗时: ' + Number(end - start) + '毫秒');

setTimeout(function () {
    console.log(5);
}, 100);

// for阻塞耗时 < 200ms
// 1
// 4 4 4```
// 5
// 2
// 3

// 200ms < for阻塞耗时 < 300ms:
// 1
// 4 4 4 4 4```
// 2
// 5
// 3

// for阻塞耗时 > 300ms
// 1
// 4 4 4 4 4```
```

```
// 2
// 3
// 5
```

代码七

```
setTimeout(() => {
  console.log(1)
}, 0);

new Promise((resolve) => {
  console.log(2);
  for (let i = 0; i < 10000; i++) {
    if (i === 9999) {
      resolve();
       console.log(i);
    }
  }
  console.log(3);
}).then(() => {
  console.log(4);
});

console.log(5);
// 2 9999 3 5 4 1
```