官方文档:

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

<https://www.jianshu.com/p/5ab90b7bb250>

## 什么是swoole 一个面向生产环境的 [PHP](https://baike.baidu.com/item/PHP/9337) 异步，并行，高效的网络通信引擎，使 PHP 开发人员可以编写高性能的异步并发 TCP、UDP、Unix Socket、HTTP，WebSocket 服务

Swoole 是一个使用 c、C++ 语言编写的基于异步事件驱动和协程并行网络通信引擎

Swoole面试题

<https://blog.csdn.net/qq_40796396/article/details/101421585>

网络调试工具下载

## Php的FPM

### 什么是php-fpm

Php-fpm(FastCGI Process Manager ,FastCGI是一个进程管理器)是一个FastCGI管理器

### 作用与好处

## 1.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：**

**Swoole的安装可以根据官网上github地址 wget下载解压安装 然后执行下面命令**

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

宝塔添加扩展

extension=/www/server/php/73/lib/php/extensions/no-debug-non-zts-20180731/swoole.so

报Cannot find php-config. Please use --with-php-config=PATH





./configure --with-php-config=/www/server/php/73/bin/php-config && \

编译完成之后重启php服务

### **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。

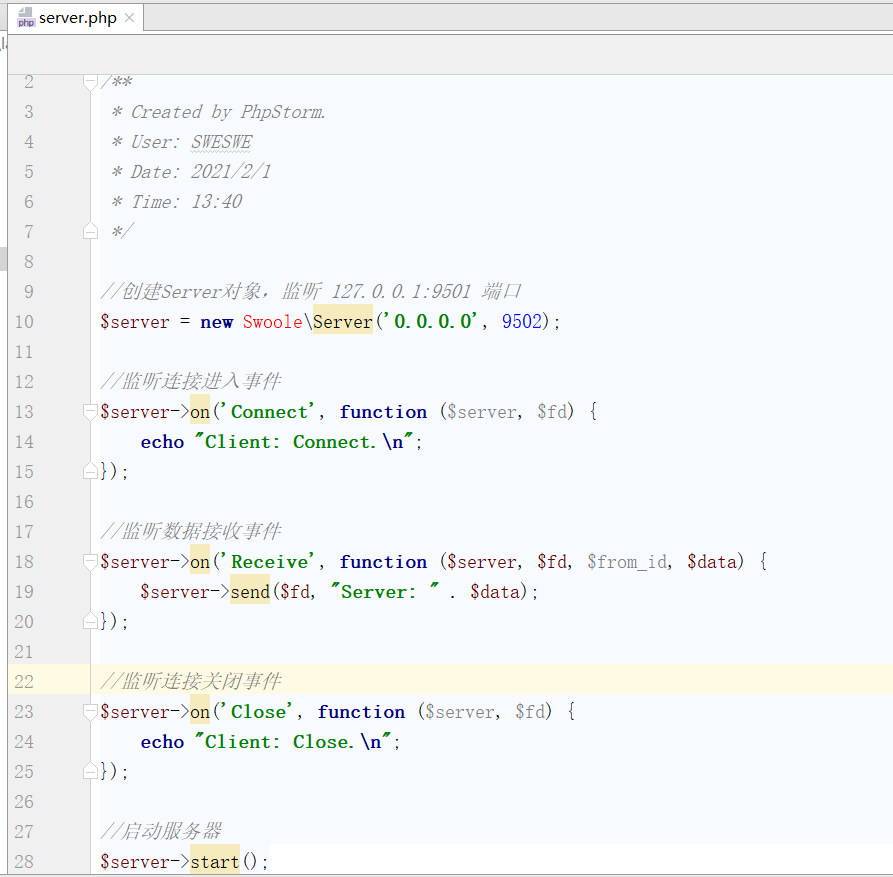
1 yum install php72w-devel

**安装 swoole**

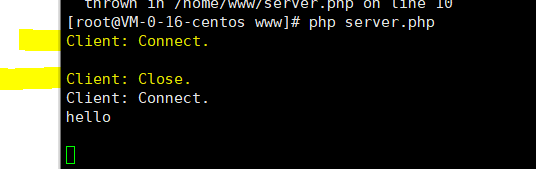
1 pecl install swoole

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

安装成功之后可以参考文档建立tcp服务的案例



然后启动swoole



## Swoole核三大功能

常用进程、异步、协程（处理异步io）

## 服务端

### Tcp服务端

#### 完整事例：

*//创建server对象 ,0.0.0.0开放给任意客户端，开放给具体某个客户端，填写其ip,开放给内网服务ip为127.0.0.1*$server=**new** Swoole\Server(**'0.0.0.0'**,9503);  
*//混合udp\tcp，同时监听内网和外网端口  
  
//设置运行时参数  
//$server->set(array(  
// 'daemonize'=>true,//程序将转入后台作为守护进程运行。长时间运行的服务器端程序必须启用此项 默认为false  
//));  
  
//注册事件回调*$server->on(**'Connect'**,**function**($server,$fd){  
 **echo"连接服务端成功\n"**;  
  
});  
  
$server->on(**'Receive'**,**function**($server,$fd,$form\_id,$data){  
 **echo "接收客户端数据:**{$data}**\n"**;  
 $server->send($fd,**"sever**{$data}**"**);  
});  
$server->on(**'Close'**,**function**($server,$fd){  
 **echo "服务端关闭"**;  
});  
  
*//启动服务器*$server->start();

#### 1.1创建server对象

*//创建server对象 ,0.0.0.0开放给任意客户端，开放给具体某个客户端，填写其ip,开放给内网服务ip为127.0.0.1*$server=**new** Swoole\Server(**'0.0.0.0'**,9503);

$server = new \Swoole\Server(string $host, int $port = 0, int $mode = SWOOLE\_PROCESS, int $sockType = SWOOLE\_SOCK\_TCP);

$host ip

$port 端口

$mode [SWOOLE\_PROCESS](https://wiki.swoole.com/#/learn?id=swoole_process) 多进程模式（默认），[SWOOLE\_BASE](https://wiki.swoole.com/#/learn?id=swoole_base) 基本模式

$server server类型

SWOOLE\_TCP/SWOOLE\_SOCK\_TCP tcp ipv4 socket

SWOOLE\_TCP6/SWOOLE\_SOCK\_TCP6 tcp ipv6 socket

SWOOLE\_UDP/SWOOLE\_SOCK\_UDP udp ipv4 socket

SWOOLE\_UDP6/SWOOLE\_SOCK\_UDP6 udp ipv6 socket

[SWOOLE\_UNIX\_DGRAM](https://github.com/swoole/swoole-src/blob/master/examples/unixsock/dgram_server.php) unix socket dgram

[SWOOLE\_UNIX\_STREAM](https://github.com/swoole/swoole-src/blob/master/examples/unixsock/stream_server.php) unix socket stream

#### 1.2set()设置参数

Swoole\Server->set(array $setting): void

$server->set(array(

'reactor\_num' => 2, // reactor thread num

'worker\_num' => 4, // worker process num

'backlog' => 128, // listen backlog

'max\_request' => 50,

'dispatch\_mode' => 1,

));

配置参数意义后面在将

#### 1.3on()注册事件回调

Swoole\Server->on(string $event, mixed $callback): void

在swoole中on方法用来执行定义好的一些函数，event是那些方法名 callback是回调函数

但是swoole的源码是如何实现on方法的我暂时还没看到

重复调用on方法会 覆盖上一次设定

$event 事件名称 名称首字母要大写

$callback 回调函数

具体有哪些事件回调参考下面的swoole事件，事件名称字符串不要加 on

#### 服务端运行结果



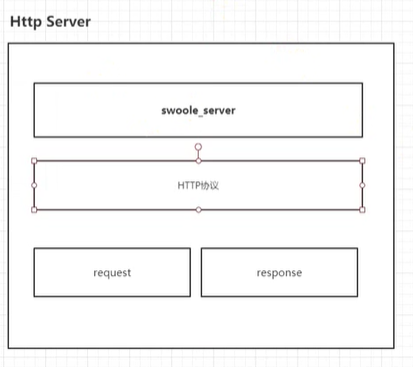
### UDP服务端

#### 完整事例

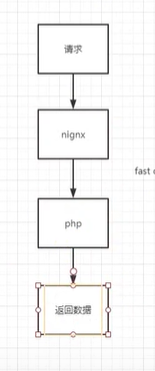
*//udp服务端*$server=**new** Swoole\Server(**'0.0.0.0'**,9504,***SWOOLE\_PROCESS***,***SWOOLE\_SOCK\_UDP***);  
  
*//监听数据接收事件  
//udp没有连接的概念，因此启动服务后，客户端无需Connect,可以直接向server监听的端口发送数据包，对应的事件为onPacket*$server->on(**'Packet'**, **function** ($server, $data, $clientInfo) {  
 **echo '接收客户端数据:'**.$data;  
 *//基于ip+端口发送信息* $server->sendto($clientInfo[**'address'**], $clientInfo[**'port'**], **"Server：**{$data}**"**);  
});  
  
*//启动服务器*$server->start();

### http服务器

#### 组成



传统的http服务器流程：



一个请求过来 会先到nignx然后通过fast cgi 发送到server通过FPM处理php代码然后返回数据

#### http\Reuqest

http请求对象，保存了http客户端请求的相关信息，包括get、post、cookie、Header等。

##### Header

http请求的头部信息，数组类型

heard数组的打印内容

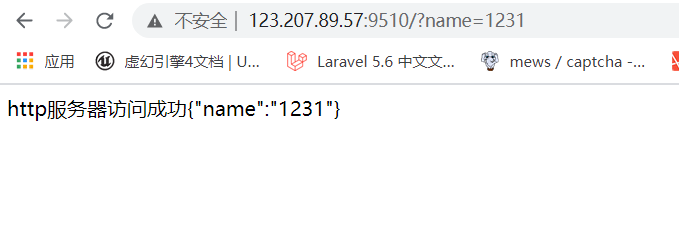


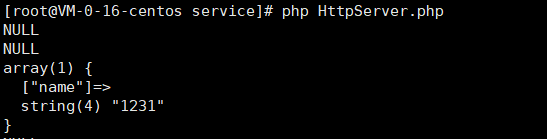
##### Get

echo $request->get['hello'];

// 获取所有GET参数

var\_dump($request->get);

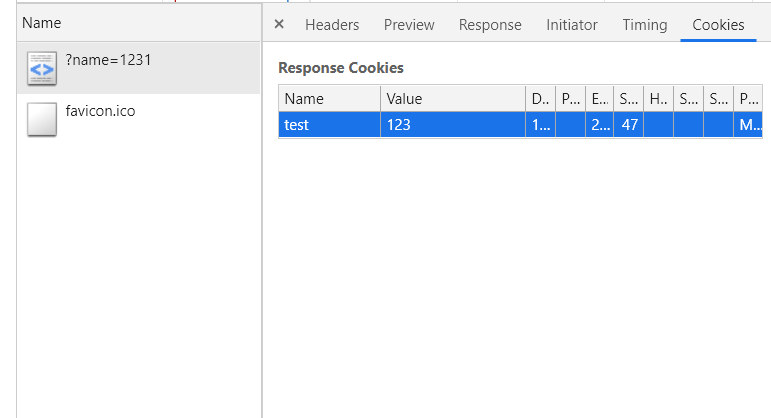




##### Post

##### Cookie

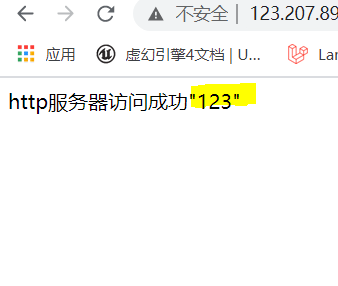
###### 设置cookie



$response->cookie(**'test'**,123,*time*()+1800);

###### 获取cookie

$request->**cookie**[**'test'**];



还有其他参数

参考文档：https://wiki.swoole.com/#/http\_server?id=cookie

### websocket服务器

##### 完整实例







##### onopen

onOpen(Swoole\WebSocket\Server $server, Swoole\Http\Request $request);

参数

$request 是一个 [**HTTP**](https://wiki.swoole.com/#/http_server?id=httprequest) 请求对象，包含了客户端发来的握手请求信息

onOpen 事件函数中可以调用 [**push**](https://wiki.swoole.com/#/websocket_server?id=push) 向客户端发送数据或者调用 [**close**](https://wiki.swoole.com/#/server/methods?id=close) 关闭连接

onOpen 事件回调是可选的

**当 WebSocket 客户端与服务器建立连接并完成握手后会回调此函数。**

*//客户端与服务器建立连接回调此函数*$websocket->on(**'open'**,**function** (){  
  
});

##### [onMessage](https://wiki.swoole.com/#/websocket_server?id=onmessage)

onMessage(Swoole\WebSocket\Server $server, Swoole\WebSocket\Frame $frame)

参数

$frame 是 **[Swoole\WebSocket\Frame](https://wiki.swoole.com/" \l "/websocket_server?id=swoolewebsocketframe)** 对象，包含了客户端发来的数据帧信息

onMessage 回调必须被设置，未设置服务器将无法启动

客户端发送的 ping 帧不会触发 onMessage，底层会自动回复 pong 包，也可设置 **[open\_websocket\_ping\_frame](https://wiki.swoole.com/" \l "/websocket_server?id=open_websocket_ping_frame)**参数手动处理



**当服务器收到来自客户端的数据帧时会回调此函数。**

*//当服务器收到客户端数据会回调此函数*$websocket->on(**'message'**,**function** (){  
  
});

##### Push

**向 WebSocket 客户端连接推送数据，长度最大不得超过 2M。**

##### Websocket服务器优化

将websocket服务器代码变成一个类

### Task使用

#### 使用场景

执行耗时操作，发送邮件、广播。如果直接去执行这些函数就会阻塞当前进程，导致服务器响应变慢。Swoole提供了异步任务处理的功能，可以投递一个异步任务到taskworker进程中执行，不影响当前的处理速度。

#### 方法

##### onTask()

function onTask(Swoole\Server $server, int $task\_id, int $src\_worker\_id, mixed $data);

在task进程内被调用，worker进程可以使用task函数向task\_worker进程投递新的任务。当前Task进程在调用onTask回调函数时会将进程状态切换成忙碌，这是不再接收新的task, **当**[onTask](https://wiki.swoole.com/" \l "/server/events?id=ontask)**函数返回时会将进程状态切换为空闲然后继续接收新的**Task**。**

**参数：**

Swoole\Server $server server对象

$task\_id 执行的task进程的id $task\_id 和worker\_id组合起来才是唯一的，不同的work进程投递的任务id可能会相同。

$data 任务的数据内容

##### onFinish

function onFinish(Swoole\Server $server, int $task\_id, mixed $data)

此回调函数在worker进程被调用，当worker进程投递的任务在task进程中完成是，task进程会通过swoole\server->finish()方法将任务处理的结果发送给worker进程

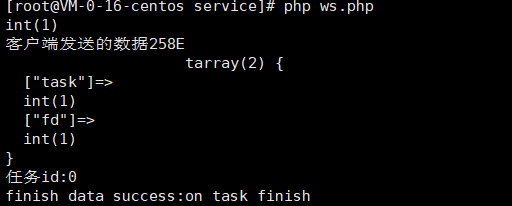
$server server对象

$task\_id 任务id

$data 返回的数据

#### 实例

**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2021/3/25  
 \* Time: 14:58  
 \*/***class** ws  
{  
 **protected $ws**;  
 **public function** \_\_construct()  
 {  
 *//new swoole 对象* $this->**ws**=**new** Swoole\Websocket\Server(**'0.0.0.0'**,9510);  
  
 *//设置进程参数* $this->**ws**->set([  
 **'worker\_num'**=>2,*//worker进程* **'task\_worker\_num'**=>4*//task进程* ]);  
 *//监听打开连接事件* $this->**ws**->on(**'open'**,[$this,**'onOpen'**]);*//[$this,'onOpen']等价于$this->>onOpen()  
 //监听消息连接事件* $this->**ws**->on(**'message'**,[$this,**'onMessage'**]);*//关于swoole回调函数的执行  
 //监听task事件* $this->**ws**->on(**'task'**,[$this,**'onTask'**]);*//  
 //监听finish事件* $this->**ws**->on(**'finish'**,[$this,**'onFinish'**]);*//  
 //监听连接关闭* $this->**ws**->on(**'close'**,[$this,**'onClose'**]);  
 *//启动服务器* $this->**ws**->start();  
 }  
 **public function** onOpen($ws,$request){  
 *var\_dump*($request->**fd**);  
 }  
 **public function** onMessage($ws,$frame){  
 **echo "客户端发送的数据**{$frame}**"**;  
 *//发送一条数据* $data=[  
 **'task'**=>1,  
 **'fd'**=>$frame->**fd**,  
 ];  
 *//耗时10s* $ws->task($data);  
 $ws->push($frame->**fd**,**'客户端连接服务器成功'**);  
 }  
  
 **public function** onTask($ws,$task\_id,$worker\_id,$data){  
 *var\_dump*($data);  
 *sleep*(10);  
 **return "on task finish"**;*//告诉worker进程task已完成* }  
 *//ontsak方法的返回值 不是上面的data数组* **public function** onFinish($ws,$task\_id,$data){  
 **echo"任务id:**{$task\_id}**\n"**;  
 **echo "finish data success:**{$data} **\n"**;  
 }  
 **public function** onClose($ws,$fd){  
 **echo"客户端:**{$fd}**关闭"**;  
 }  
}  
$object=**new** ws();

onTask()方法中执行了10s的耗时任务，可以看到 onmessage()中websocket客户端之前输出**客户端连接服务器成功**

10s后onfinish（）中的数据才输出

### 定时器

毫秒精度的定时器。底层基于 epoll\_wait 和 setitimer 实现，数据结构使用最小堆，可支持添加大量定时器。

在同步 IO 进程中使用 setitimer 和信号实现，如 Manager 和 TaskWorker 进程

在异步 IO 进程中使用 epoll\_wait/kevent/poll/select 超时时间实现

#### 性能

底层使用最小堆数据结构实现定时器，定时器的添加和删除，全部为内存操作，因此性能是非常高的。

**官方的基准测试脚本 [timer.php](https://github.com/swoole/benchmark/blob/master/timer.php" \t "_blank) 中，添加或删除 10 万个随机时间的定时器耗时为 0.08s 左右**

#### 方法

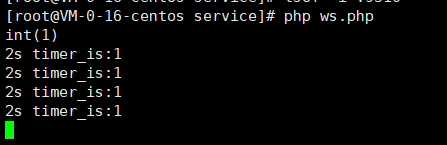
##### tick()

tick()设置一个时间闹钟定时器，与after定时器不同的是tick定时器会持续触发，知道调用timer::clear清除

Swoole\Timer::tick(int $msec, callable $callback\_function, ...$params): int

1. 定时器仅在当前进程空间内有效
2. 定时器是纯异步实现的，不能与同步IO的函数一起使用 ，否则定时器的执行时间会发生错乱
3. 定时器在执行的过程中可能存在一定的误差

**if**($request->**fd**==1){  
 swoole\_timer\_tick(2000,**function**($timer\_id){  
 **echo "2s timer\_is:**{$timer\_id}**\n"**;  
 });  
}



##### after()

在指定的时间后执行函数。Swoole\Timer::after 函数是一个一次性定时器，执行完成后就会销毁。

swoole\_timer\_after(5000,**function** ()**use**($ws,$frame){  
 **echo "5s after \n"**;  
 $ws->push($frame->**fd**,**'定时器发送消息'**);  
});

Swoole\Timer::after(int $msec, callable $callback\_function, ...$params): int

##### clear()

### 进程

Swoole提供的进程管理模块，用来代替pcntl

#### Php自带的pcntl

Pcntl是一个可以利用操作系统的fork系统调用在php中实现多线程的进程控制扩展，当使用fork系统调用后执行的代码会是并行的。Pcntl仅适用与linux平台的cli模式下使用

Php自带的pcntl缺点：

没有提供进程间的通信

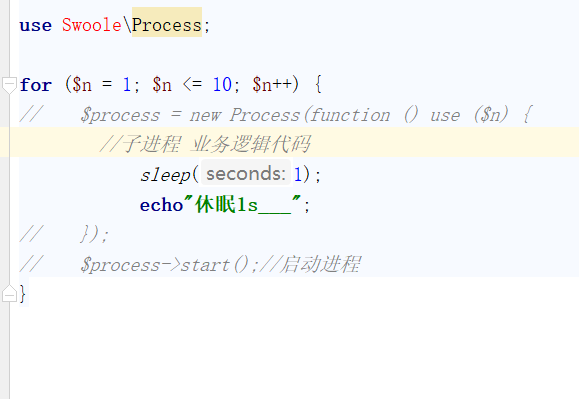
不支持重定向标准输入和输出

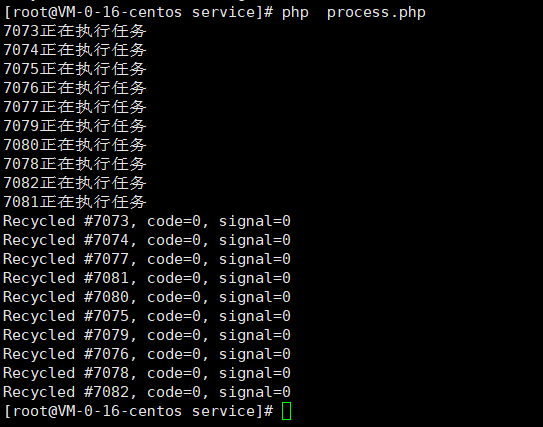
只提供fork这样的原始接口，容易使用错误

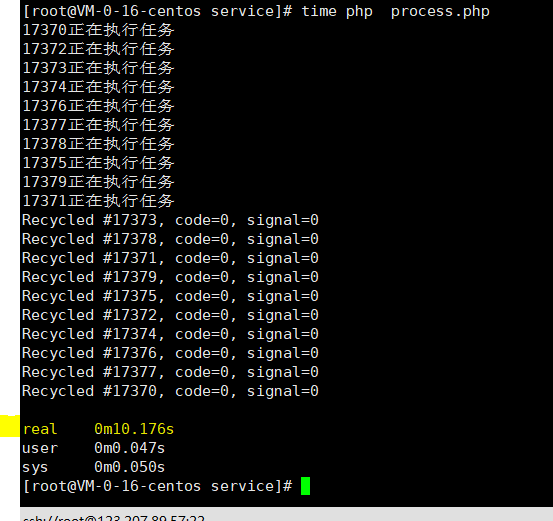
#### Swoole进程的优点

* 可以方便的实现进程间通讯
* 支持重定向标准输入和输出，在子进程内 echo 不会打印屏幕，而是写入管道，读键盘输入可以重定向为管道读取数据
* 提供了 [**exec**](https://wiki.swoole.com/#/process/process?id=exec) 接口，创建的进程可以执行其他程序，与原 PHP 父进程之间可以方便的通信
* 在协程环境中无法使用 Process 模块，可以使用 runtime hook+proc\_open 实现，参考[**协程进程管理**](https://wiki.swoole.com/#/coroutine/proc_open)

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

#### 方法

##### \_\_construct()

Swoole\Process::\_\_construct(callable $function, bool $redirect\_stdin\_stdout = false, int $pipe\_type = SOCK\_DGRAM, bool $enable\_coroutine = false);

参数：

callable $function 子进程创建成功后要执行的函数

bool $redirect\_stdin\_stdout 重定向子进程的标准输入和输出

启用此选项后，在子进程内输出内容将不是打印屏幕，而是写入到主进程管道。读取键盘输入将变为从管道中读取数据。默认为阻塞读取。参考 [**exec()**](https://wiki.swoole.com/#/process/process?id=exec) 方法内容

int $pipe\_type

启用 $redirect\_stdin\_stdout 后，此选项将忽略用户参数，强制为 1。如果子进程内没有进程间通信，可以设置为 0

bool $enable\_coroutine

在 callback function 中启用协程，开启后可以直接在子进程的函数中使用协程 API

默认值为false

##### exec()

执行一个外部函数，此函数是exec系统调用的封装

Swoole\Process->exec(string $execfile, array $args);

**string $execfile** 指定可执行文件的绝对路径，如 "/usr/bin/python"

**array $args** exec 的参数列表【如 array('test.py', 123)，相当于 python test.py 123】

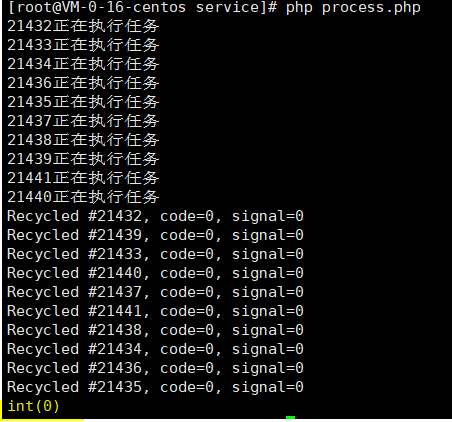
执行成功后，当前进程的代码段将会被新程序替换。子进程蜕变成另外一套程序。父进程与当前进程仍然是父子进程关系

#### Swoole进程的封装

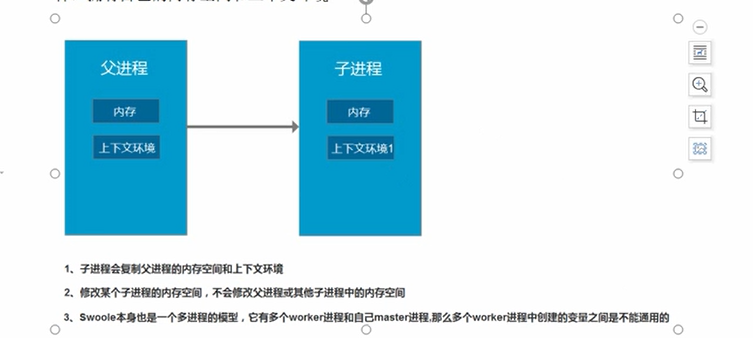
#### 进程间通讯(IPC)

*//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

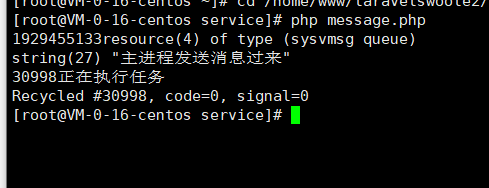
共享数据：多个进程想要操作共享数据，一个进程对数据的修改，其他进程应该立刻看到

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



#### 进程的使用场景

执行多个url，原始方案同步顺序执行，---🡪问题:执行慢

解决方案：引入swoole process 按需开启N个子进程执行

### table

由于php不支持多进程，因此swoole使用多进程模式，在多进程模式下存在进程内存隔离，在工作进程内修改 global 全局变量和超全局变量时，在其他进程是无效的。

Swoole\_table是一个基于共享内存和锁实现的超高性能并发数据结构

用于解决多进程 / 多线程数据共享和同步加锁问题。Table 的内存容量不受 PHP 的 memory\_limit 控制

#### 进程隔离的解决方案

使用外部存储

数据库：mysql,MongoDB

缓存服务器：redis,Memcache

磁盘文件：多进程并发读写时需要加锁

普通的数据库和磁盘文件操作，存在较多 IO 等待时间。因此推荐使用：

Redis 内存数据库，读写速度非常快，但是有 TCP 连接等问题，性能也不是最高的。

/dev/shm 内存文件系统，读写操作全部在内存中完成，无 IO 消耗，性能极高，但是数据不是格式化的，还有数据同步的问题

Swoole\_table解决

#### 优势

* 性能强悍，单线程每秒可读写 200 万次；
* 应用代码无需加锁，Table 内置行锁自旋锁，所有操作均是多线程 / 多进程安全。用户层完全不需要考虑数据同步问题；
* 支持多进程，Table 可以用于多进程之间共享数据；
* 使用行锁，而不是全局锁，仅当 2 个进程在同一 CPU 时间，并发读取同一条数据才会进行发生抢锁。

#### 方法

##### [column()](https://wiki.swoole.com/#/memory/table?id=column)

内存表增加一列

Swoole\Table->column(string $name, int $type, int $size = 0);

参数：

$name 指定字段的名称

$type 指定字段类型

$size 指定字符串字段的最大长度



##### Get()

print\_r($table->get(**'swe2'**));

##### Incr()增加

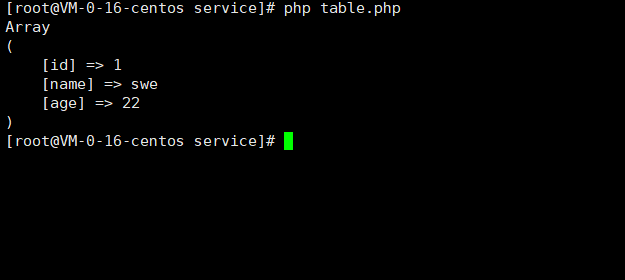
*//自增*$table->incr(**'swe2'**,**'age'**,10);

##### Decr()减少

$table->decr(**'swe2'**,**'age'**,5);

#### 实例

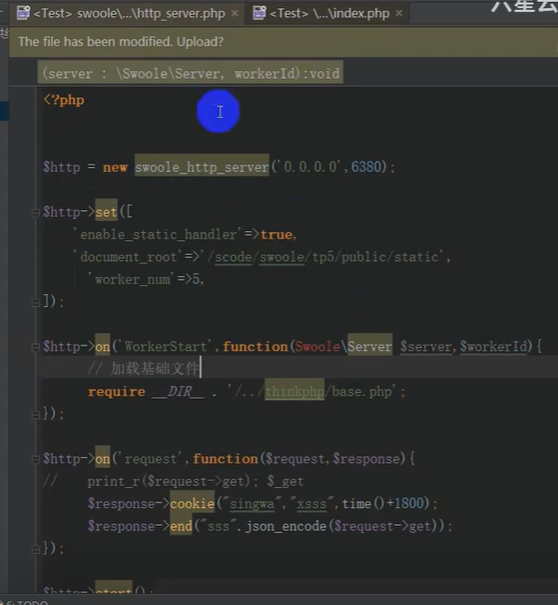
*//创建内存表*$table=**new** swoole\_table(1024);  
  
*//内存表增加一行*$table->column(**'id'**,$table::*TYPE\_INT*,4);  
$table->column(**'name'**,$table::*TYPE\_STRING*,64);  
$table->column(**'age'**,$table::*TYPE\_INT*,4);  
$table->create();  
*//*$table->set(**'swe'**,[**'id'**=>1,**'name'**=>**'swe'**,**'age'**=>22]);  
*print\_r*($table->get(**'swe'**));



### Swoole适配到tp框架

参考: https://blog.csdn.net/jeremy\_ke/article/details/85054652

[onWorkerStart](https://wiki.swoole.com/#/server/events?id=onworkerstart) 调用 index.php



## 客户端

### tcp同步阻塞客户端

#### 完整事例

*//同步阻塞客户端  
//创建client对象*$client = **new** Swoole\Client(***SWOOLE\_SOCK\_TCP***,***SWOOLE\_SOCK\_SYNC***);  
**if**(!$client->connect(**'0.0.0.0'**,9503,-1)){  
 **exit**(**'连接失败'**.$client->**errCode**);  
}  
*//向服务器发送远程服务器*$client->send(**'客户端向服务器发送数据'**);  
*//从服务器接收数据***echo "接收服务端数据:"**.$client->recv();  
*//关闭连接*$client->close();

#### 1.1创建Clinent对象

Swoole\Client::\_\_construct(int $sock\_type, int $is\_sync = SWOOLE\_SOCK\_SYNC, string $key);

参数说明：

int $sock\_type表示客户端支持的类型 有：

SWOOLE\_SOCK\_TCP、SWOOLE\_SOCK\_TCP6、SWOOLE\_SOCK\_UDP、SWOOLE\_SOCK\_UDP6

int $is\_sync = SWOOLE\_SOCK\_SYNC表示同步还是异步默认是异步的

string $key ,长连接的key 默认值是ip:port

#### 1.2connenct判断是否连接到服务器

Swoole\Client->connect(string $host, int $port, float $timeout = 0.5, int $flag = 0): bool

$host ip

$port 端口

$timeout 超时时间

$flag :

在UDP类型时表示是否启用udp\_connect, 此 UDP 将会丢弃非指定 host/port 的数据包

在TCP类型时，$flage=1表示设置为非阻塞socket,之后fd会变异步io,connect会立即返回，如果将flage设为1，name在send/recv前必须使用swoole\_client\_select来检测是否完成连接

##### 1.21失败重连

#### 1.3send()发送数据到远程服务器

Swoole\Client->send(string $data): int|false

参数：

String $data 发送的内容 支持二进制数据

返回值：返回已发送数据长度

失败返回 false并设置errCode属性

如果未执行 connect，调用 send 会触发警告

发送的数据没有长度限制

发送的数据太大 Socket 缓存区塞满，程序会阻塞等待可写

#### 1.4recv()接收服务器数据

Swoole\Client->recv(int $size = 65535, int $flags = 0): string | false

参数：

$size:接收数据的缓存区最大长度

$flags 额外参数

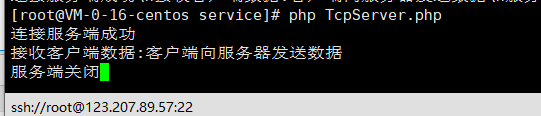
#### 1.5关闭连接

$client->close();

#### 客户端运行结果



#### 服务端接收客户端数据返回结果



#### 客户端接收服务端数据



### 异步客户端

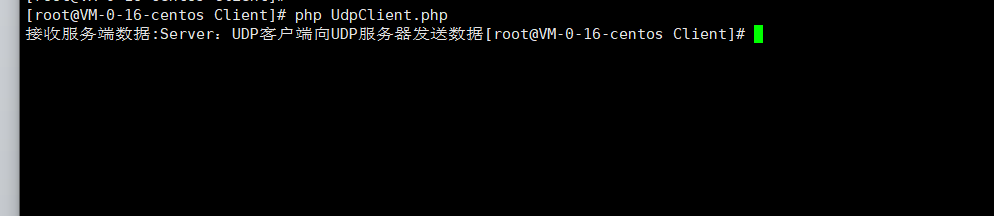
使用场景：服务端不能及时返回数据内容

### UDP客户端

#### 完整事例

*//创建client对象*$client = **new** Swoole\Client(***SWOOLE\_SOCK\_UDP***,***SWOOLE\_SOCK\_SYNC***);  
  
*//向服务器发送远程服务器*$client->sendTo(**'0.0.0.0'**,9504,**'UDP客户端向UDP服务器发送数据'**);  
*//从服务器接收数据***echo "接收服务端数据:"**.$client->recv();  
*//关闭连接*$client->close();

#### UDP客户端运行结果



#### UDP服务端接收客户端数据结果



### 4.http客户端

## Swoole的回调机制

参考：https://blog.csdn.net/u011913330/article/details/95732817

## Swoole事件

### onPacket

接收都UDP数据包时，回调此函数，发生在worker进程中

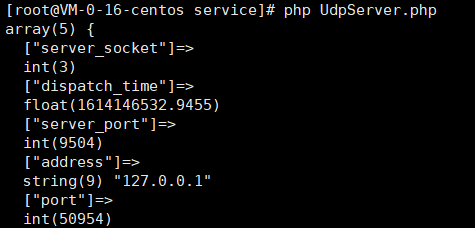
function onPacket(Swoole\Server $server, string $data, array $clientInfo);

$server 服务对象

$data 收到的数据

$clientInfo 客户端信息包括address,port,server\_port等客户端信息

$clientInfo信息：



$server->on(**'Packet'**, **function** ($server, $data, $clientInfo) {  
 *var\_dump*($clientInfo);  
 $server->sendto($clientInfo[**'address'**], $clientInfo[**'port'**], **"Server：**{$data}**"**);  
});

## Swoole方法

### sendTo()

向任意客户端发，以ip+端口为标识发送UDP数据包

Swoole\Server->sendto(string $ip, int $port, string $data, int $serverSocket = -1): bool

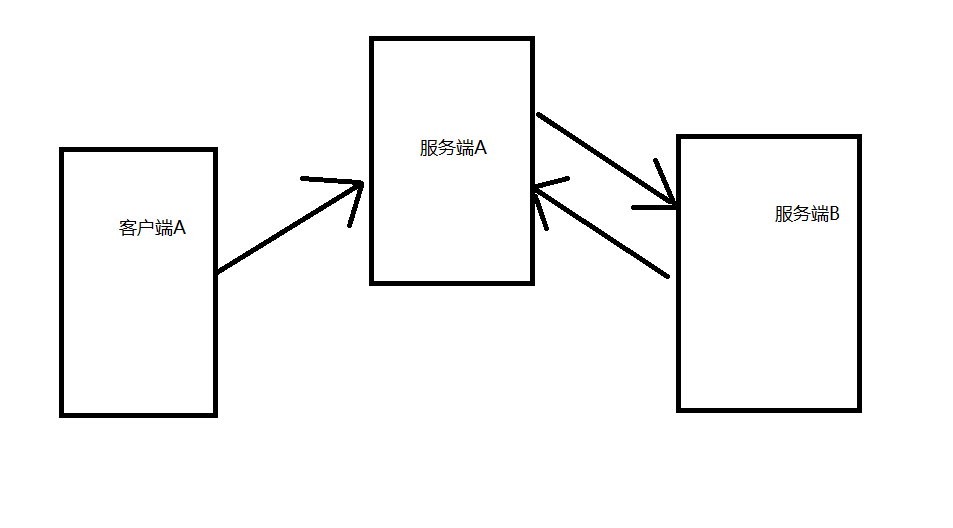
$serverSocket 指定使用哪个端口发送数据包的对应端口---当服务器同时监听多个端口，此参数可以指定哪个端口发送数据包

$port2 = $server->listen("127.0.0.1", 9502, SWOOLE\_SOCK\_UDP);监听新端口

*//基于ip+端口发送信息*$server->sendto($clientInfo[**'address'**], $clientInfo[**'port'**], **"Server：**{$data}**"**);

## tcp案例一

实现功能：



客户端A请求服务端A获取信息，服务端A请求服务端B获取信息 ，最后返回给客户端A

代码：

### 客户端A

**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2021/2/24  
 \* Time: 14:55  
 \*/*$client=**new** Swoole\Client(***SWOOLE\_SOCK\_TCP***,***SWOOLE\_SOCK\_SYNC***);  
**if**(!$client->connect(**'0.0.0.0'**,9505,-1)){  
 **exit**(**'连接失败'**.$client->**errCode**);  
}  
$client->send(**'连接服务器A'**);  
**echo '客户端A获取的数据：'**.$client->recv();  
$client->close();

### 服务端A

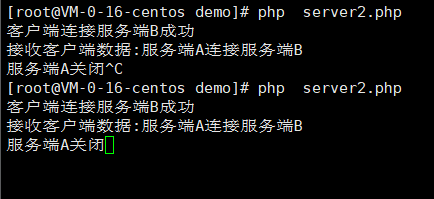
**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2021/2/24  
 \* Time: 14:55  
 \*/*$server=**new** Swoole\Server(**'0.0.0.0'**,9505);  
  
$server->on(**'Connect'**,**function** ($server,$fd){  
 **echo "客户端连接服务端A成功\n"**;  
});  
  
$server->on(**'Receive'**,**function** ($server,$fd,$form\_id,$data){  
 **echo "接收客户端数据:**{$data}**\n"**;  
 *//创建客户端* $client=**new** Swoole\Client(***SWOOLE\_SOCK\_TCP***,***SWOOLE\_SOCK\_SYNC***);  
 **if**(!$client->connect(**'0.0.0.0'**,9506,-1)){  
 **exit**(**'连接失败'**.$client->**errCode**);  
 }  
 $client->send(**'服务端A连接服务端B'**);  
 $message=$client->recv();  
 $client->close();  
 $server->send($fd,**"服务端A返回的数据\_A+"**.$message);  
});  
  
$server->on(**'Close'**,**function** ($server,$fd){  
 **echo '服务端A关闭'**;  
});  
  
$server->start();

### 服务端B

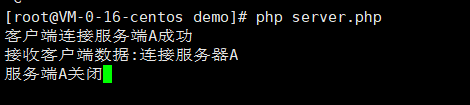
**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2021/2/24  
 \* Time: 14:56  
 \*/*$server=**new** Swoole\Server(**'0.0.0.0'**,9506);  
  
$server->on(**'Connect'**,**function** ($server,$fd){  
 **echo "客户端连接服务端B成功\n"**;  
});  
  
$server->on(**'Receive'**,**function** ($server,$fd,$form\_id,$data){  
 **echo "接收客户端数据:**{$data}**\n"**;  
 $server->send($fd,**"服务端B返回的数据\_B"**);  
  
});  
  
$server->on(**'Close'**,**function** ($server,$fd){  
 **echo '服务端A关闭'**;  
});  
  
$server->start();

### 运行结果

服务端B



服务端A

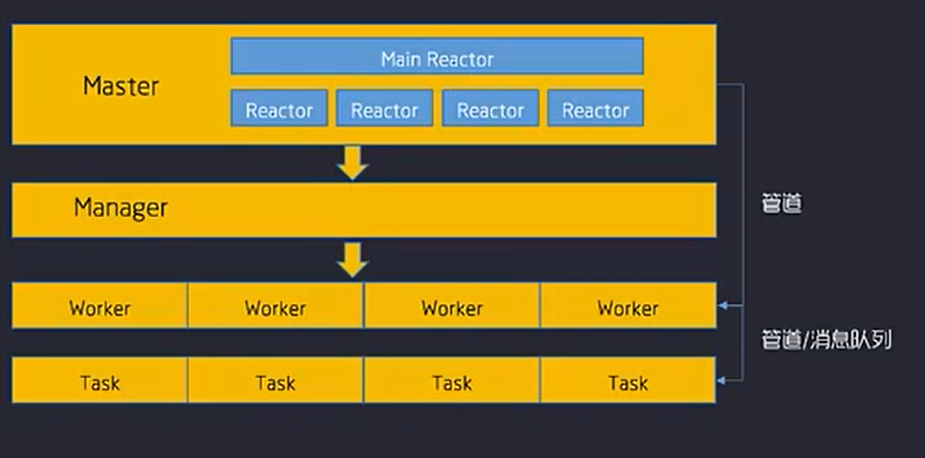


客户端A



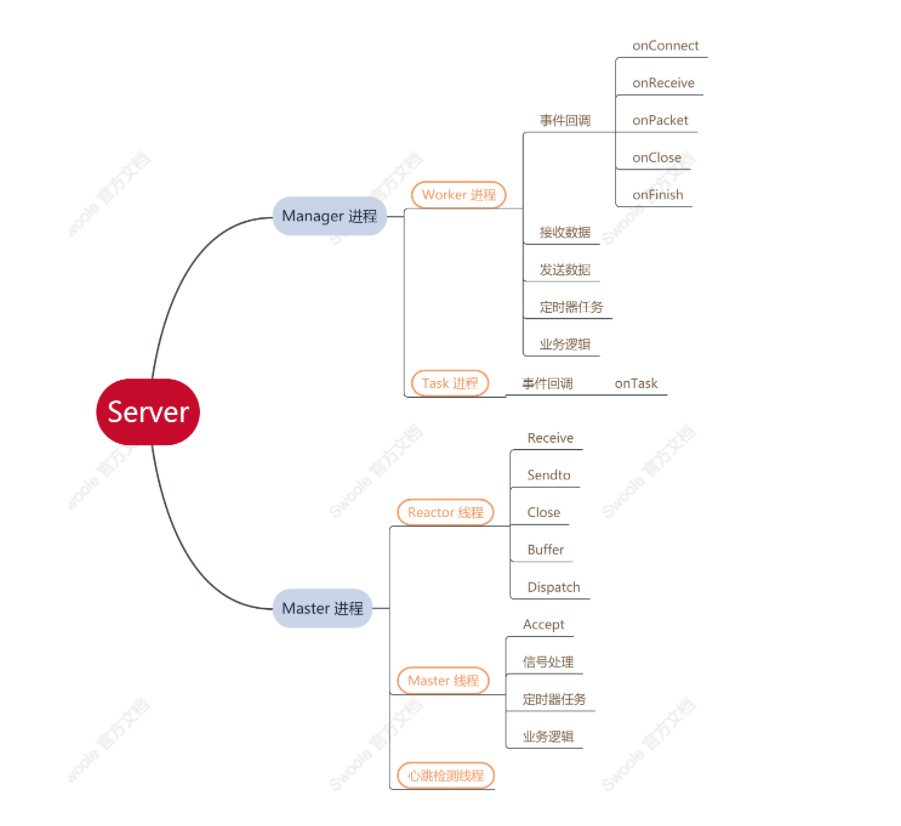
## Websocket

## Swoole结构

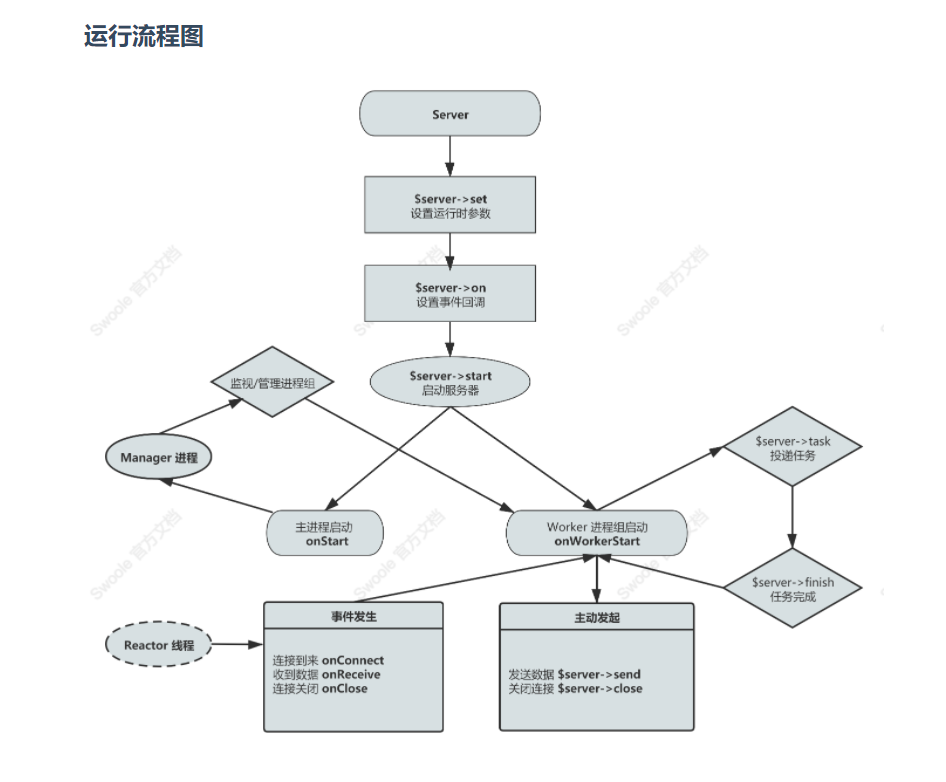


Master进程是swoole的主进程，处理swoole核心的事件驱动。

### 进程流程图



### 运行流程图



## IO阻塞

### 同步



像上面的加载文件的代码一样，php的代码也是一步一步执行，会带来Io阻塞的问题

### 异步



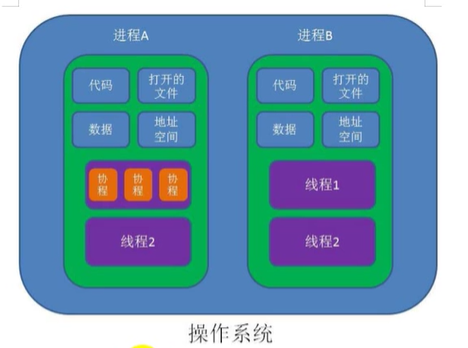
以前端的异步举例，如ajax,ajax在执行一个请求的时候不会等待请求的返回结果 而是直接打印出lol,这样Io就不会阻塞，但会带来新的问题，打印完lol不知道代码执行到哪只能等待回调，执行是嵌套的，一旦业务逻辑多就会造成层层嵌套，可读性差，前端用promise解决

而swoole解决异步嵌套使用协程

并发 1万次请求从mysql读取海量数据仅需0.2秒

**use** Swoole\Coroutine;  
*//协程并发测试  
//并发 1万次请求从mysql读取海量数据仅需0.2秒*$s=*microtime*(**true**);*//返回当前unix时间戳的微妙数***for**($i=100;$i--;){  
 *//创建协程* Coroutine::*create*(**function** (){  
 *//连接mysql* $mysql=**new** Swoole\Coroutine\MySQL();  
 $mysql->connect([  
 **'host'**=>**'175.27.223.19'**,  
 **'user'**=>**'SWE16081'**,  
 **'password'**=>**'CPabemcTHsc6wwrb'**,  
 **'database'**=>**'swe16081'**,  
 ]);  
 $statement = $mysql->prepare(**'SELECT `id` FROM `users`'**);  
 **for** ($n = 100; $n--;) {  
  
 $result=$statement->execute();  
 *var\_dump*($result);  
 *assert*(*count*($result) > 0);  
 }  
 });

## swoole协程



进程：具有一定独立功能的程序关于某个数据集合的一次行动，占有内存最大所以切换进程开销比较大，但是稳定安全，进程间内存隔离。

线程：是进程一个实体是cpu调度和分配的基本单位，线程通信通过数据共享，上下文切换快，资源开销少，但容易丢失数据（加锁—）造成较大开销）

### 1.什么是协程

寄宿在线程下一种更轻量的并发协作的轻量线程。

一群大佬觉得线程不好用，就自己写了一个轻量级线程，一个线程可以有多个协程。

Swoole的协程的底层是单线程实现的，因此同一时间只有一个协程工作，协程的执行是串行的，这与线程不同，多个线程会被操作系统调度到多个CPU并行执行的

Swoole4拥有高可用性的内置协程，可以使用完全同步的代码来实现异步性能，php代码没有任何额外的关键字，底层会自动进行协程调度。

可以将协程理解为超轻量级的线程，可以在一个进程中创建成千上万个协程

并发 1万次请求从mysql读取海量数据仅需0.2秒

**use** Swoole\Coroutine;  
*//协程并发测试  
//并发 1万次请求从mysql读取海量数据仅需0.2秒*$s=*microtime*(**true**);*//返回当前unix时间戳的微妙数***for**($i=100;$i--;){  
 *//创建协程* Coroutine::*create*(**function** (){  
 *//连接mysql* $mysql=**new** Swoole\Coroutine\MySQL();  
 $mysql->connect([  
 **'host'**=>**'175.27.223.19'**,  
 **'user'**=>**'SWE16081'**,  
 **'password'**=>**'CPabemcTHsc6wwrb'**,  
 **'database'**=>**'swe16081'**,  
 ]);  
 $statement = $mysql->prepare(**'SELECT `id` FROM `users`'**);  
 **for** ($n = 100; $n--;) {  
  
 $result=$statement->execute();  
 *var\_dump*($result);  
 *assert*(*count*($result) > 0);  
 }  
 });

### 2.协程与线程的区别

#### 调度方式

线程的调度方式为系统调度，调度策略有：分时调度、抢占调度。

协程的调度方式为协作是调度方式，不受内核控制，由自由策略调度切换。

### 3.协程为什么快

减少IO阻塞带来的性能损耗

### 4.协程优缺点

### 5.协程容器

所有的协程必须在协程容器里创建，swoole程序启动时大部分情况会自动创建协程容器，

Swoole启动程序的方式有三种

1. 调用异步服务端程序的start方法，此方法会在事件回调中创建协程容器
2. 调用swoole提供两个进程管理模块Process和Process\Pool的start方法，此方式会在进程启动的时候创建协程容器
3. 裸写协程的启动方式，需要先创建一个协程容器Coroutine\run()

**use** Swoole\Coroutine;  
**use function** Swoole\Coroutine\run;  
*//协程容器*run(**function** (){  
 *//在协程容器中创建 协程* Coroutine::*create*(**function** (){  
 **echo "创建一个协程："**.Coroutine::*getCid*();  
 });  
});



 Coroutine\run() 函数其实是对 Swoole\Coroutine\Scheduler 类 (协程调度器类) 的封装

使用Scheduler 创建携程 set()，getOption()有点问题

### 6.swoole协程的使用

协程的使用方法参考Swoole4文档

#### 6.1创建协程的方法

所有的协程必须在协程容器里创建，swoole程序启动时大部分情况会自动创建协程容器，

Swoole启动程序的方式有三种

1.调用异步服务端程序的start方法，此方法会在事件回调中创建协程容器

2.调用swoole提供两个进程管理模块Process和Process\Pool的start方法，此方式会在进程启动的时候创建协程容器

3.裸写协程的启动方式，需要先创建一个协程容器Coroutine\run()

##### go()创建协程

go()没有创建协程容器为何可以创建协程？

go(**function** (){  
 **echo "创建一个协程："**.Swoole\Coroutine::*getCid*();  
});



##### create协程的创建

**use** Swoole\Coroutine;  
**use function** Swoole\Coroutine\run;  
*//协程容器*run(**function** (){  
 *//在协程容器中创建 协程* Coroutine::*create*(**function** (){  
 **echo "创建一个协程："**.Coroutine::*getCid*();  
 });  
});



### [Coroutine\Scheduler](https://wiki.swoole.com/#/coroutine/scheduler?id=coroutinescheduler)

协程调度器

作用？暂时不理解，下面的set，getOption()与其有关

#### 6.2set()设置协程运行参数

Swoole\Coroutine\Scheduler->set(array $options): bool

Swoole\Coroutine::set(array $options);

参数



返回结果为bool

##### 使用实例

$options=[  
 **'max\_coroutine'**=>100*//全局最大协程数，超过则无法创建协程*];  
$flage=Coroutine::*set*($options);  
**if**(!$flage){  
 **exit**(**'设置协程参数出错'**);  
}

线上的设置参数一直返回false

#### 6.2 getOptions()获取协程设置的选项

Swoole>=4.6可用

Swoole\Coroutine::getOptions(): null|array;

### 7.协程的使用场景

高并发服务：秒杀系统、高性能API接口、RPC服务器(这是什么)

爬虫，可实现非常巨大的并发能力，即使非常慢速的网络环境，也可以高效利用带宽

即时通信服务：IM聊天、游戏服务器、物联网、消息服务器等。确保消息通讯完全无阻塞，每个消息包均可即时的被处理。

#### 1.高性能API

Swoole编写http server

#### 2.秒杀场景

对于秒杀场景，并发量很大，而协程可以极大地的提高并发，但并发高了并不一定是一个好的事情。



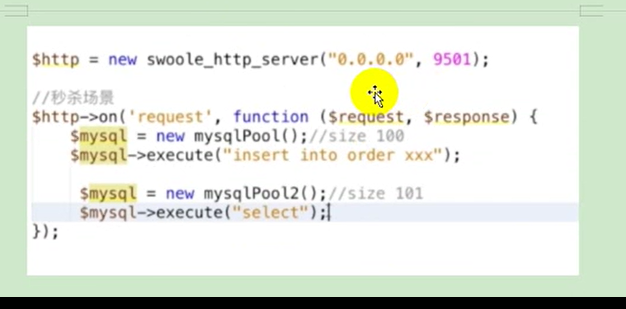
New一个http服务端，连接数据库执行插入请求

并发量过万，1万个用户同时执行添加操作，连接池大小100,可以同时处理100个请求,

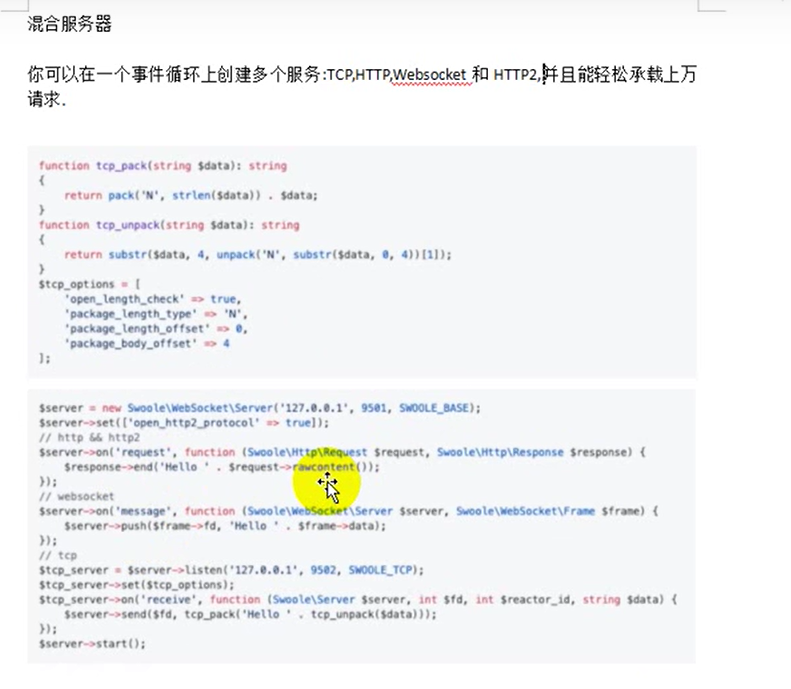
如果这时候有1000个请求进来，因为协程可以承受很大的并发量，所以每个协程进来都会接受这个请求，但是mysql处理不了，这时候就会阻塞900个请求，等待100个请求执行完，这时和php的fpm效果一样。

协程可以承受大的并发量单处理不来。所以在处理高并发的时候尽量把流量往外档，缓存的机制

协程理想的状态是能处理多少并发就来多少请求。



## 混合服务器



## channel通道

进程间有内存隔离，交换数据通过本地通信的手段。

通道是协程之间通信交换数据的唯一渠道，在swoole中通道常用于连接池的实现和协程并发的调度。

通道，用于协程间通讯，支持多生产者协程和多消费者协程。底层自动实现了协程的切换和调度

### 实现原理

通道于php的array类似，仅占内存，没有申请其他额外的资源，所有的操作都是内存操作，无IO消耗。

底层使用php引用计数实现，无内存拷贝，即使是传递巨大的字符串或数组也不会产生额外的性能消耗。

Channel基于引用计数实现，是零拷贝的。

### 连接池实例



## 进程

Worker进程

## Swoole框架内核

## 分布式框架RPC通讯机制

### RPC远程调用

#### 1.什么是RPC

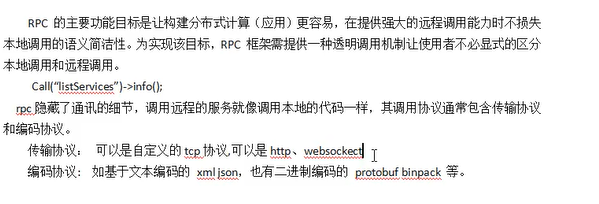
RPC是一种通过网络从远程计算机上请求服务而不需要了解底层网络

技术的协议。

比如：有两台服务器。一个应用放在A服务器上，想要调用B服务器上应用的方法或函数，由于不在一个内存空间，不能直接调用，就需要通过网络来表带调用的寓意和传达调用的数据，这就是RPC

#### 2.为什么需要RPC

RPC的主要功能目标是让构建分布式计算(应用)更容易，在提供强大的远程调用能力时不损失本地调用的语义简洁性，为实现该目标，RPC框架需提供一种透明机制让使用者不必显示的区分本地调用和远程调用



#### 3.RPC服务端封装

#### 4.RPC客户端封装