## 安装swoole

手动安装与编译

参考：

<https://wiki.swoole.com/#/environment>

安装前必须保证系统已经安装了下列软件

* php-7.1 或更高版本
* gcc-4.8 或更高版本
* make
* autoconf

### linux安装gcc

**yum -y install gcc gcc-c++ autoconf pcre pcre-devel make automake**  
**yum -y install wget httpd-tools vim**

gcc –version 查看版本

**yum -y install gcc gcc-c++ pcre-devel make**

**编译：**

cd swoole-src && \

phpize && \

./configure && \

make && sudo make install

打开扩展

在 php.ini 中加入一行 extension=swoole.so 来启用 Swoole 扩展

### **pecl安装swoole**

#### 安装 PEAR

安装后，可以使用 pecl 命令来安装软件。可以通过 PHP 官方提供的 pecl 命令从 PHP 的官方扩展库安装扩展。

1 yum install php72w-pear

#### 安装 phpize

安装后，可以使用 phpize 命令从源码编译 PHP 扩展，得到 \*.so 文件。

1 yum install php72w-devel

#### 安装 swoole

1 pecl install swoole

参考：<https://www.cnblogs.com/phpk/p/10930471.html>

目前流行的两个laravel swoole扩展

两个扩展再github上都有文档

采用composer安装这两个扩展下载下来的文件和github上的项目一样

## [swooletw](https://github.com/swooletw)/[laravel-swoole](https://github.com/swooletw/laravel-swoole)

<https://github.com/swooletw/laravel-swoole>

## [hhxsv5](https://github.com/hhxsv5)/[laravel-s](https://github.com/hhxsv5/laravel-s)

<https://github.com/hhxsv5/laravel-s>

### 安装要求：



要先安装swoole

参考：

https://wiki.swoole.com/#/environment

### 安装

composer require "hhxsv5/laravel-s:~3.7.0" -vvv

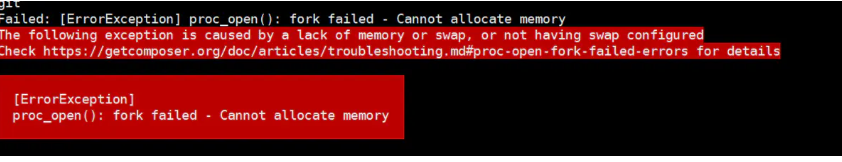
一开始我在本地windows安装，提示缺少

- hhxsv5/laravel-s v3.7.3 requires ext-pcntl \*

是因为本地没有这个扩展，有一种做法是忽略这个扩展我觉得不好

所以我们将项目放在linux去执行这个安装命令

在linux安装下报错



解决办法：

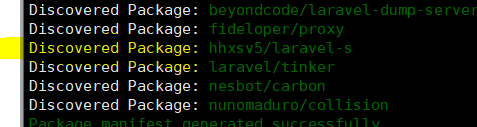
参考： <https://www.jianshu.com/p/d9f6bead033b>

dd if=/dev/zero of=/var/swap.1 bs=1M count=1024

mkswap /var/swap.1

swapon /var/swap.1

安装成功

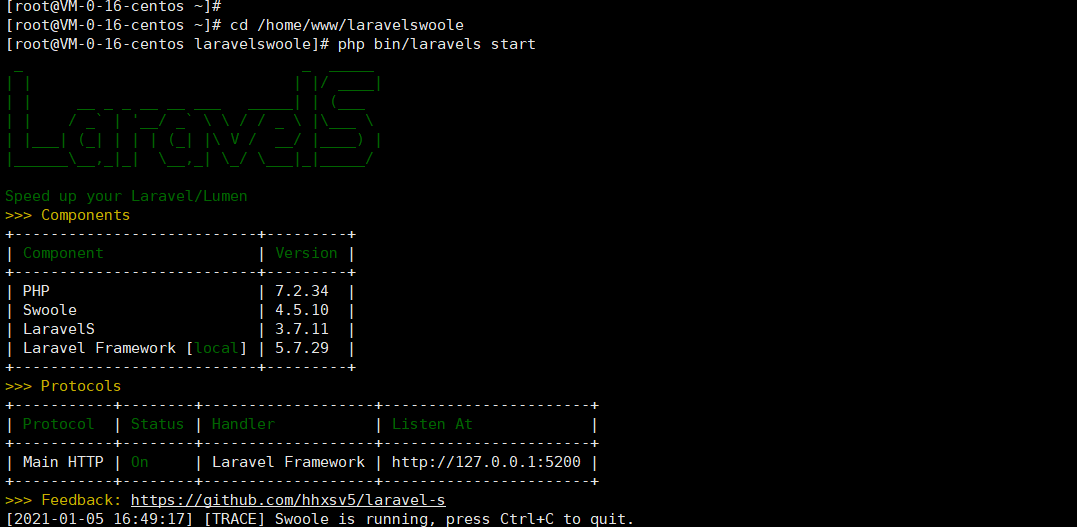


### 发布配置文件

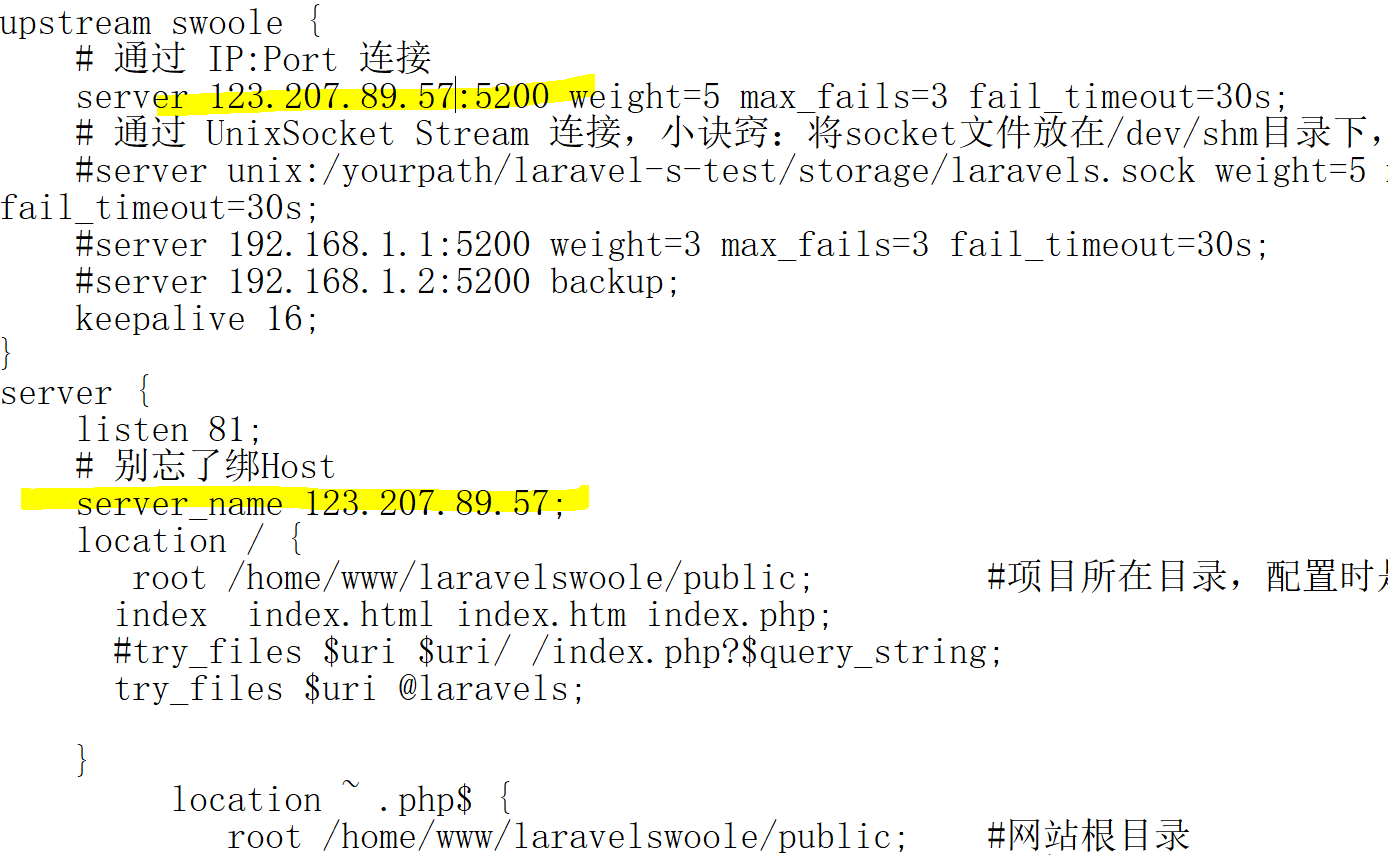
php artisan laravels publish

### 运行

php bin/laravels start



Nginx配置



设置一样会报错 502 Bad Gateway 上面的应该是127.0.0.1

### Supervisor守护进程

#### 安装supervisor

yum install supervisor

#### 设置开机自启动

编辑服务文件

# vim /usr/lib/systemd/system/supervisord.service

1

内容如下

[Unit]

Description=Supervisor daemon

[Service]

Type=forking

PIDFile=/var/run/supervisord.pid

ExecStart=/bin/supervisord -c /etc/supervisord.conf

ExecStop=/bin/supervisorctl shutdown

ExecReload=/bin/supervisorctl reload

KillMode=process

Restart=on-failure

RestartSec=42s

[Install]

WantedBy=multi-user.target

保存退出

Esc:x

1

启动服务

# systemctl enable supervisord

Systemctl start supervisord

#### 配置一个进程

编辑/etc/supervisord.conf文件

[program:laravel-echo-server]

process\_name=%(program\_name)s

directory=/home/www/wechatdemo5

command=laravel-echo-server start

autostart=true

autorestart=true

user=root

numprocs=1

redirect\_stderr=true

stdout\_logfile=/var/log/supervisor/laravel-echo-server.log

日志文件laravel-echo-server.log要自己建

[program:laravel-s-test]

directory=/home/www/laravelswoole

command=php bin/laravels start

autostart=true

autorestart=true

user=root

numprocs=1

redirect\_stderr=true

stdout\_logfile=/var/log/supervisor/laravels.log

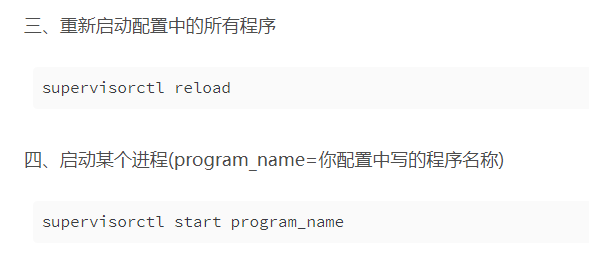
#### 启动：

supervisord -c /etc/supervisord.conf

或者输入supervisord      启动

关闭所有

supervisorctl shutdown



supervisorctl reload

supervisorctl start laravel-echo-server

基本命令<https://www.jianshu.com/p/c8071234f081>

监听某个端口

lsof -i:6001

一、开启命令

1. supervisor的服务器端部分启动命令：

* sudo unlink /var/run/supervisor.sock
* supervisord -c /etc/supervisor/supervisord.conf  
  此时默认开启了所有服务

1. supervisor的客户端部分命令：

* supervisorctl status 查看进程运行状态
* supervisorctl start 进程名 启动进程
* supervisorctl stop 进程名 关闭进程
* supervisorctl restart 进程名 重启进程
* supervisorctl update 重新载入配置文件
* supervisorctl shutdown 关闭supervisord
* supervisorctl clear 进程名 清空进程日志
* supervisorctl 进入到交互模式下。使用help查看所有命令。
* start stop restart + all 表示启动，关闭，重启所有进程。

二、关闭命令

supervisorctl stop all先关闭supervisor服务  
之后再关闭supervisord服务  
kill -9 pid

### Websocket使用

#### 1.创建websocket Handle类

**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2021/1/6  
 \* Time: 9:22  
 \*/***namespace** App\Services;  
**use** Hhxsv5\LaravelS\Swoole\WebSocketHandlerInterface;  
**use** Swoole\Http\Request;  
**use** Swoole\Http\Response;  
**use** Swoole\WebSocket\Frame;  
**use** Swoole\WebSocket\Server;  
*/\*\*  
 \** ***@see https://wiki.swoole.com/#/start/start\_ws\_server*** *\*/***class** WebSocketService **implements** WebSocketHandlerInterface  
{  
 *// 声明没有参数的构造函数* **public function** \_\_construct()  
 {  
 }  
 *// public function onHandShake(Request $request, Response $response)  
 // {  
 // 自定义握手：https://wiki.swoole.com/#/websocket\_server?id=onhandshake  
 // }* **public function** onOpen(Server $server, Request $request)  
 {  
 *// 在触发onOpen事件之前，建立WebSocket的HTTP请求已经经过了Laravel的路由，  
 // 所以Laravel的Request、Auth等信息是可读的，Session是可读写的，但仅限在onOpen事件中。  
 // \Log::info('New WebSocket connection', [$request->fd, request()->all(), session()->getId(), session('xxx'), session(['yyy' => time()])]);  
 // 此处抛出的异常会被上层捕获并记录到Swoole日志，开发者需要手动try/catch* $server->push($request->**fd**, **'Welcome to LaravelS'**);  
 }  
 **public function** onMessage(Server $server, Frame $frame)  
 {  
 *// \Log::info('Received message', [$frame->fd, $frame->data, $frame->opcode, $frame->finish]);  
 // 此处抛出的异常会被上层捕获并记录到Swoole日志，开发者需要手动try/catch* $server->push($frame->**fd**, *date*(**'Y-m-d H:i:s'**));  
 }  
 **public function** onClose(Server $server, $fd, $reactorId)  
 {  
 *// 此处抛出的异常会被上层捕获并记录到Swoole日志，开发者需要手动try/catch* }  
}

#### 更改配置文件config/laravels.php

**'websocket'** => [  
 **'enable'** => **true**,  
 **'handler'** => \App\Services\WebSocketService::***class***,  
 *// 'handler' => XxxWebSocketHandler::class,*],

**'swoole'** => [  
 *// 表示每60秒遍历一次，一个连接如果600秒内未向服务器发送任何数据，此连接将被强制关闭* **'heartbeat\_idle\_time'** => 600,  
 **'heartbeat\_check\_interval'** => 60,

// dispatch\_mode只能设置为2、4、5，https://wiki.swoole.com/#/server/setting?id=dispatch\_mode

'dispatch\_mode' => 2,

#### 3.Nginx配置websocket

map $http\_upgrade $connection\_upgrade {

default upgrade;

'' close;

}

upstream swoole {

# 通过 IP:Port 连接

server 127.0.0.1:5200 weight=5 max\_fails=3 fail\_timeout=30s;

# 通过 UnixSocket Stream 连接，小诀窍：将socket文件放在/dev/shm目录下，可获得更好的性能

#server unix:/yourpath/laravel-s-test/storage/laravels.sock weight=5 max\_fails=3 fail\_timeout=30s;

#server 192.168.1.1:5200 weight=3 max\_fails=3 fail\_timeout=30s;

#server 192.168.1.2:5200 backup;

keepalive 16;

}

server {

listen 80;

# 别忘了绑Host

#server\_name 123.207.89.57;

server\_name www.herry.club herry.club;

location / {

root /home/www/laravelswoole/public; #项目所在目录，配置时是相对 /nginx 这个路径

index index.html index.htm index.php;

#try\_files $uri $uri/ /index.php?$query\_string;

try\_files $uri @laravels;

}

location ~ .php$ {

root /home/www/laravelswoole/public; #网站根目录

fastcgi\_pass 127.0.0.1:9000; #Nginx通过本机的9000端口将PHP请求转发给PHP-FPM进行处理。

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

include fastcgi\_params; #Nginx调用fastcgi接口处理PHP请求。

}

# 当请求PHP文件时直接响应404，防止暴露public/\*.php

#location ~\* \.php$ {

# return 404;

#}

# Http和WebSocket共存，Nginx通过location区分

# !!! WebSocket连接时路径为/ws

# Javascript: var ws = new WebSocket("ws://laravels.com/ws");

location =/ws {

# proxy\_connect\_timeout 60s;

# proxy\_send\_timeout 60s;

# proxy\_read\_timeout：如果60秒内被代理的服务器没有响应数据给Nginx，那么Nginx会关闭当前连接；同时，Swoole的心跳设置也会影响连接的关闭

# proxy\_read\_timeout 60s;

proxy\_http\_version 1.1;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Real-PORT $remote\_port;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_set\_header Scheme $scheme;

proxy\_set\_header Server-Protocol $server\_protocol;

proxy\_set\_header Server-Name $server\_name;

proxy\_set\_header Server-Addr $server\_addr;

proxy\_set\_header Server-Port $server\_port;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection $connection\_upgrade;

proxy\_pass http://swoole;

}

location @laravels {

# proxy\_connect\_timeout 60s;

# proxy\_send\_timeout 60s;

# proxy\_read\_timeout 60s;

proxy\_http\_version 1.1;

proxy\_set\_header Connection "";

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Real-PORT $remote\_port;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header Host $http\_host;

proxy\_set\_header Scheme $scheme;

proxy\_set\_header Server-Protocol $server\_protocol;

proxy\_set\_header Server-Name $server\_name;

proxy\_set\_header Server-Addr $server\_addr;

proxy\_set\_header Server-Port $server\_port;

proxy\_pass http://swoole;

}

}

#### 4.控制器推送数据

*//websocket 控制器推送数据* **public function** pushMessage(){  
  
 $fd=1;  
 $swoole=app(**'swoole'**);  
 $success = $swoole->push($fd, **'Push data to fd#1 in Controller'**);*//返回bool值  
// return '132';  
 var\_dump*($success);  
 }

5.

#### 关于$request->fd的问题

$fd是客户端连接的唯一表示符

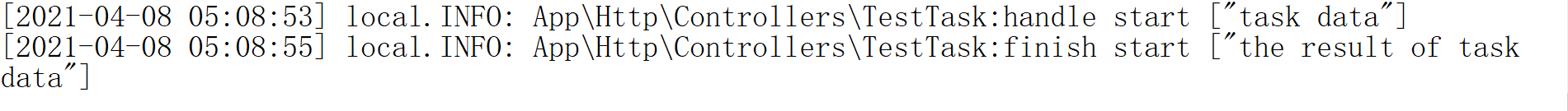
### 异步任务

#### 1.创建任务类

**namespace** App\Http\Controllers;  
**use** Hhxsv5\LaravelS\Swoole\Task\Task;  
**use** Illuminate\Support\Facades\Log;  
  
**class** TestTask **extends** Task{  
  
 **private $data**;  
 **private $result**;  
 **public function** \_\_construct($data)  
 {  
 $this->**data** = $data;  
 }  
 *// 处理任务的逻辑，运行在Task进程中，不能投递任务* **public function** handle()  
 {  
 Log::*info*(***\_\_CLASS\_\_*** . **':handle start'**, [$this->**data**]);  
 *sleep*(2);*// 模拟一些慢速的事件处理  
 // 此处抛出的异常会被上层捕获并记录到Swoole日志，开发者需要手动try/catch* $this->**result** = **'the result of '** . $this->**data**;  
 }  
 *// 可选的，完成事件，任务处理完后的逻辑，运行在Worker进程中，可以投递任务* **public function** finish()  
 {  
 Log::*info*(***\_\_CLASS\_\_*** . **':finish start'**, [$this->**result**]);  
*// Task::deliver(new TestTask2('task2')); // 投递其他任务* }  
}

#### 2.投递任务

$task = **new** TestTask(**'task data'**);  
*var\_dump*(**"aa"**);  
*// $task->delay(3); // 延迟3秒投递任务  
// $task->setTries(3); // 出现异常时，累计尝试3次*$ret = Task::*deliver*($task);  
*var\_dump*($ret);*// 判断是否投递成功*



### 毫秒定时器

# 在larvavel中使用原生swoole

在swoole官方文档中，可以看到swoole可以作为一个单独的框架运行，因为像laravelS这样的别人集成好的包别人也是对原生的swoole做了处理，所以下面要来尝试用原生的swoole来造一个轮子

## 自定义artisan命令运行swoole

参考别人的资料：<https://www.cnblogs.com/matoo/p/5569690.html>

独立的swoole框架中 我们在server.php创建该服务，需要运行php server.php 运行，显然在laravel中是行不通的，所以要自定义artisan命令

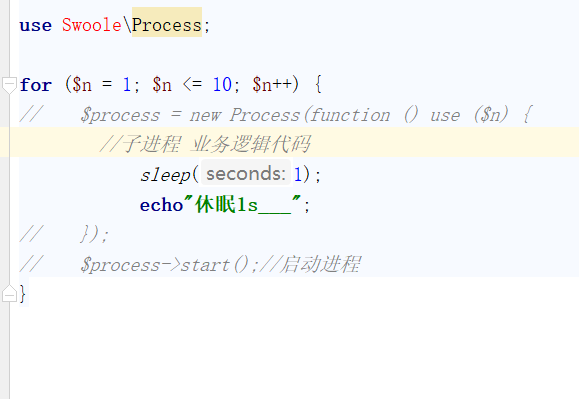
## laravel使用原生swoole造轮子

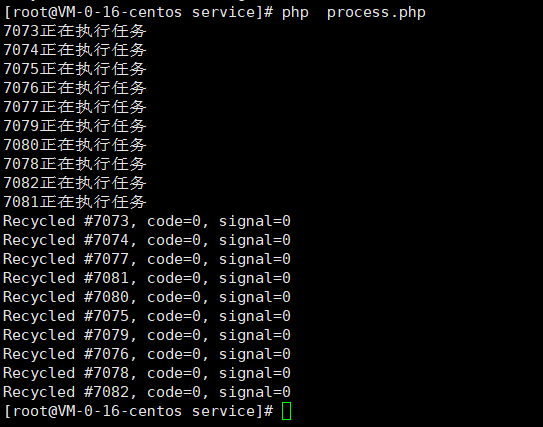
swoole可以作为框架单独运行，因此我们可以将swoole构造tcp,udp服务器的代码放在laravel中然后去线上服务器启动。

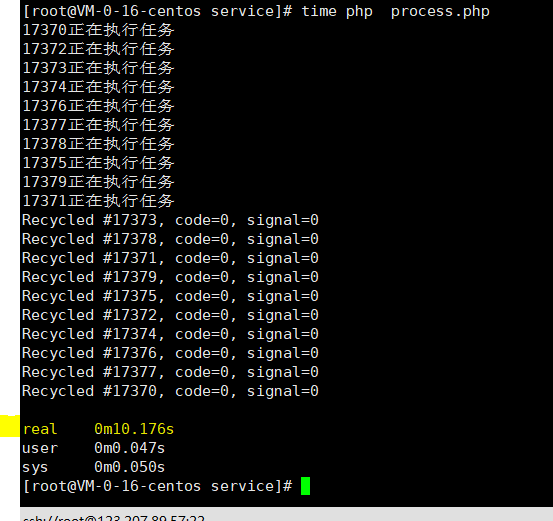
那么如何向laravelS一样把swoole做成一个扩展呢

## 进程

不使用进程，for循环中sleep1s要十秒执行结束，使用进程1s结束









Pstree –ap|grep php

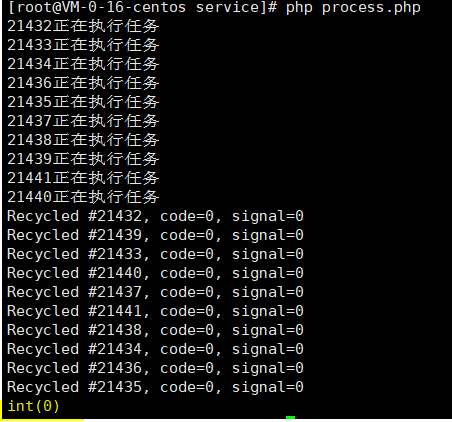
### Swoole创建进程

### Swoole进程的封装

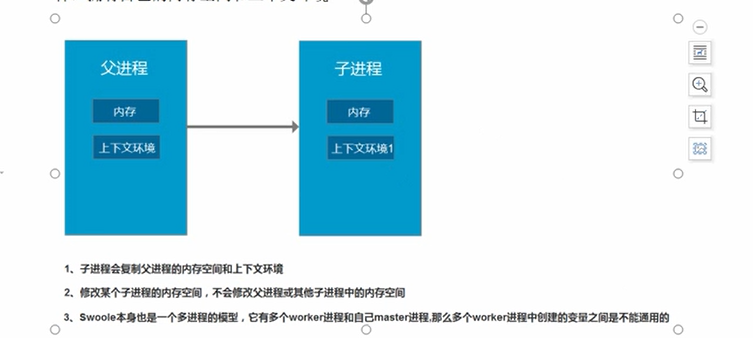
### 进程间通讯(IPC)

*//外面的代码是主进程，process是子进程，是独立的内存空间*

**use** Swoole\Process;  
$a=0;  
**for** ($n = 1; $n <= 10; $n++) {  
 $process = **new** Process(**function** () **use** (&$a) {  
 *//子进程 业务逻辑代码  
 sleep*(10);  
 $a++;  
 **echo** *getmypid*().**"正在执行任务"**.***PHP\_EOL***;  
 });  
 $process->start();*//启动进程*}  
  
*//回收子进程***while**( $status = Process::*wait*(**true**)){  
 **echo "Recycled #**{$status[**'pid'**]}**, code=**{$status[**'code'**]}**, signal=**{$status[**'signal'**]}**"** . ***PHP\_EOL***;  
}  
*var\_dump*($a);



为了能让进程之间共享变量，进程之间据需要互相通讯



进程间通信指至少两个进程或线程间传递数据或信号的一些技术或方法，每个进程都有自己的一部分独立的系统资源，彼此是隔离的，为了能使不同的进程相互访问资源并进行协调工作，才有了进程间通信。

#### 进程通信目的

数据传输：一个进程需要将他的数据发送给另一个进程，发送的数据量在一个字节到几个M之间

共享数据：多个进程想要操作共享数据，一个进程对数据的修改，其他进程应该立刻看到

进程控制：有些进程希望完全控制另一个进程的执行（如debug进程），此时控制进程希望能够拦截另一个进程的所有异常，并能够及时到他的状态。

#### 进程通讯的方式

管道、命名管道、消息队列、信号、信号量、共享内存、套接字（socket）

##### 消息队列

