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++ 语言编写的基于异步事件驱动和协程的并行网络通信引擎

Swoole面试题

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

网络调试工具下载

## 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。

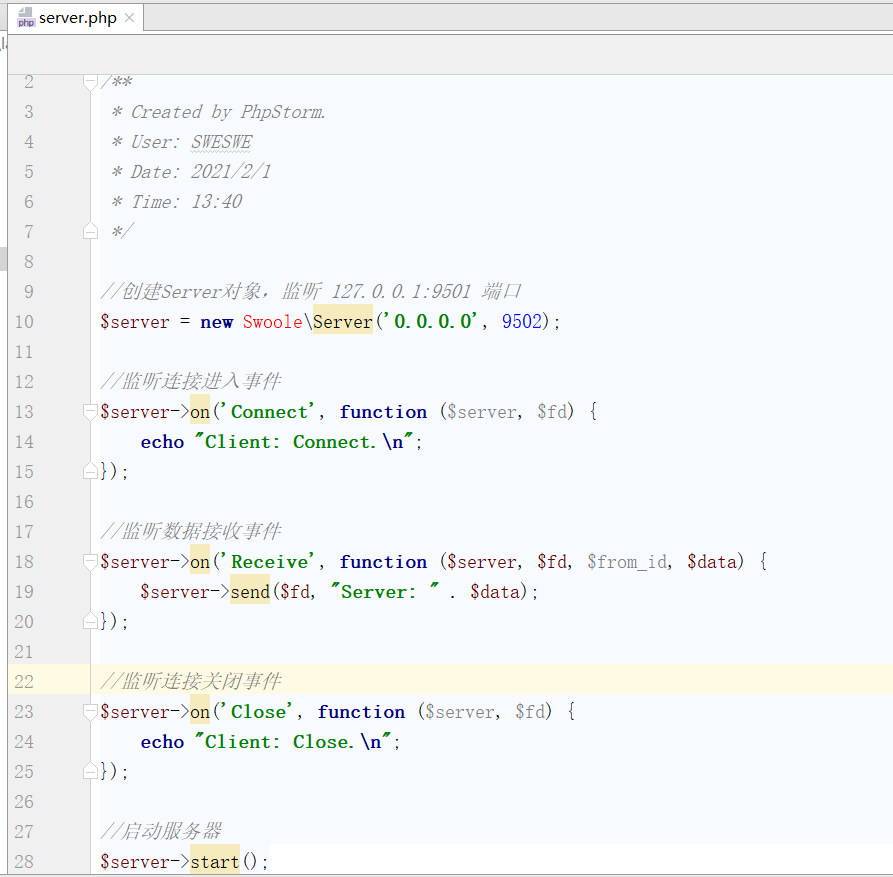
1 yum install php72w-devel

**安装 swoole**

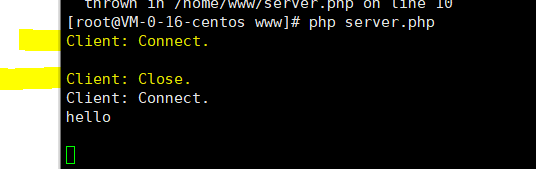
1 pecl install swoole

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

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

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();

## 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

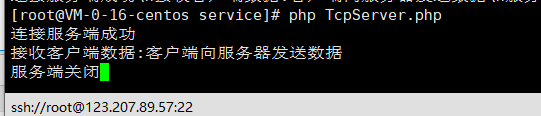
#### 1.5关闭连接

$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

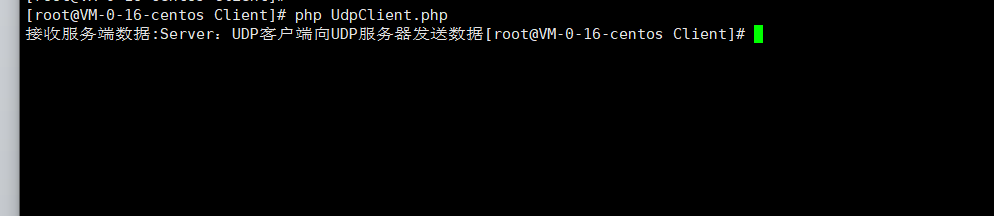
使用场景：服务端不能及时返回数据内容

### 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



## Swoole事件

### onPacket

接收都UDP数据包时，回调此函数，发生在worker进程中

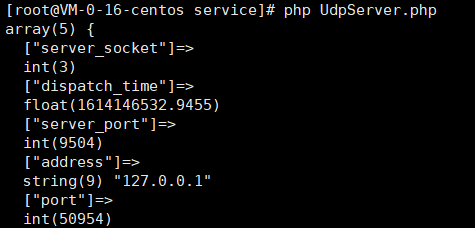
function onPacket(Swoole\Server $server, string $data, array $clientInfo);

$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

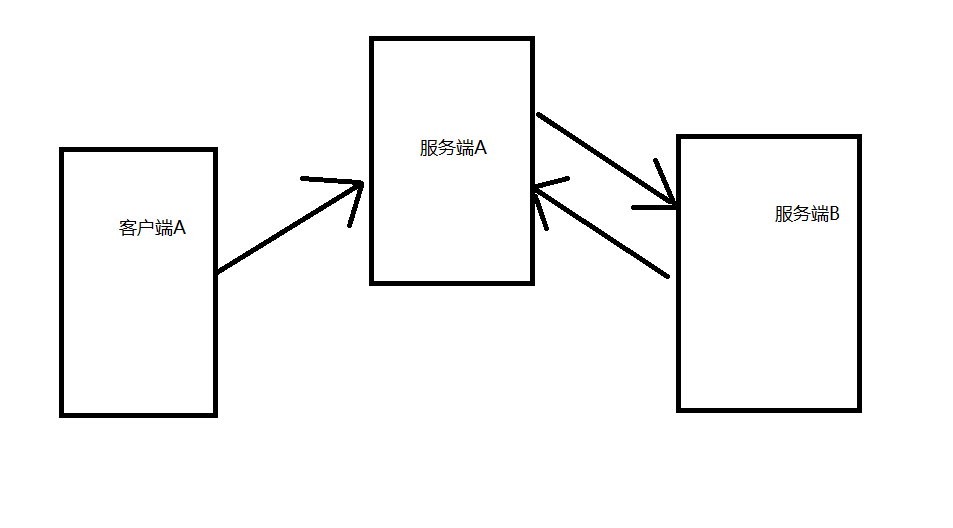
$serverSocket 指定使用哪个端口发送数据包的对应端口---当服务器同时监听多个端口，此参数可以指定哪个端口发送数据包

$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

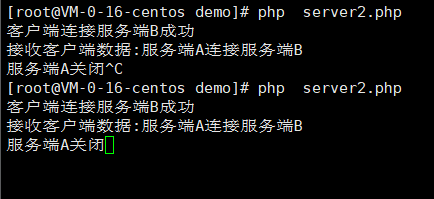
**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2021/2/24  
 \* Time: 14:55  
 \*/*$server=**new** Swoole\Server(**'0.0.0.0'**,9505);  
  
$server->on(**'Connect'**,**function** ($server,$fd){  
 **echo "客户端连接服务端A成功\n"**;  
});  
  
$server->on(**'Receive'**,**function** ($server,$fd,$form\_id,$data){  
 **echo "接收客户端数据:**{$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

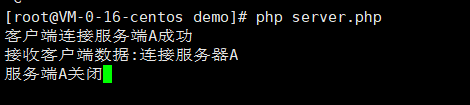
**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2021/2/24  
 \* Time: 14:56  
 \*/*$server=**new** Swoole\Server(**'0.0.0.0'**,9506);  
  
$server->on(**'Connect'**,**function** ($server,$fd){  
 **echo "客户端连接服务端B成功\n"**;  
});  
  
$server->on(**'Receive'**,**function** ($server,$fd,$form\_id,$data){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

### 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

### 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*();  
 });  
});



### 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*();  
 });  
});



6.2