# 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 test

Make install

显示



扩展



## 环境变量



## 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。 |

# 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



## 操作

开放端口

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

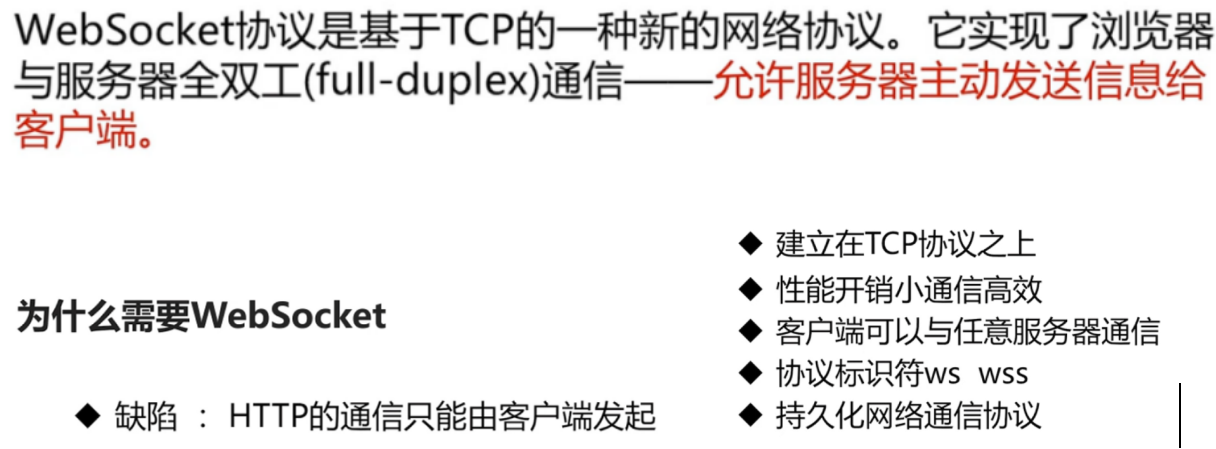
<?php  
$http = new swoole\_http\_server("0.0.0.0", 8811);  
  
$http->set([  
 'enable\_static\_handler' => true,  
 //设置根目录这里根据你自己的路径来写 后面的代码就不会执行了 直接去找静态文件了  
 'document\_root' => "/home/wwwroot/l2.l2.l2/example",  
]);  
  
$http->on('request', function (*$request*, *$response*) {  
 echo 'nihao';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