# 任务管理

JavaScript 语言的一大特点就是单线程，也就是说同一个时间只能处理一个任务。为了协调事件、用户交互、脚本、UI 渲染和网络处理等行为，防止主线程的不阻塞，（事件循环）Event Loop 的方案应用而生。
JavaScript 处理任务是在等待任务、执行任务 、休眠等待新任务中不断循环中，也称这种机制为事件循环。

- 主线程中的任务执行完后，才执行任务队列中的任务
- 有新任务到来时会将其放入队列，采取先进先执行的策略执行队列中的任务
- 比如多个

setTimeout 同时到时间了，就要依次执行
任务包括 script(整体代码)、 setTimeout、setInterval、DOM 渲染、DOM 事件、Promise、XMLHTTPREQUEST 等

## #原理分析

下面通过一个例子来详细分析宏任务与微任务

```
console.log("后盾人");
setTimeout(function() {
  console.log("定时器");
}, 0);
Promise.resolve()
  .then(function() {
    console.log("promise1");
  })
  .then(function() {
    console.log("promise2");
  });
console.log("houdunren.com");

#输出结果为
后盾人
houdunren.com
promise1
promise2
定时器
```

```
20
```

1. 先执最前面的宏任务 script，然后输出

```
script start
```

2. 然后执行到 setTimeout 异步宏任务，并将其放入宏任务队列，等待执行

3. 之后执行到 Promise.then 微任务，并将其放入微任务队列，等待执行

4. 然后执行到主代码输出

```
script end
```

5. 主线程所有任务处理完成

6. 通过事件循环遍历微任务队列，将刚才放入的 Promise.then 微任务读取到主线程执行，然后输出

```
promise1
```

7. 之后又执行 promse.then 产生新的微任务，并放入微任务队列

8. 主线程任务执行完毕

9. 现次事件循环遍历微任务队列，读取到 promise2 微任务放入主线程执行，然后输出

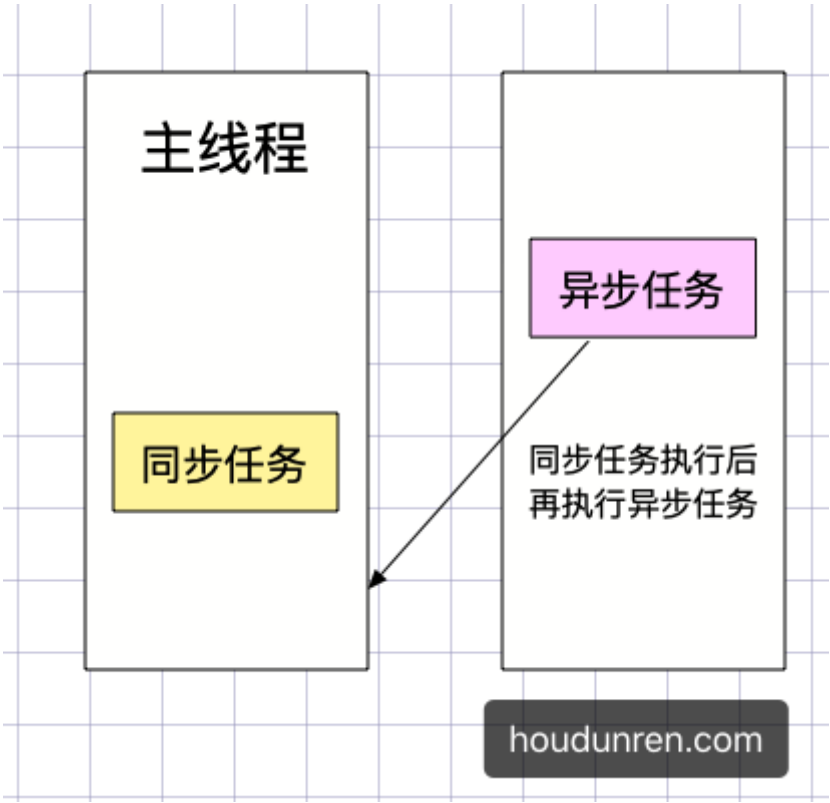
```
promise2
```

10. 主线程任务执行完毕

11. 此时微任务队列已经无任务，然后从宏任务队列中读取到 setTimeout 任务并加入主线程，然后输出

```
setTimeout

```




## #脚本加载

引擎在执行任务时不会进行 DOM 渲染，所以如果把`script` 定义在前面，要先执行完任务后再渲染 DOM，建议将`script` 放在 BODY 结束标签前。

## #定时器

定时器会放入异步任务队列，也需要等待同步任务执行完成后执行。

下面设置了 6 毫秒执行，如果主线程代码执行 10 毫秒，定时器要等主线程执行完才执行。

HTML 标准规定最小时间不能低于 4 毫秒，有些异步操作如 DOM 操作最低是 16 毫秒，总之把时间设置大些对性能更好。

```
setTimeout(func,6);

```

下面的代码会先输出 `houdunren.com` 之后输出 `后盾人`

```
setTimeout(() => {
  console.log("后盾人");
}, 0);
console.log("houdunren.com");

```

> 这是对定时器的说明，其他的异步操作如事件、XMLHTTPREQUEST 等逻辑是一样的

## #微任务

微任务一般由用户代码产生，微任务较宏任务执行优先级更高，`Promise.then` 是典型的微任务，实例化 Promise 时执行的代码是同步的，便 then 注册的回调函数是异步微任务的。
任务的执行顺序是同步任务、微任务、宏任务所以下面执行结果是 `1、2、3、4`

```
setTimeout(() => console.log(4));

new Promise(resolve => {
  resolve();
  console.log(1);
}).then(_ => {
  console.log(3);
});

console.log(2);

```

我们再来看下面稍复杂的任务代码

```
setTimeout(() => {
  console.log("定时器");
  setTimeout(() => {
    console.log("timeout timeout");
  }, 0);
  new Promise(resolve => {
    console.log("settimeout Promise");
    resolve();
  }).then(() => {
```

```
    console.log("settimeout then");
  });
}, 0);
new Promise(resolve => {
  console.log("Promise");
  resolve();
}).then(() => {
  console.log("then");
});
console.log("后盾人");

```

以上代码执行结果为

```
Promise
后盾人
then
定时器
settimeout Promise
settimeout then
timeout timeout

```

## #实例操作

### #进度条

下面的定时器虽然都定时了一秒钟，但也是按先进行出原则，依次执行

```
let i = 0;
setTimeout(() => {
  console.log(++i);
}, 1000);

setTimeout(() => {
  console.log(++i);
}, 1000);

```

下面是一个进度条的示例，将每个数字放在一个任务中执行

```
<body>
  <style>
    body {
      padding: 30px;
    }
    #hd {
      height: 30px;
      background: yellowgreen;
      width: 0;
      text-align: center;
      font-weight: bold;
    }
  </style>
  <div id="hd"></div>
</body>

<script>
  function view() {
    let i = 0;
    (function handle() {
      hd.innerHTML = i + "%";
      hd.style.width = i + "%";
      if (i++ < 100) {
        setTimeout(handle, 20);
      }
    })();
  }
  view();
  console.log("定时器开始了...");
</script>

```

## #任务分解

一个比较耗时的任务可能造成游览器卡死现象，所以可以将任务拆分为多小小异步小任务执行。下面是一个数字统计的函数，我们会发现运行时间特别长

```

```

```
console.time("runtime");
function hd(num) {
  let count = 0;
  for (let i = 0; i <= num; i++) {
    count += i;
  }
  console.log(count);
  console.timeEnd("runtime");
}
let num=987654321;
hd(num);
console.log("houdunren.com"); //需要等待上面执行完才会执行

```

现在把任务分解成小块放入任务队列，游览器就不会出现卡死的现象了，也不会影响后续代码的执行

```
console.time("runtime");
let count = 0;
let num = 987654321;
function hd() {
  for (let i = 0; i < 100000000; i++) {
    if (num <= 0) break;
    count += num--;
  }
  if (num > 0) {
    console.log(num);
    setTimeout(hd);
  } else {
    console.log(num);
    console.log(count);
  }
}
hd();
console.log("houdunren.com"); //立刻显示出来

```

交给微任务处理是更好的选择

```
async function hd(num) {
```

```
  let res = await Promise.resolve().then(_ => {
    let count = 0;
    for (let i = 0; i < num; i++) {
      count += num--;
    }
    return count;
  });
  console.log(res);
}
hd(987654321);
console.log("后盾人");
```