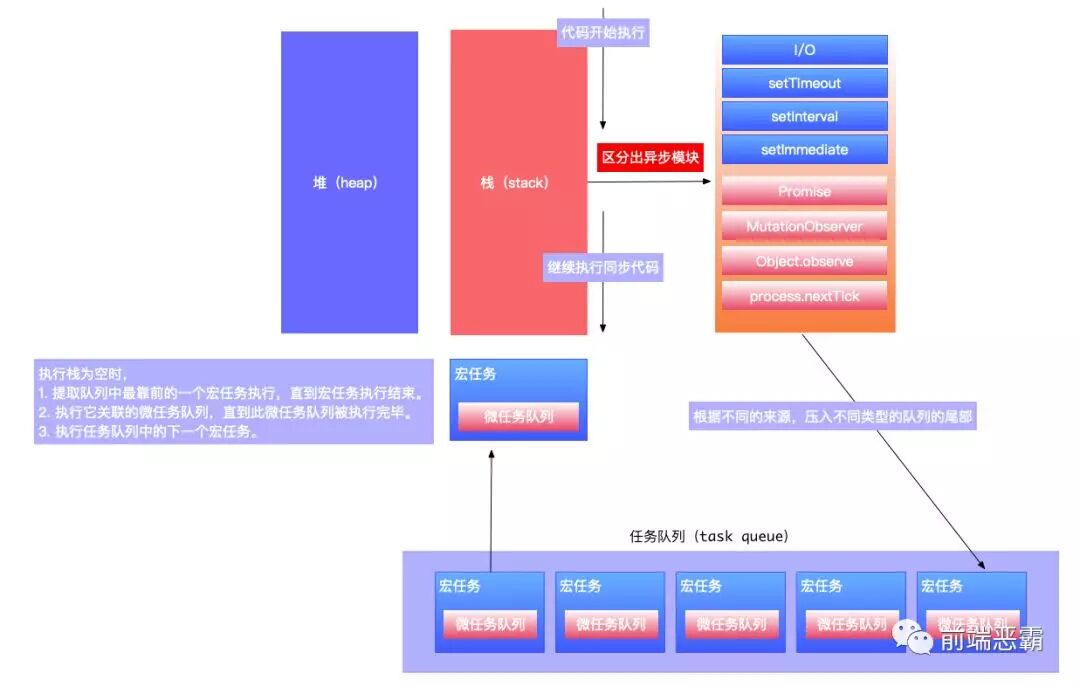
Js事件循环图



Promise

1. 全局的变量先执行，但是得考虑到js的执行顺序
2. 如果有promise，则执行promise里面的逻辑，如果遇到定时器，则将定时器指追加到任务队列等待执行
3. Promise里面的程序执行完成之后，再次执行then方法里面的逻辑

Class

1. class命名的类还是构造函数，但是比ES5的更像类，有自己的作用域
2. Class类可以创建一盒自身调用的静态方法，可以不实例化，如果实例化，不会被new关键字继承或者复制到实例对象中去
3. 原型方法必须经过实例化，然后复制到实例对象中去，然后进行调用
4. Class类内部默认是严格模式‘use strict’
5. class类中，可以声明constructor函数，如果不声明constructor函数,则浏览器会自动帮你创建constructor构造器
6. ES6的继承通过extends进行继承，super在子类里面接受
7. 继承时，super代表的是父类，super会调用自身的构造器函数，只想父类的构造器函数，然后进行复制并赋值，将父类的this指针只想子类

js模块化讲解：

js模块化，就是将程序按特定功能封装起来，并暴露，且在其他js文件内引入

优点：

1、更好的体现了js的模块化思想（作用域思想）

2、对程序进行了深度封装

3、解决了全局变量冲突

4、函数重名冲突,

5、提高代码的复用性，

6、减少了内存占用

7、按需加载

8、解耦

9、提高开发开发效率

10、降低维护成本，易于维护，迭代

框架，就是通过模块化封装 暴露 引入的

js模块化分为4中方式：

1、CMD

CMD是一种模块化产出的概念，体现者是SeaJS, 也是通过require进行加载

SeaJS是阿里的 玉伯 整出来的

都是通过define函数进行加载的

按需加载

不进行依赖前置

就近原则

define(function(require,exports,module){

var a = require('./a.js');

a.add\_uum(5);

a.reduce\_num()

module.exports = {

a1:a.add\_uum(5),

a2:a.reduce\_num()

};

var b = require('./b.js');

// 使用b文件功能

module.exports = b;

var $ = require('jquery');

// 使用$

module.exports = $;

})

// 2、AMD :其实是CommonJS的中转版本，可以实现同步加载

// 是一种js模块化概念的产出，最好的实践者或者代表就是 requireJS

// requireJS 体现了AMD规范，按需加载 ，提前加载 ，异步加载

// 都是通过define函数进行定义模块

// 依赖前置

// 全部加载

// 如下

define('loop',['swiper'],function(s){

// 具体的轮播图操作

// var a = require('./a.js');

// var b = require('./b.js');

// a.add\_uum(5);

});

define('layout',['bootstrap'],function(bs){

// 全局布局的逻辑

});

// 3、CommonJS

// 是js模块化概念的产出，最好的实践者是Node.js,同步加载模块

// 适用于服务端，不能使用在浏览器端

// 通过require加载文件

// 暴露：module.exports = {}

// 4、ES6 module

// 1、

// 默认暴露：export default 变量

// 引入：import xxx from 'xxx;

// 2、

// 直接暴露：export 变量

// import { xxxa,xxxb,xxxc } from 'xxx';

// 共同的引入：import \* as xxx from 'xxx';

1.AMD,CMD,ES6都是异步加载模块（CMD：按需加载）（AMD：是全部加载）

2.CommonJS：是同步加载模块

3.AMD要指定入口JS文件

4.node环境中默认有require环境，允许你直接使用

AMD 暴露 =》 require

CMD 暴露 =》 SeaJS

CommonJS 暴露 =》 node.js

ES6 暴露 =》 Module