进程管理

refs:

<https://github.com/openstf/stf/blob/master/lib/util/procutil.js>

<https://alexzywiak.github.io/managing-multiple-child-processes-in-nodejs/index.html>

<http://www.bu.edu/tech/files/text/batchcode.txt>

<http://blog.argteam.com/coding/hardening-nodejs-production-process-supervisor/>

动态库调用封装ba

：<https://github.com/node-ffi/node-ffi>

<https://medium.com/@NorbertdeLangen/communicating-between-nodejs-processes-4e68be42b917>

场景

* 不同任务的进程协同启动

同时启动，或有先后顺序

require(‘co’)

* 相同任务的进程并行启动

同时启动，控制并行数量

require(‘bluebird’).map

将进程启动封装在一个promise对象中：

启动后进入unresolved状态，promise对象挂起。

进程执行完毕，正常退出，进入resolve状态，异常退出进入reject状态。

若启动失败进入reject状态。

使用child\_process模块中的fork来启动任务对应的js脚本。

需要处理的几个关键进程事件，ChildProcess对象上的事件：

* exit

ChildProcess.on(‘exit’, function(code, signal){

})

ChildProcess所关联的进程结束退出之后触发此事件，

进程正常退出，返回exitcode，则code参数为对应exitcode。

通过SIG结束的进程，将在signal参数接收到对应的SIG代码

Nodejs进程的内部实现中，包含了对SIGINT与SIGTERM的处理，进程在接收到这些信号之后并不会立即退出，而是会在执行与完进程相关的资源释放工作之后，才触发js脚本中所监听的SIGINT以SIGTERM事件。

* error

进程在一下三个场景会触发error事件

* + 无法启动
  + 无法kill
  + 父进程向子进程发送消息失败

该类型的消息直接在send中用callback处理掉，不要放在error里凑热闹。

时，

* close

进程关联的stdio stream关闭会触发此事件，stdio stream可能会被多个进程共享。

* disconnect

父进程中调用ChildProcess.disconnect，或在子进程中调用process.disconnect均会触发此事件，改事件触发后，ChildProcess对象上的connected属性转为false，父子进程之间无法在收发消息（message事件）

* message

子进程调用process.send发送json对象，或socket句柄给父进程时会触发此消息。

结束子进程

* 被动：既等待子进程执行完毕，自动退出
* 主动：

ChildProcess对象上的kill方法，用与给子进程发送SIG，默认参数为SIGTERM。如SIG消息无法送达，则触发ChildPrcess上的error事件。

子进程被封装进promise对象内，cancelable

监视进程退出

* Process对象

Process.on(“exit”)

Process.on(“SIGTERM”)

Process.on(“SIGINT”)

SIGTERM/SIGINT在非windows平台上均有默认的信号处理方法，在进程退出并返回退出代码128+signal代码之前，重置终端模式。

如果重写了默认信号处理方法，则会改变SIGTERM与SIGINT的默认行为

SIGTERM windows平台，不支持此信号。

SIGINT，CTRL+C产生该信号，终端处于raw 模式下不会产生此信号。

* ChildProcess对象

交互处理

进程对象Process与自进程对象ChildProcess上均包含一个send方法，send方法的内部实现为IPC。两进程均可在message消息处理方法上，上接收处理来自对方的消息。

收发

* + 标准JSON对象

包含key为cmd，且value以NODE为前缀的消息将被视为nodejs进程的内部消息，被框架代码吃了

* + Socket对象

<https://nodejs.org/docs/latest-v6.x/api/child_process.html>

Example：sending a server object

Example：sending a socket object

ChildProcess处的错误处理：

* + 在send方法中提供callback来处理消息

var flag = send(object, function(error){

})

包装成一个promise来调用

* + 监听ChildProcess上的error事件

ChildProcess.on(‘error’, function(){

})

* + 返回值：
    - true
    - false，IPC通道关闭，目标消息队列已满

process supervisor manage PIDs

underneath the same application

knowledges

终端模式（终端，伪终端字符设备，类unix系统）：输入到终端的字符如何被解析。

Cooked mode：经过预处理再提交给程序

Raw mode：特殊字符无预处理，直接提交给程序

Cbreak mode：介于cooked mode与raw mode之间的一种模式。

<https://en.wikipedia.org/wiki/Terminal_mode>