**Guide to asyncawait v1.0**

|  |
| --- |
| 翻译：董天逸 |

**1. 介绍**

asyncawait在Node.js JavaScript解决callback hell问题。灵感来自于C #异步/等特征， asyncawait使你写出现在每一个异步操作的功能块，等待结果再继续下面的语句。例如，您 可以用简单的JavaScript编写以下内容：

var foo = async (function() {

var resultA = await (firstAsyncCall());

var resultB = await (secondAsyncCallUsing(resultA));

var resultC = await (thirdAsyncCallUsing(resultB));

return doSomethingWith(resultC);

});

其中，有一个前提，是语义上等同于：

function foo2(callback) {

firstAsyncCall(function (err, resultA) {

if (err) { callback(err); return; }

secondAsyncCallUsing(resultA, function (err, resultB) {

if (err) { callback(err); return; }

thirdAsyncCallUsing(resultB, function (err, resultC) {

if (err) {

callback(err);

} else {

callback(null, doSomethingWith(resultC));

}

});

});

});

}

尽管该函数的同步外观，但它不会阻塞节点的事件循环。三个异步操作中的每个执行期间都会暂停执行， 但Node的事件循环可以执行其他代码，而这些操作正在等待。您可以在HTTP请求处理程序中编写上面的 示例那样的代码，并用许多同时连接实现高吞吐量，就像基于回调的异步处理程序一样。  
  
总之，asyncawait娶异步代码的高并发的视觉清晰度和同步代码的简洁性。而不是通过回调和错误的背 后，你可以返回值和使用try/catch块。而不需要专门的异步控制流结构，而每一样，你可以 用纯JavaScript构造如for和while循环。

**2. 特征/技巧总结**

* 消除回调意大利面条代码。
* 允许使用普通JavaScript控制流结构进行异步操作。
* 语法是纯JavaScript，和行为像C #异步/等待。
* 无缝互操作与大多数其他图书馆，包括快递、摩卡，下划线，蓝鸟，等。
* 快速轻量级。
* 完全无阻塞。
* 不需要ES6发生器。
* 没有代码预处理或特殊构建步骤，只需正常编写和执行代码即可。
* 节点光纤构建。
* TypeScript和x-to-javascript友好（因为不需要ES6发生器）。
* TypeScript typings嵌入式。
* 只有在Node.js，不是在浏览器（因为它使用节点纤维）。

**3. 那怎么使用呢？**

像CO，asyncawait可以暂停运行功能无阻塞节点的事件循环。图书馆建在协程，但使用不同的技术。 共使用6发电机，它工作在节点> = v0.11.2（与和谐的标志），并将被所有流行的JavaScript的环境 和工具，总有一天会被支持。  
  
asyncawait使用结纤维。它的工作原理与清楚的ES6/ ES5的JavaScript， 如果你的工具不支持ES6生成器是伟大的。这可能是一个重要的考虑因素时，使用JavaScript语言编写， 如TypeScript或CoffeeScript。  
  
通过在预处理步骤中转换JavaScript源代码，可以获得类似的结果。streamline.js是这种方法的一个例子。 代码使用asyncawait执行通常没有任何代码转换或预处理

**4. 相比…**

asyncawait代表之一在Node.js编写复杂的异步代码的几种可行的方法，以其独特的取舍。显著的替代 品包括异步、蓝知更鸟和有限，每个都有自己的权衡。下表总结了一些备选方案及其利弊。有关替代方案 比较的更多信息，请查看比较文件夹。  
  
asyncawait或许是个不错的选择（一）如果你需要高度并发的吞吐量，（b）你的异步代码必须清晰简 洁，代码（C）和（D）目标Node.js，你受限于沙三/ ES5的语法（例如你写在TypeScript或 CoffeeScript）。  
  


脚注：[ 1 ]每个同步调用阻塞节点的事件循环。所有并发任务都被阻塞，事件循环处于空闲状态， 直到调用完成为止。[ 2 ]平原回调迅速成为笨重复杂的异步任务。看的比较。[ 3 ]虽然比普通回调， 这些款式仍然产生更长和更复杂的代码比同步或协同的基础代码。看的比较。[ 4 ]一些工具不支持 （还）6发电机，包括编译为JavaScript语言如TypeScript和CoffeeScript。[ 5 ] ES6还有片状 的浏览器支持。[ 6 ]严格限于Node.js环境（即没有浏览器）由于节点利用纤维。

**5. 性能**

asyncawait如何执行？答案取决于你关心什么样的表现。作为一个粗略的指导，与裸露的回调相比， 期待你的代码是70%，少于66%的缩进短，奔跑在光秃秃的回调速度79%。好了，不要相信那些数字 （实际上是真的），但是请检查比较文件夹中的代码，并运行自己的基准。

**6. 快速启动**

**安装**

npm install asyncawait

**Async/Await 101**

asyncawait正好提供了两个功能：async()和await()。您可以使用这些代码引用这些函数：

var async = require('asyncawait/async');

var await = require('asyncawait/await');

使用异步宣布悬浮功能。在一个悬浮的功能，利用等待中止执行直到awaitable表达产生的结果。 awaitable表达式通常涉及执行异步操作。  
  
注意间隔后异步等待的例子。它们只是简单的函数，但空间使它们看起来更像关键字。 如果你真的想让他们站出来，你可以用像这样的名字**await**或等待，或任何适合你的。  
  
**基本的例子**

var async = require('asyncawait/async');

var await = require('asyncawait/await');

var Promise = require('bluebird');

var fs = Promise.promisifyAll(require('fs')); // adds Async() versions that return promises

var path = require('path');

var \_ = require('lodash');

/\*\* Returns the number of files in the given directory. \*/

var countFiles = async (function (dir) {

var files = await (fs.readdirAsync(dir));

var paths = \_.map(files, function (file) { return path.join(dir, file); });

var stats = await (\_.map(paths, function (path) { return fs.statAsync(path); })); // parallel!

return \_.filter(stats, function (stat) { return stat.isFile(); }).length;

});

// Give it a spin

countFiles(\_\_dirname)

.then (function (num) { console.log('There are ' + num + ' files in ' + \_\_dirname); })

.catch(function (err) { console.log('Something went wrong: ' + err); });

功能countfiles返回在某一目录下的文件数。找到这个号码，它必须执行多个异步操作 （使用fs.readdir和FS，STAT）。countfiles被声明为一个函数的定义里面悬浮用异步（…）。 当countfiles叫做一个目录的字符串，它开始执行异步和立即返回一个承诺的结果。在内部， countfiles似乎同步控制流。每次等待调用暂停执行，直到其参数生成一个结果，然后该结果将成 为等待调用的返回值。  
  
**更多例子** The examples文件夹包含更多的例子。比较几种examples文件夹also contains，每个编码在六个 不同风格（using平原callbacks，使用同步的唯一代码，使用异步图书馆，using the青鸟图书馆， using the公司图书馆，and using this asyncawait Library）。

**7. 异步深度：悬浮功能**

下面的小节涉及到下面的代码：

var suspendable = async (function defn(a, b) {

assert(...) // may throw

var result = await (...)

return result;

});

var suspendable2 = async.cps (function defn(a, b) {...});

var suspendable3 = async.thunk (function defn(a, b) {...});

var suspendable4 = async.result (function defn(a, b) {...});

**接受参数和返回值**   
  
悬浮功能可以接受参数。调用悬浮（1, 2）将调用defn（1, 2）。悬浮功能可以变长。他们报告相同数 量的定义（即suspendable.length和defn.length都返回2）。  
  
一个可中断函数的定义可能回来，有或没有价值，或它可能扔。返回没有值等于返回未定义。定义的函 的返回值是该悬挂功能的结果（见得到悬浮功能结果）。  
  
**处理错误和异常**  
  
一个可中断函数的定义可能抛出异常，直接或间接地。如果有任何的等待电话defn异步产生错误的结果， 错误将作为内部defn例外了。  
  
在一个悬浮的功能定义，异常可以使用普通的try/catch块。任何未处理的异常抛出在defn将成为可持 续的误差结果。  
  
**获得可持续的功能结果**  
  
一个悬浮功能异步执行的，所以它通常不能返回其结果（或抛出一个错误）直接。默认情况下，异步产生 回报的承诺悬浮功能。可持续的回报承诺与defn的返回值实现，或拒绝与defn的例外。还支持其他方式 传递结果/错误：

* 返回一个承诺：suspendable(1, 2).then(function (val) {…}, function (err) {…});
* acceptng节点式回调：suspendable2(1, 2, function (err, val) {…});
* 返回一个 lazily-executed thunk：suspendable3(1, 2)(function (err, val) {…});
* 直接返回值：try { var val = suspendable4(1, 2); } catch (err) {…}

注意，suspendable4只能来自另一个悬浮的功能称为。另外，它可以创建可悬浮的功能，comminucate 结果是多方面的，如接受回调函数并返回一个承诺。你可以使用async.mod功能实现。  
  
**保存这一上下文**  
  
当一个悬浮的函数被调用时，它的这种情况下是通过调用它的定义。例如，当悬浮。电话（myobj时，1, 2） 执行时，将被称为defn参数1和2，这个值myobj时。  
  
**创建和使用异步迭代器**  
  
异步函数可以用来创建异步迭代器。这些类似于6迭代器，除了那next()函数是一个悬浮功能服从所有的 规则在这个部分描述。async.iterable创建它返回一个迭代器的next()异步函数返回一个值的一个承诺， {，}结果了。  
  
异步迭代器有超过其价值的一个foreach()迭代方法。有关更多信息，请看descendentfilepaths.js和 iteration.js实例。  
  
**Eager versus Lazy Execution**  
  
调用一个悬浮功能如悬浮开始异步执行立即按照承诺正常语义。与此相反，咚返回悬浮功能没有开始执行 直到回调传递给咚。悬浮功能如suspendable3因此有懒惰的语义。  
  
**嵌套、组合和递归**  
  
悬浮功能可称为等待的表情，因为他们承诺（或thunk或值），因此awaitable。因此，调用悬浮功能可以 任意嵌套组成，可以递归。  
  
**The async.mod Function**  
  
异步函数的每一个变体（即异步，async.cps，async.iterable，等）有一个MOD方法接受一个对象并返回 一个选项功能变异。选项对象可以包含以下四个属性的任意组合：

{

returnValue: <string>; // Recognised values: 'none', 'promise', 'thunk', 'result'

acceptsCallback: <boolean>;

isIterable: <boolean>;

maxConcurrency: <number>; // Recognised values: falsy values and positive numbers

}

略性将从异步变被改装成继承他们的价值。例如，调用异步。MOD（{ acceptscallback：true}）和异步。 CPS。MOD（{中：'promise”}）是等价的。两个调用返回一个异步函数，可以用来创建，接受回调并返回 一个承诺悬浮功能。

**8. await在深度：awaitable表达式**

下面的小节涉及到下面的代码：

var suspendable = async (function () {

var promise1 = new Promise(.../\* eventually produces the value 'p1' \*/);

var promise2 = new Promise(.../\* eventually produces the value 'p2' \*/);

var thunk1 = function(callback) {.../\* eventually produces the value 't1' \*/});

var thunk2 = function(callback) {.../\* eventually produces the value 't2' \*/});

var thunk3 = ..., thunk4 = ...;

var r1 = await (promise1);

var r2 = await (thunk1);

var r3 = await (3.14);

var r4 = await ([promise2, 2, ['222', thunk2]]);

var r5 = await ({ t3: thunk3, t4: thunk4 });

return [r1, r2, r3, r4, r5];

});

**await有什么用？**  
  
等待需要一个参数，它必须是一个awaitable表达。一个awaitable表达可能有下列：

1. 一个承诺（或任何thenable对象），如VAR R1 =等待（生机勃勃）。悬浮的功能将暂停直到生机勃勃了。 承诺的分辨率值（'p1”）将成为等待调用的返回值，并分配给R1。如果生机勃勃是拒绝，拒绝价值将 被扔在悬浮的例外。
2. 一想到，在VAR R2 =等待（thunk1）。“咚thunk1将立即被调用，并悬挂将暂停直到控制返回到thunk的 回调。thunk的结果（'t1”）将成为等待调用的返回值，并分配给R2。如果thunk1返回一个错误，误差值 将被扔在悬浮的例外。
3. 一个原始值，如数字、字符串或NULL，如var中的等待（3.14）。等待调用将立即返回原始值，在这种情 况下，将值3.14分配给R3。
4. 一个数组或对象的所有元素均为平原，awaitables，如VAR R4 =等待 （[ promise2，2，[ ‘222 ’，thunk2 ] ]）。注意，这个定义是递归的，允许嵌套对象图。 悬浮的功能将暂停直到所有包含awaitables（promise2、2、222和thunk2）产生了自己的价值， 此时等待调用将返回与所有awaitable表达式的结果替换的对象图的一个克隆 （[ ‘p2 ’，2 [ '222”、“T2”] ]）。如果包含的awaitables产生误差，误差值将被扔在悬浮的例外。

注意，用多个参数等待调用（或带有零参数）相当于调用一个包含所有参数的数组。  
  
**获得awaitable版本的节点样式API**  
  
在传统的js代码，异步回调函数将作为最后一个参数，不返回任何值。因此，调用这些函数不awaitable。 然而，awaitable版本可能使用像鸟promisifyall()相对容易获得，或thunkify。  
  
**最大并发**  
  
串行执行一系列等待调用。例如，VaR R1 =执行等待（生机勃勃）是在VaR R2 =执行等待完成（thunk1） 开始。

相反，一个等待调用数组或对象的所有过程中平原的awaitables兼。例如，当声明变量R5 =等待 （{：thunk3 T3，T4：thunk4 }）都thunk3和thunk4叫做立即和异步任务的并发执行。  
  
图书馆如lodash强调互操作顺利，asyncawait，为生产阵列同时执行任务，并对结果消费阵列。  
  
**变化的等待**  
  
有几种不同的等待功能，替代行为的awaitable表达当数组或普通的对象。在一个应用实例awaittop.js 一看。

await.top(n)变体接受一个数n，恢复悬浮功能当第一N awaitable表达式包含在awaitable数组或对象 产生价值的平原。await.top(n)调用的返回值是一个数组，其中包含最快的n个结果，它们按顺序解析。

await.in变异就像是等待，但不克隆awaitable表达接待作为参数。所含awaitables结果代入放入原 awaitable阵列或普通的对象，这是等待返回值调用。

**9. 方法**

Handling HTTP Routes with Express  
Asynchronous Testing with Mocha

**10. API Reference**

**function async(fn: Function) –> (…args) –> Promise**

创建一个可以在每个异步操作中暂停的函数。FN包含函数体的悬浮。异步返回一个函数的形式（…args） –>承诺。传递给该函数的任何参数都传递给FN。返回的承诺在FN返回时解决，如果FN抛出，则拒绝。

**function async.cps(fn: Function) –> (…args, callback) –> void**

变异型异步产生悬浮功能，接受一个节点式的回调和返回什么。看得到悬浮功能结果。

**function async.thunk(fn: Function) –> (…args) –> Thunk**

变异型异步产生悬浮函数返回一个thunk。看得到悬浮功能结果。

**function async.result(fn: Function) –> (…args) –> any**

变异型异步产生悬浮功能，直接返回结果，而只能是来自另一个悬浮的功能称为。看得到悬浮功能结果。

**function async.iterable(fn: Function) –> (…args) –> AsyncIterator**

变异型异步产生的一个函数返回一个异步的迭代器，其next()法是一种可中断函数返回一个承诺。参见创建和使用异步迭代器。

**function async.mod(options) –> AsyncFunction**

实现的异步函数任意变量的创建。接受一个选择对象并返回一个异步功能变异。看到async.mod功能。

**function await(expr: Awaitable) –> Any**

摆一个悬浮功能直到awaitable表达式expr结果。结果变成等待调用的返回值。如果表达式产生一个错误，然后一个例外是在悬浮功能提高。

**function await.top(n: number) –> (expr: Array|Object) –> Array**

变异的结果等待由N快的解决awaitables包含在它的参数。见等待的变化。

**function await.in(expr: Array|Object) –> Array|Object**

变等待返回原数组和对象，而不是一个克隆的数组和对象，将结果包含awaitables到位。见等待的变化。

**11. 致谢**

asyncawait使用下面的技术：

* 结纤维：实现协程是有限的Node.js。6发电机可能更简单，但纤维更灵活，支持一个更为广泛的空间设 计的可能性。如果6发电机这个开放和灵活的就好了。
* 蓝鸟：这个承诺的图书馆是一个核心组成部分asyncawait和高性能JavaScript代码的写作灵感的源泉。
* TypeScript：asyncawait写的打字稿（看在src文件夹），并包括一个类型声明文件。TypeScript使 JavaScript开发更快、更不容易出错，更可扩展的，一般更愉快。
* lodash：下划线，但更好的。

**12. 许可证**

版权所有（C）gerwien 2014年至2015年。 此权限是理所当然的，免费的，任何人获得该软件的副本和相关的文档文件（“软件”）的交易软件， 在没有限制没有限制的权利，包括使用、复制、修改、发布、distribute，合并，复制和/或 sublicense，卖的软件许可证的人，和谁是一对提供的对软件的操作系统，在符合以下条件：

上述版权声明和本许可的通知应包括在所有副本的软件或大量的错。

该软件提供了“是”，没有任何一种服务，快递或暗示，包括但不限于《外merchantability， 健身和noninfringement特别用途。在事件的作者或版权持有人应承担责任的任何索赔的是， 损害赔偿或其他责任，无论是在一个动作的合同，侵权或其他产生的输出，或连接，或使用的软件或 其他软件中的dealings。