child_process

Node.js 全解 - 子进程

版权声明

联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03

目录

- 进程
- 线程
- Node.js 的进程控制
- Node.js 的线程控制

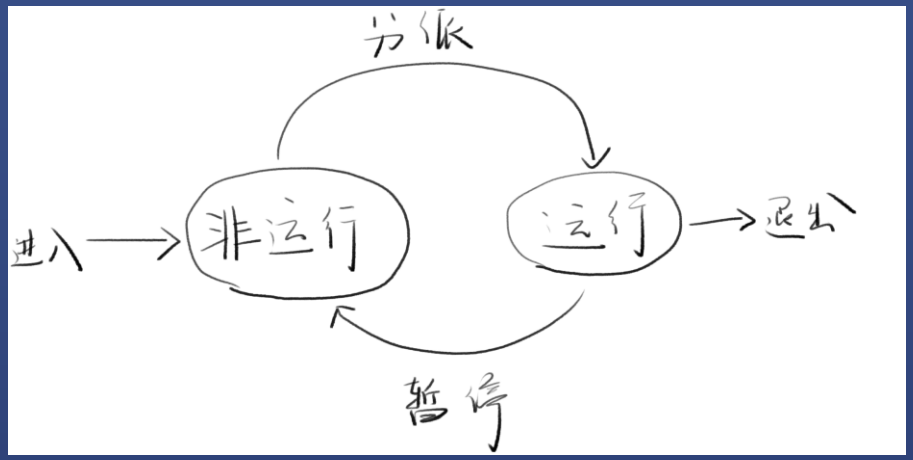
进程 Process

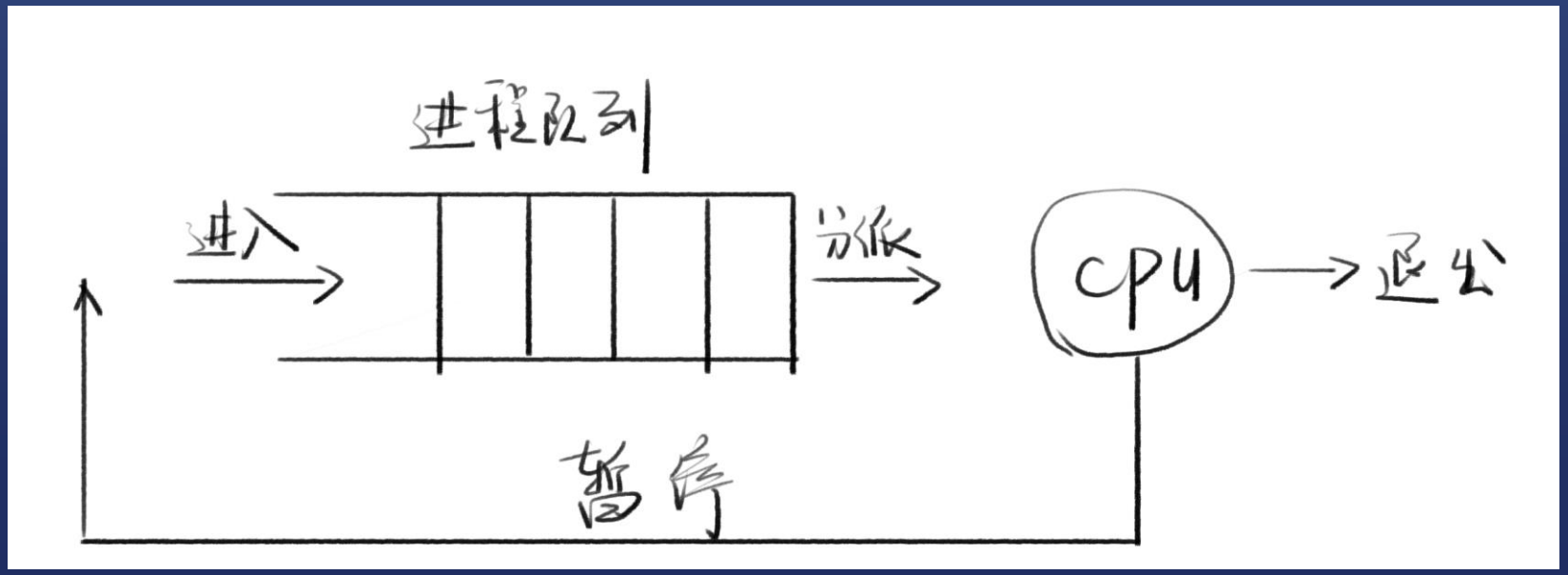
- 场景
  - notepad.exe 是一个程序，不是进程
  - 双击 notepad.exe 时，操作系统会开启一个进程
- 定义
  - 进程是程序的执行实例
  - 程序在CPU上执行时的活动叫做进程
  - 实际上并没有明确的定义，只有一些规则
- 特点
  - 一个进程可以创建另一个进程（父进程与子进程）
  - 通过任务管理器可以看到进程

了解 CPU

- 特点
  - 一个单核 CPU，在一个时刻，只能做一件事情
  - 那么如何让用户同时看电影、听声音、写代码呢?
  - 答案是在不同进程中快速切换
  - 此处以渣男交多个女朋友举例
- 多程序并发执行
  - 指多个程序在宏观上并行，微观上串行
  - 每个进程会出现「执行 - 暂停 - 执行」的规律
  - 多个进程之前会出现抢资源（如打印机）的现象

进程的两个状态

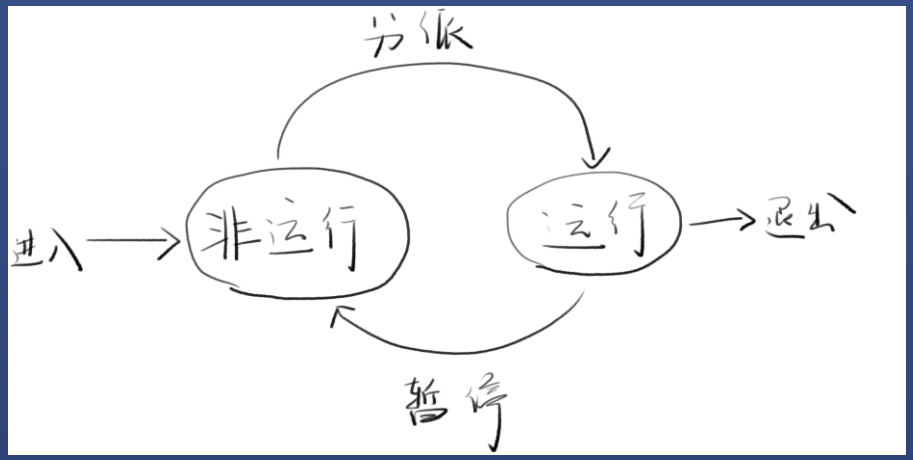


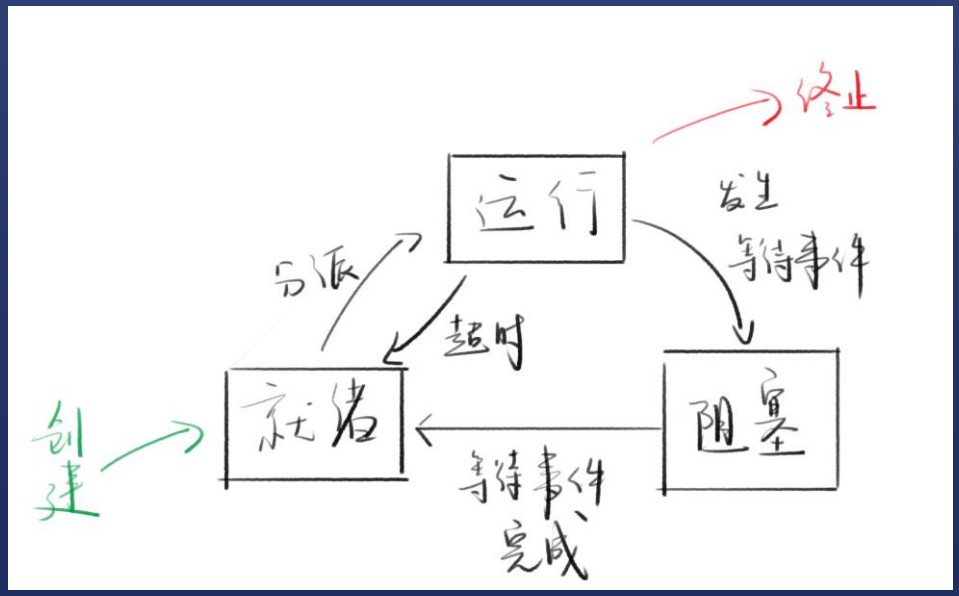


阻塞(se)

- 等待执行的进程中
  - 都是非运行态
  - 一些(A)在等待 CPU 资源
  - 另一些(B)在等待 I/O 完成（如文件读取）
  - 如果这个时候把 CPU 分配给 B 进程，B 还是在等 I/O
  - 我们把这个 B 叫做阻塞进程
  - 因此，分派程序只会把 CPU 分配给非阻塞进程

进程的三个状态





线程 Thread 的引入

- 分阶段
  - 在面向进程设计的系统中，进程是程序的基本执行实体
  - 在面向线程设计的系统中，进程本身不是基本运行单位，而是线程的容器
- 引入原因
  - 进程是执行的基本实体，也是资源分配的基本实体
  - 导致进程的创建、切换、销毁太消耗 CPU 时间了
  - 于是引入线程，线程作为执行的基本实体
  - 而进程只作为资源分配的基本实体
  - 此处可以以设计师和工程师分开招聘举例

线程 Thread

- 概念
  - CPU 调度和执行的最小单元
  - 一个进程中至少有一个线程，可以有多个线程
  - 一个进程中的线程共享该进程的所有资源
  - 进程的第一个线程叫做初始化线程
  - 线程的调度可以由操作系统负责，也可以用户自己负责
- 举例
  - 浏览器进程里面有渲染引擎、V8引擎、存储模块、网络模块、用户界面模块等
  - 每个模块都可以放在一个线程里
- 分析
  - 子进程 V.S. 线程

child_process

用于新建子进程

child_process

- 使用目的

✓ 子进程的运行结果储存在系统缓存之中（最大200Kb）

✓ 等到子进程运行结束以后，主进程再用回调函数读取子进程的运行结果

API

- exec(cmd, options, fn)
  - ✓ execute 的缩写，用于执行bash命令
  - ✓ 同步版本：execSync
- 流
  - ✓ 返回一个流（见下页）
- Promise
  - ✓ 可以使其 Promise 化（用 util.promisify）
- 有漏洞
  - ✓ 如果 cmd 被注入了，可能执行意外的代码
  - ✓ 推荐使用 execFile

使用流

```
var exec = require('child_process').exec;
var child = exec('ls -l');

child.stdout.on('data', function(data) {
  console.log('stdout: ' + data);
});
child.stderr.on('data', function(data) {
  console.log('stdout: ' + data);
});
child.on('close', function(code) {
  console.log('closing code: ' + code);
});
```

options

- 几个常用的选项
  - ✓ cwd - Current working directory
  - ✓ env - 环境变量
  - ✓ shell - 用什么 shell
  - ✓ maxBuffer - 最大缓存，默认 1024 * 1024 字节

API

- execFile
  - ✓ 执行特定的程序
  - ✓ 命令行的参数要用数组形式传入，无法注入
  - ✓ 同步版本：execFileSync
- 支持流吗?
  - ✓ 试一试就知道了

API

- spawn
  - ✓ 用法与execFile方法类似
  - ✓ 没有回调函数，只能通过流事件获取结果
  - ✓ 没有最大 200Kb 的限制（因为是流）
- 经验
  - ✓ 能用 spawn 的时候就不要用 execFile

API

- fork
  - 创建一个子进程，执行Node脚本
  - fork('./child.js') 相当于 spawn('node', ['./child.js'])
- 特点
  - 会多出一个 message 事件，用于父子通信
  - 会多出一个 send 方法
  - 见下页

代码

- n.js

```
var n = child_process.fork('./child.js');
n.on('message', function(m) {
  console.log('PARENT got message:', m);
});
n.send({ hello: 'world' });
```

- child.js

```
process.on('message', function(m) {
  console.log('CHILD got message:', m);
});
process.send({ foo: 'bar' });
```

那线程呢

刚才讲的都是进程啊

一些历史

- child_process.exec
  - ✓ v0.1.90 加入 Node.js
- new Worker
  - ✓ v10.5.0 加入 Node.js
  - ✓ v11.7.0 之前需要 --experimental-worker 开启
- 这个线程 API 太新了
  - ✓ 所以我们应该不会经常用到
- 效率
  - ✓ 目前效率并不够高，文档（中文）自己都写了

worker_threads API

- API 列表
  - ✓ isMainThread
  - ✓ new Worker(filename)
  - ✓ parentPort
  - ✓ postMessage
- 事件列表
  - ✓ message
  - ✓ exit

再见

如果你对进程、线程感兴趣推荐学习任意一本关于操作系统的教科书