前端异步工具

回调函数

字面上的理解，回调函数就是一个参数，将这个函数作为参数传到另一个函数里面，当那个函数执行完之后，再执行传进去的这个函数。这个过程就叫做回调。

在JavaScript中，回调函数具体的定义为： 函数A作为参数(函数引用)传递到另一个函数B中，并且这个函数B执行函数A。我们就说函数A叫做回调函数。如果没有名称(函数表达式)，就叫做匿名回调函数。

同时补充回调函数应用场合和优缺点：

- 资源加载：动态加载js文件后执行回调，加载iframe后执行回调，ajax操作回调，图片加载完成执行回调，AJAX等等。
- DOM事件及Node.js事件基于回调机制(Node.js回调可能会出现多层回调嵌套的问题)。
- setTimeout的延迟时间为0，这个hack经常被用到，settimeout调用的函数其实就是一个 callback的体现。

  CSS hack是通过在CSS样式中加入一些特殊的符号，让不同的浏览器识别不同的符号（什么样的浏览器识别什么样的符号是有标准的，CSS hack就是让你记住这个标准），以达到应用不同的CSS样式的目的。
- 链式调用：链式调用的时候，在赋值器(setter)方法中(或者本身没有返回值的方法中)很容易实现链式调用，而取值器(getter)相对来说不好实现链式调用，因为你需要取值器返回你需要的数据而不是this指针，如果要实现链式方法，可以用回调函数来实现。
- setTimeout、setInterval的函数调用得到其返回值。由于两个函数都是异步的，即：他们的调用时序和程序的主流程是相对独立的，所以没有办法在主体里面等待它们的返回值，它们被打开的时候程序也不会停下来等待，否则也就失去了setTimeout及setInterval的意义了，所以用return已经没有意义，只能使用callback。callback的意义在于将timer执行的结果通知给代理函数进行及时处理。

回调函数这种方式

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“，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。

1. Promise（CommandJS提出的一种规范 ）

https://www.cnblogs.com/whybxy/p/7645578.html

原理：

Promise是一个构造函数，自己身上有all、reject、resolve这几个眼熟的方法，原型上有then、catch等同样很眼熟的方法。

Promise，就是一个特殊的Javascript对象，用来传递异步操作的消息，避免了层层嵌套的回调函数。它为异步操作提供了统一的API接口，可供进一步处理。

Promise就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。

它也代表了某种承诺，即无论你异步操作成功与否，这个对象最终都会返回一个值给你。比如在Ajax请求成功后调用 resolve回调函数来处理结果，如果请求失败则调用 reject回调函数来处理错误。

有三种状态：Pending（进行中）、Resolved（已完成，又称Fulfilled）和Rejected（已失败）。一开始请求发出后，状态是Pending,表示正在等待处理完毕，这个状态是中间状态而且是单向不可逆的。成功获得值后状态就变为fulfilled，然后将成功获取到的值存储起来，后续可以通过调用 then方法传入的回调函数来进一步处理。而如果失败了的话，状态变为rejected,错误可以选择抛出（throw)或者调用reject方法来处理。

catch方法，它其实它和then的第二个参数一样，用来指定reject的回调，效果和写在then的第二个参数里面一样。不过它还有另外一个作用：在执行resolve的回调（也就是上面then中的第一个参数）时，如果抛出异常了（代码出错了），那么并不会报错卡死js，而是会进到这个catch方法中。

Promise的all方法提供了并行执行异步操作的能力，并且在所有异步操作执行完后才执行回调。

all方法的效果实际上是「谁跑的慢，以谁为准执行回调」，那么相对的就有另一个方法「谁跑的快，以谁为准执行回调」，这就是race方法，这个词本来就是赛跑的意思，用法与all一样。

Promise基本语法如下

- Promise实例必须实现then这个方法
- then()必须可以接收两个函数作为参数

- then()返回的必须是一个Promise实例tor函数

2. es7 async await

async 函数的实现原理，就是将 Generator 函数和自动执行器，包装在一个函数里。它是Generator 函数的语法糖，async函数将 Generator 函数的星号（*）替换成async，将yield替换成await。async函数对 Generator 函数的改进，体现在以下四点。

（1）内置执行器。

async函数的执行，与普通函数一模一样，只要一行

（2）更好的语义。

async和await，比起星号和yield，语义更清楚了。async表示函数里有异步操作，await表示紧跟在后面的表达式需要等待结果。

（3）更广的适用性。

async函数的await命令后面，可以是 Promise 对象和原始类型的值（数值、字符串和布尔值，但这时会自动转成立即 resolved 的 Promise 对象

（4）返回值是 Promise。

可以用then方法指定下一步的操作。

进一步说，async函数完全可以看作多个异步操作，包装成的一个 Promise 对象，而await命令就是内部then命令的语法糖。

基本用法

async函数返回一个 Promise 对象，async函数内部return语句返回的值，会成为then方法回调函数的参数，使用then方法添加回调函数。当函数执行的时候，一旦遇到await就会先返回，等到异步操作完成，再接着执行函数体内后面的语句，async函数内部抛出错误，会导致返回的 Promise 对象变为reject状态。抛出的错误对象会被catch方法回调函数接收到

语法

有时，我们希望即使前一个异步操作失败，也不要中断后面的异步操作。这时可以将第一个await放在try...catch结构里面，这样不管这个异步操作是否成功，第二个await都会执行。

另一种方法是await后面的 Promise 对象再跟一个catch方法，处理前面可能出现的错误

2. es6 Generator

语法上，首先可以把它理解成，Generator 函数是一个状态机，封装了多个内部状态。

执行 Generator 函数会返回一个遍历器对象，也就是说，Generator 函数除了状态机，还是一个遍历器对象生成函数。返回的遍历器对象，可以依次遍历 Generator 函数内部的每一个状态。

形式上，Generator 函数是一个普通函数，但是有两个特征。一是，function关键字与函数名之间有一个星号；二是，函数体内部使用yield表达式，定义不同的内部状态（yield在英语里的意思就是“产出”）。Generator 函数的调用方法与普通函数一样，也是在函数名后面加上一对圆括号。不同的是，调用 Generator 函数后，该函数并不执行，返回的也不是函数运行结果，而是一个指向内部状态的指针对象。下一步，必须调用遍历器对象的next方法，使得指针移向下一个状态。也就是说，每次调用next方法，内部指针就从函数头部或上一次停下来的地方开始执行，直到遇到下一个yield表达式（或return语句）为止。换言之，Generator 函数是分段执行的，yield表达式是暂停执行的标记，而next方法可以恢复执行。

4. node.js nextTick setImmidate vue.js nextTick this.$nextTick

5. async.js

promise、generator 、async/await的原理解释

Promise，就是一个对象，用来传递异步操作的消息，避免了层层嵌套的回调函数。它代表了某个未来才会知道结果的事件（通常是一个异步操作），并且这个事件提供统一的API，可供进一步处理。

（1）对象的状态不受外界影响。有三种状态：Pending（进行中）、Resolved（已完成，又称Fulfilled）和Rejected（已失败）。

（2）一旦状态改变，就不会再变，任何时候都可以得到这个结果。Promise对象的状态改变，只有两种可能：从Pending变为Resolved和从Pending变为

Rejected。只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果

Generator函数，返回一个部署了Iterator接口的遍历器对象，用来操作内部指针。以后，每次调用遍历器对象的next方法，就会返回一个有着value和done两个属性的对象。value属性表示当前的内部状态的值，是yield语句后面那个表达式的值；done属性是一个布尔值，表示是否遍历结束。

async函数可以理解为Generator函数的语法糖，使用async内置了执行器，无需调用next方法进行逐步调用。且其返回值为Promise。

await命令后是一个Promise对象。如果不是，会被转成一个立即resolve的Promise对象。await只能用在async函数中，不能用在普通函数中