# Javascript异步编程的4种方法

作者： 阮一峰

日期： 2012年12月21日

你可能知道，Javascript语言的执行环境是"单线程"（single thread）。

所谓"单线程"，就是指一次只能完成一件任务。如果有多个任务，就必须排队，前面一个任务完成，再执行后面一个任务，以此类推。

这种模式的好处是实现起来比较简单，执行环境相对单纯；坏处是只要有一个任务耗时很长，后面的任务都必须排队等着，会拖延整个程序的执行。常见的浏览器无响应（假死），往往就是因为某一段Javascript代码长时间运行（比如死循环），导致整个页面卡在这个地方，其他任务无法执行。

为了解决这个问题，Javascript语言将任务的执行模式分成两种：同步（Synchronous）和异步（Asynchronous）。

"同步模式"就是上一段的模式，后一个任务等待前一个任务结束，然后再执行，程序的执行顺序与任务的排列顺序是一致的、同步的；"异步模式"则完全不同，每一个任务有一个或多个回

调函数（callback），前一个任务结束后，不是执行后一个任务，而是执行回调函数，后一个任务则是不等前一个任务结束就执行，所以程序的执行顺序与任务的排列顺序是不一致的、异步的。

"异步模式"非常重要。在浏览器端，耗时很长的操作都应该异步执行，避免浏览器失去响应，最好的例子就是Ajax操作。在服务器端，"异步模式"甚至是唯一的模式，因为执行环境是单线程的，如果允许同步执行所有http请求，服务器性能会急剧下降，很快就会失去响应。

本文总结了"异步模式"编程的4种方法，理解它们可以让你写出结构更合理、性能更出色、维护更方便的Javascript程序。

## 一、回调函数

这是异步编程最基本的方法。

假定有两个函数f1和f2，后者等待前者的执行结果。

```
f1();
f2();
```

如果f1是一个很耗时的任务，可以考虑改写f1，把f2写成f1的回调函数。

```
function f1(callback){
    setTimeout(function () {
        // f1的任务代码
        callback();
    }, 1000);
}
```

执行代码就变成下面这样：

```
f1(f2);
```

采用这种方式，我们把同步操作变成了异步操作，f1不会堵塞程序运行，相当于先执行程序的主要逻辑，将耗时的操作推迟执行。

回调函数的优点是简单、容易理解和部署，缺点是不利于代码的阅读和维护，各个部分之间高度耦合（Coupling），流程会很混乱，而且每个任务只能指定一个回调函数。

## 二、事件监听

另一种思路是采用事件驱动模式。任务的执行不取决于代码的顺序，而取决于某个事件是否发生。

还是以f1和f2为例。首先，为f1绑定一个事件（这里采用的jQuery的写法）。

```
f1.on('done', f2);
```

上面这行代码的意思是，当f1发生done事件，就执行f2。然后，对f1进行改写：

```
function f1(){
    setTimeout(function () {
        // f1的任务代码
        f1.trigger('done');
    }, 1000);
}
```

f1.trigger('done')表示，执行完成后，立即触发done事件，从而开始执行f2。

这种方法的优点是比较容易理解，可以绑定多个事件，每个事件可以指定多个回调函数，而且可以"去耦合"（Decoupling），有利于实现模块化。缺点是整个程序都要变成事件驱动型，运行流程会变得很不清晰。

## 三、发布/订阅

上一节的"事件"，完全可以理解成"信号"。

我们假定，存在一个"信号中心"，某个任务执行完成，就向信号中心"发布"（publish）一个信号，其他任务可以向信号中心"订阅"（subscribe）这个信号，从而知道什么时候自己可以开始执行。这就叫做"发布/订阅模式"（publish-subscribe pattern），又称"观察者模式"（observer pattern）。

这个模式有多种实现，下面采用的是Ben Alman的Tiny Pub/Sub，这是jQuery的一个插件。

首先，f2向"信号中心"jQuery订阅"done"信号。

```
jQuery.subscribe("done", f2);
```

然后，f1进行如下改写：

```
function f1(){
    setTimeout(function () {
        // f1的任务代码
        jQuery.publish("done");
    }, 1000);
}
```

jQuery.publish("done")的意思是，f1执行完成后，向"信号中心"jQuery发布"done"信号，从而引发f2的执行。

此外，f2完成执行后，也可以取消订阅（unsubscribe）。

```
jQuery.unsubscribe("done", f2);
```

这种方法的性质与"事件监听"类似，但是明显优于后者。因为我们可以通过查看"消息中心"，了解存在多少信号、每个信号有多少订阅者，从而监控程序的运行。

### 四、Promises对象

Promises对象是CommonJS工作组提出的一种规范，目的是为异步编程提供统一接口。

简单说，它的思想是，每一个异步任务返回一个Promise对象，该对象有一个then方法，允许指定回调函数。比如，f1的回调函数f2,可以写成：

```
f1().then(f2);
```

f1要进行如下改写（这里使用的是jQuery的实现）：

```
function f1(){
```

```
    var dfd = $.Deferred();

    setTimeout(function () {
        // f1的任务代码
        dfd.resolve();
    }, 500);

    return dfd.promise;
}
```

这样写的优点在于，回调函数变成了链式写法，程序的流程可以看得很清楚，而且有一整套的配套方法，可以实现许多强大的功能。

比如，指定多个回调函数：

```
f1().then(f2).then(f3);
```

再比如，指定发生错误时的回调函数：

```
f1().then(f2).fail(f3);
```

而且，它还有一个前面三种方法都没有的好处：如果一个任务已经完成，再添加回调函数，该回调函数会立即执行。所以，你不用担心是否错过了某个事件或信号。这种方法的缺点就是编写和理解，都相对比较难。

**五、参考链接**

* Asynchronous JS: Callbacks, Listeners, Control Flow Libs and Promises

（完）

## 文档信息

- 版权声明：自由转载-非商用-非衍生-保持署名（创意共享3.0许可证）
- 发表日期： 2012年12月21日
- 更多内容： 档案 » JavaScript

- 购买文集： 《如何变得有思想》
- 社交媒体： twitter， weibo
- Feed订阅：

## 相关文章

- **2016.05.25:** React Router 使用教程

  > 真正学会 React 是一个漫长的过程。

- **2016.04.12:** 跨域资源共享 CORS 详解

  > CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）。

- **2016.04.08:** 浏览器同源政策及其规避方法

  > 浏览器安全的基石是"同源政策"（same-origin policy）。很多开发者都知道这一点，但了解得不全面。

- **2016.03.12:** Node 应用的 Systemd 启动

  > 前面的文章介绍了 Systemd 的操作命令和基本用法，今天给出一个实例，如何使用 Systemd 启动一个 Node 应用。