JS异步发展史及Promise

- JS的异步发展史

  callback、发布订阅、观察者、*promise*、*generator*、*async/await*

- Promise的常用用法及实现

  resolve/reject、*then*、*catch*、*promise.all*、*promise.finally…*

JS的异步发展史

Javascript语言的执行环境是“单线程”。

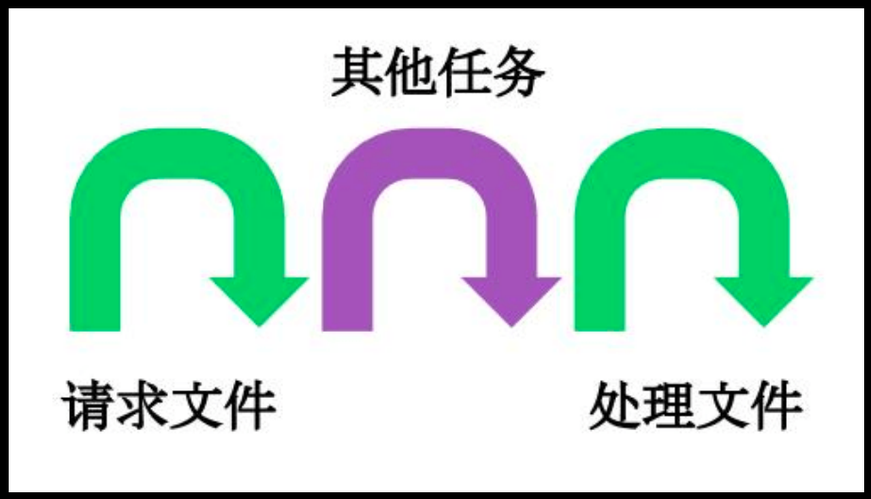
单线程的坏处： 只要有一个任务耗时很长，后面的任务都必须排队等着，会拖延整个程序的执行。

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）。

异步

异步asynchronous：

通俗的说异步就是一个任务分成两段，先执行第一段，然后转而执行其他任务，等做好了准备，再回过头执行第二段。




```
// 代码A
fs.readFile('../age.txt', 'utf8', function (err, data) {
    // 代码C
    console.log(data);
});
// 代码B
```

1、callback回调函数

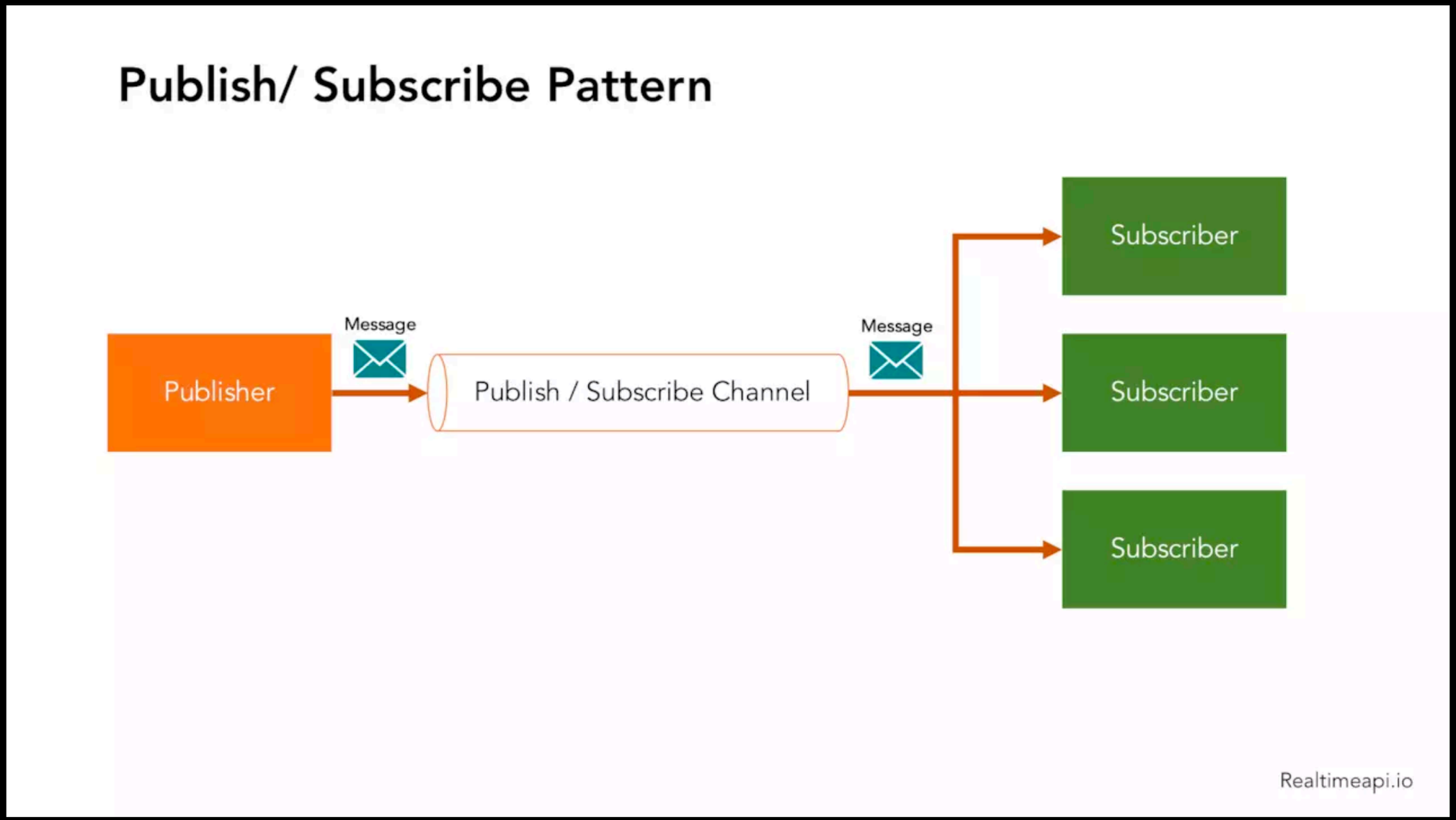
```
fs.readFile('aaa.txt', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

回调函数的缺点：

- 回调地狱：多层嵌套问题，难以维护。
- 违反直觉：我们的直觉就是顺序执行，从上到下。
- 错误追踪：错误不好处理，try/catch，err
- 并发执行：例如同时读多个文件，不知道谁先完成

2、发布订阅

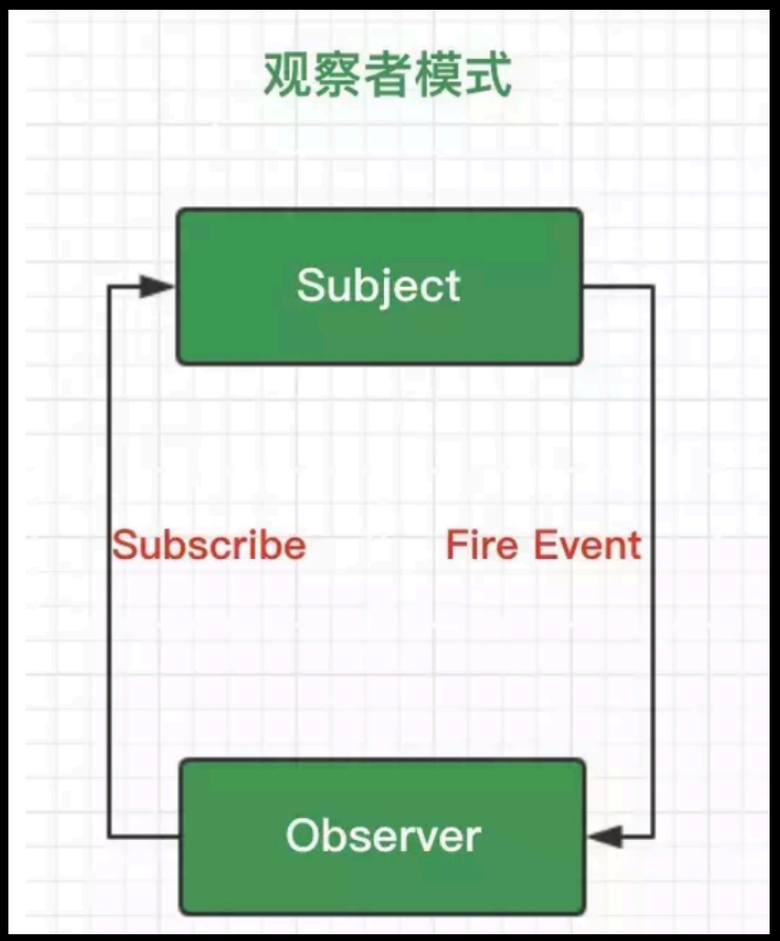
发布订阅实现了一个事件与多个回调函数的关联



3、观察者模式

只要当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新

观察者模式中存在两个角色：观察者和被观察者。



4、Promise

Promise 是异步编程的一种解决方案，ES6 将其写进了语言标准，统一了用法，原生提供了promise对象

Promise解决的问题：

- 回调地狱
- 错误捕获不好处理错误
- 异步并发问题

Promise常用属性方法：

- promise.prototype.then()
- promise.prototype.catch()
- promise.prototype.finally()
- promise.all

5、generator

Generator 函数是 ES6 提供的一种异步编程解决方案。

执行 Generator 函数会返回一个迭代器Iterator对象。
Generator生成器 => Iterator迭代器。

使用Generator可控制异步流程，使异步看起来和同步一样。
相比Promise，Generator可以省略then方法。

使用方法: 见代码

Generator是一个生成器，生成一个迭代器对象

迭代器有一个next方法，调用一次就会继续向下执行，直到遇到下一个yield或return

next()方法可以带一个参数，该参数会被当做上一条yield语句的返回值，
并赋值给yield前面等号前的变量

每遇到一个yield,就会返回一个{value:xxx,done:bool}的对象，然后暂停

```
/**
 * 2
 */
function* buy() {
    let a = yield 1;
    console.log(a);
    let b = yield 2;
    console.log('b',b);
    return b;
}

let it2 = buy();
it2.next('hello'); // 第一次的next传递参数是无效的
it2.next('world');
it2.next('zf');
```

6、async/await

ES2017 标准引入了 async 函数。
async 函数就是generator的语法糖。

async对 Generator 函数的改进，体现在以下四点

1. **内置执行器**
2. **更好的语义**
3. **更广的适用性**
4. **返回值是 Promise**

使用方法: 见代码

感谢观看