题目

写了多年JS代码的你知道JS的事件循环机制吗?

回答

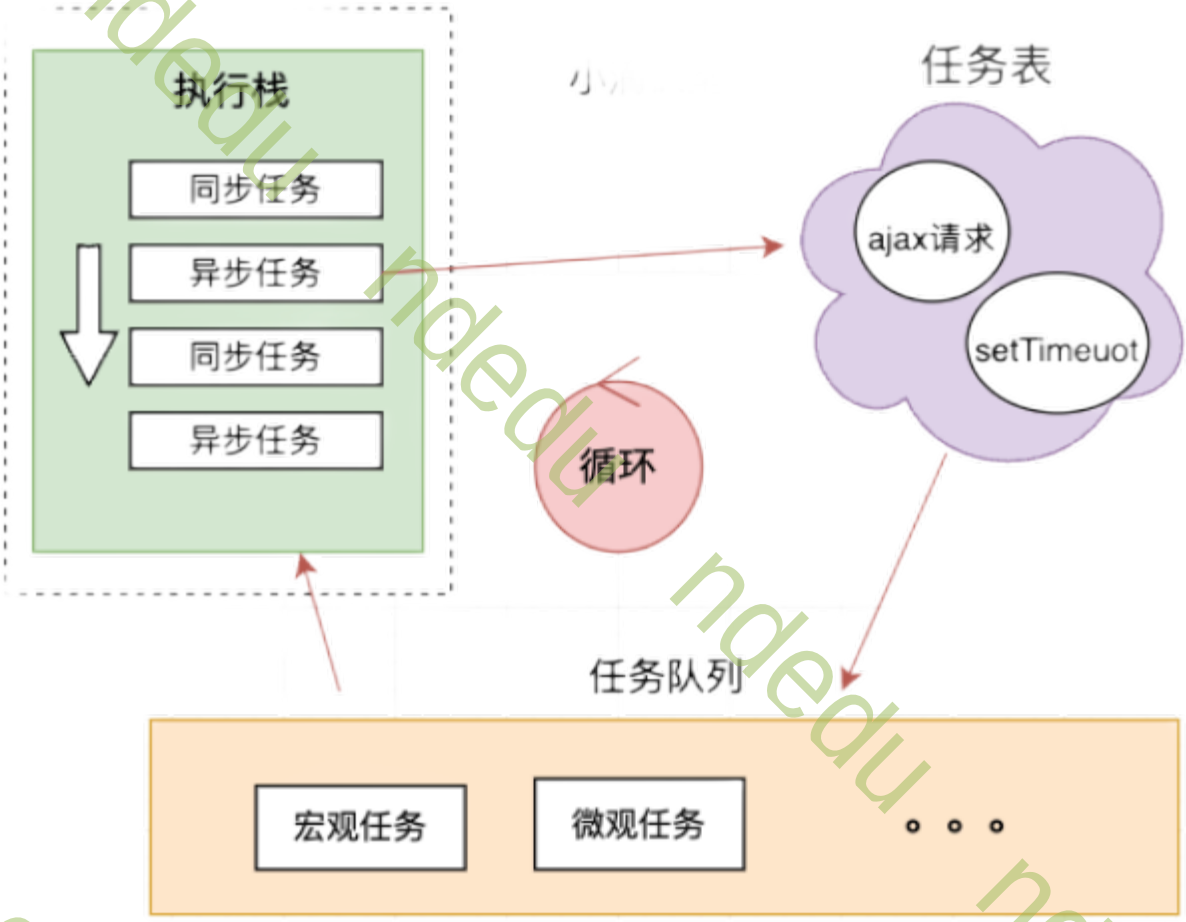
- 思路
  - JavaScript语言
    - 是一门**单线程**的**非阻塞**的脚本语言
      - 为什么 JS 是一门单线程的语言?

        > 因为在浏览器中，需要对各种的DOM操作；
        > 当JS是多线程的话，如果有两个线程同时对同一个DOM进行操作，一个是在这个DOM上绑定事件，另外一个是删除该DOM，此时就会产生歧义
        > 因此为了保证这种事情不会发生，所以JS以单线程来执行代码，保证了一致性

      - JS 非阻塞应该如何理解?

        > 当JS代码从上往下执行，遇到需要进行一项异步任务的时候，
        > 主线程会挂起这个任务，继续往下执行代码，然后在异步任务返回结果的时候再根据一定规则去执行

        思考：那么这个非阻塞是如何实现的呢？此时就需要用到事件循环（event loop）
  - 事件循环



- 结合代码分析事件循环

```
// 同步任务
console.log('首次同步任务开始');

// 异步任务（宏任务）
setTimeout(() => {
  console.log('setTimeout 1');
  new Promise((resolve) => {
    console.log('Promise1');
    resolve();
  }).then(() => {
    console.log('Promise then 1');
  });
}, 1000);

// 同步任务
console.log('首次同步任务结束');

// 异步任务（微任务）
new Promise((resolve) => {
  console.log('Promise2');
  resolve();
}).then(() => {
```

```
  console.log('Promise then 2');
});
```

- 异步任务分类：宏任务（setTimeout），微任务（promise）
- 所有同步任务都在主线程上执行，形成一个执行栈
- 遇到异步任务放到任务表中，等事件执行完成（ajax请求完成、setTimeout设置时间到期），之后放入到任务队列
- 当执行栈的同步任务执行完成之后，就会执行任务队列的第一个异步任务，其中把宏观任务和微观任务都执行完成后才进行下一次循环

ndedu