# 内容



# PHP

## 安装

下载<http://php.net/get/php-7.3.3.tar.bz2/from/a/mirror>

解压tar -xjvf php-7.3.3.tar.bz2

安装./configure --prefix=/home/work/study/soft/php

安装libxml2

sudo apt-get install libxml2

sudo apt-get install libxml2-dev

ubuntu/debian:

apt-get install libxml2-dev

centos/redhat:

yum install libxml2-devel

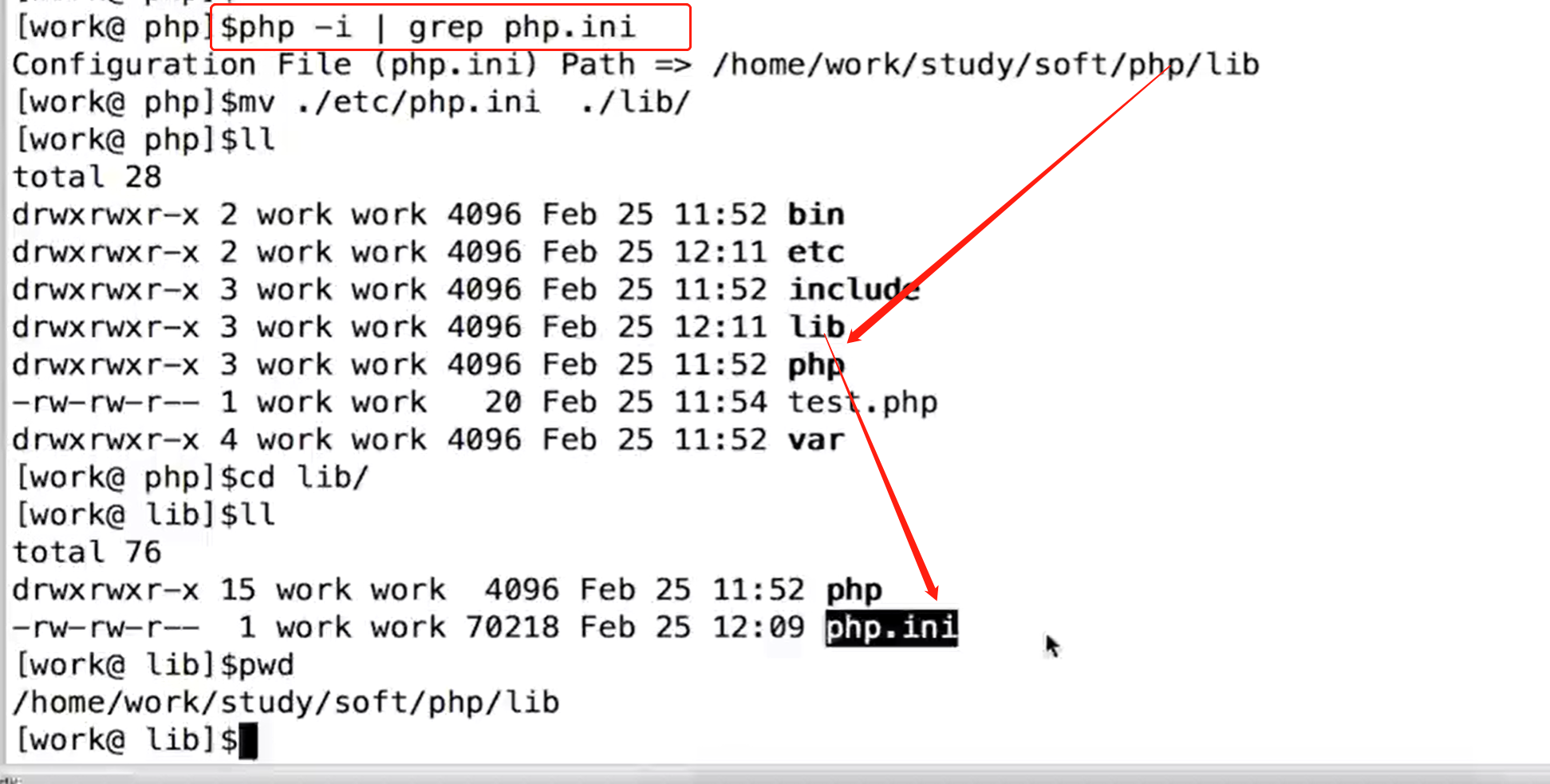
编译make Make testMake install

扩展php -m

## 添加环境变量：配置与生效



## 问题



## Swoole安装

### 源码

git clone https://gitee.com/swoole/swoole.git



### Phpize

phpize是外挂php扩展的

./configure --with-php-config=/home/work/study/soft/php/bin/php-config

Make

Make test

Make install

# 架构



|  |
| --- |
| 请求到达 Main Reactor  |  Main Reactor根据Reactor的情况，将请求注册给对应的Reactor  (每个Reactor都有epoll。用来监听客户端的变化)  |  客户端有变化时，交给worker来处理  |  worker处理完毕，通过进程间通信(比如管道、共享内存、消息队列)发给  对应的reactor。  |  reactor将响应结果发给相应的连接  |  请求处理完成  因为reactor基于epoll，所以每个reactor可以处理无数个连接请求。 如此，swoole就轻松的处理了高并发。 |

# 端口操作

## 端口操作

**开放端口**

**[*root*@192 /]# systemctl start firewalld  
[*root*@192 /]# firewall-cmd --permanent --add-port=8811/tcp**

**查看端口的监听情况**

**[root@192 jim]# netstat -anp | grep 9501**

## 访问操作

**启动swoole的代码 php http.php**

**服务器端的访问，使用：[root@192 ~]# curl <http://127.0.0.1:8811>**

**浏览器端的访问，使用：<http://l2.l2.l2:8811/live/index.html>**

**记得哦 ，打开phphstudy 因为有一个虚拟域名的转发**

# TCP

查看端口的监听情况

[root@192 jim]# netstat -anp | grep 9501

连接服务端的端口

[root@192 jim]# telnet 127.0.0.1 9501

运行程序

[root@192 l2.l2.l2]# php /home/wwwroot/l2.l2.l2/tcp.php

[root@192 example]# ps aft | grep tcp.php

64402 pts/4 S+ 0:00 \\_ grep --color=auto tcp.php

64118 pts/3 Sl+ 0:00 \\_ php tcp.php

64119 pts/3 S+ 0:00 \\_ php tcp.php

64121 pts/3 S+ 0:00 \\_ php tcp.php

# UDP

<?php  
//创建Server对象，监听 127.0.0.1:9502端口，类型为SWOOLE\_SOCK\_UDP  
$serv = new swoole\_server("127.0.0.1", 9502, *SWOOLE\_PROCESS*, *SWOOLE\_SOCK\_UDP*);  
//监听数据接收事件  
$serv->on('Packet', function (*$serv*, *$data*, *$clientInfo*) {  
 *$serv*->sendto(*$clientInfo*['address'], *$clientInfo*['port'], "Server ".*$data*);  
 *var\_dump*(*$clientInfo*);  
});  
//启动服务器  
$serv->start();  
// 测试方法  
// php udp.php  
// UDP服务器可以使用netcat -u 来连接测试  
//  
// nc -u 127.0.0.1 9502  
// hello  
// Server: hello

# HTTP

## 继承



## 防火墙端口操作

开放端口

[*root*@192 /]# systemctl start firewalld  
[*root*@192 /]# firewall-cmd --permanent --add-port=8811/tcp

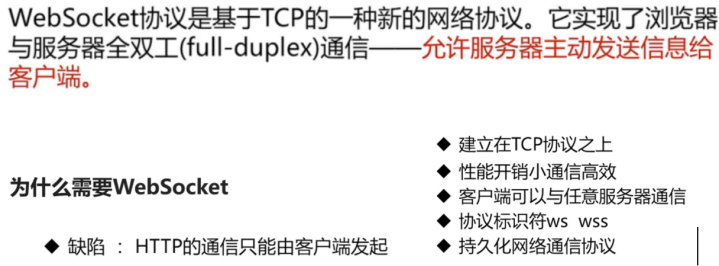
<?php  
$http = new swoole\_http\_server("0.0.0.0", 8811);  
  
$http->set([  
 'enable\_static\_handler' => true,  
 //设置根目录这里根据你自己的路径来写 后面的代码就不会执行了 直接去找静态文件了  
 'document\_root' => "/home/wwwroot/l2.l2.l2/example",  
]);  
  
$http->on('request', function (*$request*, *$response*) {  
 echo 'nihao';  
 //输出值命令行的  
 *print\_r*(*$request*->get);  
 //输出值浏览器的  
 *$response*->end('hello swoole');  
 // $response->cookie('cookie', 'value', time() + 3600);  
  
});  
  
$http->start();

# Websocket

## 概述

用来代替轮询的，轮询是浏览器间隔发起，比较浪费资源

http方法websocket也可以使用 socket继承自http



## 回调

onMessage回调函数为必选 -当服务器收到来自客户端的数据帧时会回调此函数。

onOpen-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。

onHandShake回调函数为可选--WebSocket建立连接后进行握手。WebSocket服务器已经内置了handshake，如果用户希望自己进行握手处理，可以设置onHandShake事件回调函数。

## 代码

<?php  
  
$server = new Swoole\WebSocket\Server("0.0.0.0", 8811);  
  
$server->on('open', function (Swoole\WebSocket\Server *$server*, *$request*) {  
 echo "server: handshake success with fd{*$request*->fd}\n";  
});  
  
$server->on('message', function (Swoole\WebSocket\Server *$server*, *$frame*) {  
 echo "receive from {*$frame*->fd}:{*$frame*->data},opcode:{*$frame*->opcode},fin:{*$frame*->finish}\n";  
 *$server*->push(*$frame*->fd, "this is server");  
});  
  
$server->on('close', function (*$ser*, *$fd*) {  
 echo "client {*$fd*} closed\n";  
});  
$server->start();

<!DOCTYPE html>  
<html>  
<head>  
</head>  
<body>  
<div>**我是web socket**</div>  
<script>  
 *console*.log("test");  
 var *wsURL* = 'ws://ct.ct.ct:8811';  
 var *websocket* = new WebSocket(*wsURL*);  
 //实例对象的onopen属性  
 *websocket*.onopen = function (*evt*) {  
 *websocket*.send("send message");  
 *console*.log("connected");  
 }  
 //实例化onmessage  
 *websocket*.onmessage = function (*evt*) {  
 *console*.log("server return：" + *evt*.data);  
 }  
 *websocket*.onclose = function (*evt*) {  
 *console*.log('server close：');  
 }  
 *websocket*.onerror = function (*evt*) {  
 *console*.log('server error：' + *evt*.data);  
 }  
</script>  
</body>  
</html>

# Task

***task是异步执行的一个进程 tash()会调用onTask onTask之后会执行onFinish***

$this->ws->on('message', function (Swoole\WebSocket\Server *$server*, *$frame*) {  
 echo "receive from {*$frame*->fd}:{*$frame*->data},opcode:{*$frame*->opcode},fin:{*$frame*->finish}\n";  
 $data = array(  
 'task' => 1,  
 'fd' => *$frame*->fd,  
 );  
 //在此处定义task函数，调用task函数，task函数会调用onTask函数  
 *$server*->task($data);  
 *$server*->push(*$frame*->fd, "this is server");  
});  
$this->ws->on('task', [$this, 'onTask']);

public function onTask(*$serv*, *$taskId*, *$workId*, *$data*)  
{  
 *print\_r*(*$data*);  
 *sleep*(10);  
 *print\_r*("10s");  
 return "on task finish";  
}  
  
public function onFinish(*$serv*, *$taskId*, *$data*)  
{  
 *print\_r*("end");  
 echo "taskId:".*$taskId*.*PHP\_EOL*;  
 *print\_r*(*$data*);  
}

**前端代码**

<script>  
 *console*.log("test");  
 var *wsURL* = 'ws://ct.ct.ct:8811';  
 var *websocket* = new WebSocket(*wsURL*);  
 //实例对象的onopen属性  
 *websocket*.onopen = function (*evt*) {  
 *websocket*.send("send message");  
 *console*.log("connected");  
 }  
 //实例化onmessage  
 *websocket*.onmessage = function (*evt*) {  
 *console*.log("server return：" + *evt*.data);  
 }  
 *websocket*.onclose = function (*evt*) {  
 *console*.log('server close：');  
 }  
 *websocket*.onerror = function (*evt*) {  
 *console*.log('server error：' + *evt*.data);  
 }  
</script>

# 定时器

***间隔器***

Swoole\Timer::tick(2000, function (*$timer\_id*) {  
 echo "2s--timeId--{*$timer\_id*}";  
});

***闹钟器***  
Swoole\Timer::after(5000,function() use($server, $frame){  
 echo "5s after";  
 $server->push($frame->fd,"time after");  
});

定时器是异步的，与task是一样的

# 进程

## 代码

***//主进程是process.php 子进程是$pid 使用exec创建了一个主进程***

<?php  
//主进程是process.php 子进程是$pid  
$process = new swoole\_process(function (swoole\_process *$pro*) {  
 *$pro*->exec('/usr/bin/php',['/home/wwwroot/l2.l2.l2/example/http.php']);  
}, false);  
  
$pid = $process->start();  
echo $pid . *PHP\_EOL*;  
swoole\_process::wait();

## 查看端口与进程

ps aux|grep process.php看端口

pstree -p 3056 看端口的结构树

ps aft | grep http.php 看端口的继承关系

[*root*@192 *jim*]# ps aux|grep process.php  
*root* 3056 0.0 1.5 269936 15864 *pts*/0 *S*+ 01:50 0:00 /*usr*/*bin*/*php process*.*php  
root* 3099 0.0 0.0 112716 976 *pts*/1 *R*+ 01:53 0:00 *grep* --color=*auto process*.*php*[*root*@192 *jim*]# pstree -p 3056  
php(3056)───php(3057)─┬─php(3058)───php(3060)  
 *└─*{*php*}(3059)  
[*root*@192 *jim*]# pstree -p 3099  
[*root*@192 *jim*]# pstree -p 3056  
php(3056)───php(3057)─┬─php(3058)───php(3060)  
 *└─*{*php*}(3059)  
[*root*@192 *jim*]# ps aft | grep http.php  
 3104 *pts*/1 *S*+ 0:00 \*\_ grep* --color=*auto http*.*php* 3057 *pts*/0 *Sl*+ 0:00 \*\_* /*usr*/*bin*/*php* /*home*/*wwwroot*/*l2*.*l2*.*l2*/*example*/*http*.*php* 3058 *pts*/0 *S*+ 0:00 \*\_* /*usr*/*bin*/*php* /*home*/*wwwroot*/*l2*.*l2*.*l2*/*example*/*http*.*php* 3060 *pts*/0 *S*+ 0:00 \*\_* /*usr*/*bin*/*php* /*home*/*wwwroot*/*l2*.*l2*.*l2*/*example*/*http*.*php*

## CURL的例子

<?php  
echo '开始' . *date*('y-m-d h:i:s');  
$urls = array(  
 'http://www.baidu.com',  
 'http://www.qq.com',  
 'https://github.com',  
 'https://www.csdn.net/'  
);  
//如果顺序执行的话，真的很浪费时间内！  
// foreach ($urls as $k => $v) {  
// $content[]=file\_get\_contents($v);  
// }  
// var\_dump($content);  
for ($i = 0; $i < 4; $i++) {  
 $process = new swoole\_process(function (swoole\_process *$worker*) use ($i, $urls) {  
 $content = curlData($urls[$i]);  
 echo $content . *PHP\_EOL*;  
 }, true);  
 $pid = $process->start();  
 $workers[$pid] = $process;  
}  
foreach ($workers as $k => $v) {  
 echo $v->read();  
}  
//模拟curl的请求  
function curlData(*$url*)  
{  
 *sleep*(1);  
 return *$url* . 'success' . *PHP\_EOL*;  
}  
echo '结束' . *date*('y-m-d h:i:s');

# Table

## 简介



## 代码

<?php  
$table = new swoole\_table(1024);  
//内存表增驾一行  
$table->column('id', $table::*TYPE\_INT*,8);  
$table->column('name', $table::*TYPE\_STRING*,8);  
$table->column('age', $table::*TYPE\_INT*,8);  
//建表  
$table->create();  
$table->set('key1', ['id' => 1, 'name' => 'sun', 'age' => 30]);  
$data = $table->get('key1');  
*print\_r*($data);  
$data=$table->incr('key1','age',2);  
*print\_r*($data);  
$data=$table->decr('key1','age',3);  
*print\_r*($data);  
$table->del('key1');  
*print\_r*($table['key1']);

# 协程

## 协程

<?php  
go(function () {  
 echo "goroutine 1".*PHP\_EOL*;  
});  
Swoole\Coroutine::create(function () {  
 echo "goroutine 2".*PHP\_EOL*;  
});

两种方式进行写协程

## Sleep

go(function () {  
 *sleep*(1);  
 echo "goroutine 1".*PHP\_EOL*;  
});

## 协程客户端

**协程客户端必须放在协程里面才可以执行：用来代替curl的**

<?php  
go(function (){  
 $client = new Swoole\Coroutine\Client(*SWOOLE\_SOCK\_TCP*);  
 if (!$client->connect('127.0.0.1', 8811)) {  
 echo "fail " . *PHP\_EOL*;  
 } else {  
 $client->send("test client ");  
 echo $client->recv();  
 $client->close();  
 }  
});

**也可以是个http\_server的request进行监控代替curl**

<?php  
$http = new swoole\_http\_server('0.0.0.0', 8811);  
$http->on('request', function (*$request*, *$response*) {  
 *$response*->end("st request");  
 $client = new Swoole\Coroutine\Http\Client('127.0.0.1', '8811');  
 $client->set([  
 'timeout' => 1  
 ]);  
 $client->get('/home/name');  
 //请求返回的数据  
 *print\_r*($client->body);  
 *print\_r*($client->statusCode);  
 $client->close();  
});  
$http->start();

## WS的协程

注解看

<?php  
go(function(){  
 $server = new Co\Http\Server('0.0.0.0', 8811);  
 $server->handle('/',function(*$request*,*$ws*){  
 *$ws*->upgrade();  
 while(true){  
 $data=*$ws*->recv();  
 *print\_r*($data);  
 *$ws*->push("ws client");  
 }  
 });  
 $server->start();  
});

$ws->upgrade()：向客户端发送WebSocket握手消息

while(true)循环处理消息的接收和发送

$ws->recv()接收WebSocket消息帧

$ws->push()向对端发送数据帧

$ws->close()关闭连接

$ws是一个Swoole\Http\Response对象

## Redis的协程

### 方式一：请求的方式

<?php  
  
$http = new swoole\_http\_server('0.0.0.0', 8811);  
$http->on('request', function (*$request*, *$response*) {  
 $redis = new Swoole\Coroutine\Redis();  
 $redis->connect('127.0.0.1', 6379);  
 $value = $redis->get(*$request*->get['a']);  
 *$response*->header("Content-Type", "text/plain");  
 *$response*->end($value);  
});  
  
$http->start();

### 方式二：协程go

<?php  
go(function(){  
 $redis=new Swoole\Coroutine\Redis();  
 $redis->setOptions(  
 [  
 'connect\_timeout'=>1,  
 'timeout'=>3,  
 ]  
 );  
 $redis->connect('127.0.0.1',6379);  
 $value=$redis->get("a");  
 *var\_dump*($value);  
 $value=$redis->set("b",'c');  
 *var\_dump*($value);  
});

### 方式三：runtime

<?php  
Swoole\Runtime::enableCoroutine(true);  
  
go(function(){  
 $redis = new redis();  
 $redis->connect("127.0.0.1", 6379);  
  
 $a = $redis->get('a');  
 *var\_dump*($a);  
});

## 协程执行顺序

<?php  
Swoole\Runtime::enableCoroutine(true);  
go(function () {  
 echo "1-st";  
 //redis是堵塞IO 这存在与一个协程里面 因为堵塞会交出调度权，给另外的协程进行执行  
 $redis = new redis();  
 $redis->connect("127.0.0.1", 6379);  
 $a = $redis->get('a');  
 *print\_r*($a);  
 echo "1-ed";  
});  
echo "2-content";  
go(function () {  
 echo "3-go";  
});

## 协程文件操作

<?php  
Swoole\Runtime::enableCoroutine(true);  
go(function () {  
 $fp = *fopen*("test.log", "a+");  
 *fwrite*($fp, *str\_repeat*('A', 10));  
 *fwrite*($fp, *str\_repeat*('B', 10));  
 *fclose*($fp);  
});

fopen

fread/fgets fwrite/fputs

file\_get\_contents、file\_put\_contents

Unlink mkdir rmdir

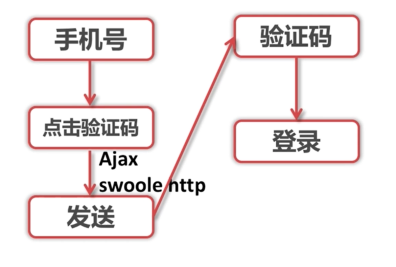
## Chan

**一个进程开辟了很多的协程，协程之间可以通过chan进行通讯 chan里面的数据是存放在内存里面的，先进先出的原则 ，可以用来存放字符串、整形。**

<?php  
$chan = new Co\Channel(10);  
go(function () use ($chan) {  
 $chan->push(['a' => 'a']);  
});  
  
go(function () use ($chan) {  
 $chan->push(['b' => 'b']);  
});  
  
go(function () use ($chan) {  
 $a = $chan->pop();  
 *print\_r*($a);  
 $len = $chan->length();  
 *print\_r*('--');  
 *print\_r*($len);  
});

# 项目

## 思路



1. **前端点击验证码，服务端收到信号，进行**
2. **拼接随机字符串+手机号，发送给sdk,并且把相关的信息存在缓存redis**
3. **sdk发送验证码到手机号**
4. **客户端进行提交，服务端收到，提取redis的信息进行验证对比**