# 异步编程（pdf85页）

4.1 函数式编程

4.1.1 高阶函数

函数作为参数或者返回值来调用。如数组的sort方法。

4.1.2 偏函数用法

在函数里调用另一个函数。

4.2 异步编程的优点和难点

优点：解决文件IO问题，并且推荐所有的操作不要多于10ms，建议把大量的计算分成小块，调用setImmediate函数计算。

难点1：捕获错误失效。一般回到函数里会包含错误信息。

难点2：函数嵌套深。异步回调里嵌套了异步回调，导致异步IO没有体现出优势。

难点3：阻塞代码。因为node没有sleep函数，所以有可能会有while（time）的情况存在，不推荐。

难点4：node单进程没有很好得利用cpu的多核特性。

难点5：node中不容易进行同步调用。

4.3 异步编程方案

4.3.1 事件发布/订阅模式

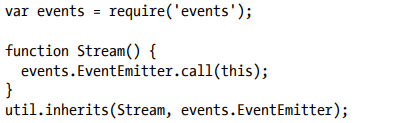
事件监听器又称发布/订阅模式，node中的事件没有冒泡和阻止默认事件，只有addListener/on() 、 once() 、 removeListener() 、

removeAllListeners()和emit()。[Demo1](demo1.js)。Node中很多模块都继承自它。

这个模式通常用来解耦业务逻辑，订阅事件的人无需关注监听器如何实现业务逻辑，数据通过消息方式可以灵活得传递。这里有两个细节点：

* 一个事件添加超过10个监听器，会得到一跳警告。一是设计者觉得10个以上容易造成内存泄漏，二是超过10个监听器容易造成浪费cpu情况。可以通过emitter.setMaxListeners(0)
* 如果EventEmitter对象出现error，就检查EventEmitter是否添加过error事件，如果没有则抛出错误。

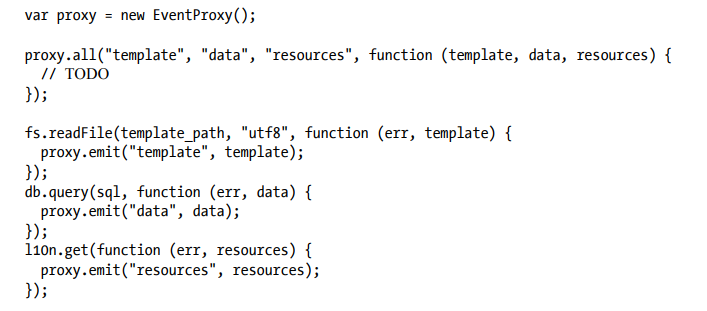
继承events模块。



多异步协作

一个函数需要在多个请求之后完成才执行，可以使用计数来判断当前回调是第几个，直到所有请求都完成。

也可以使用EventProxy，这是笔者自己写的npm模块



4.3.2 Promise/Deferred模式

Promises/A

* + Promise只有三种状态：未完成，完成，失败
  + Promise的状态只会从未完成->完成或失败，不可逆转，完成和失败不可相互转换。
  + Promise状态一但改变，不会更改。

Promises/A 中，一个Promise只要有then（）方法就行，then实现可以是继承events，实现如[demo2](demo2.js)。而then（）方法有如下规定：

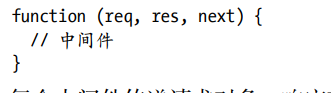
* 接收完成或失败的回调。
* 可选地支持progress作为第三种回调。
* 只接受function参数。
* 返回一个promise对象。

Promise序列化执行 [demo3](demo3.js)

1. 将所有的回调都ߚ存到队列中。
2. Promies完成时，依次执行回到，一但检测到返回了新的promise对象，就将队列转交给新的promise对象。

4.3.3 流程控制库

尾触发和next，类似promise的resolve和reject。



Async包

async提供了series（）方法来实现一组任务串行，依次执行。[Demo4](demo4.js)

async提供了parallel（）方法来实现一组任务并行。[Demo4](demo4.js)

series适合无依赖异步调用场景。

async提供了auto（）方法来实现任务自动依赖。[Demo4](demo4.js)

Step包

比async轻量，接口统一，只有step一个接口。