Table of Contents

Introduction

swoole文档

1. 入门指引

1. 入门指引

1.1 同步异步基础

1.1 同步异步基础

EventLoop方式协程

1.1.1 EventLoop方式协程

多进程方式协程

多进程方式协程

会阻塞的函数

会阻塞的函数

半协程函数介绍

半协程函数介绍

1.2 多进程管理

1.2 多进程管理

2. Server

2. Server

2.1 函数列表

2.1 函数列表

__construct

构造函数swoole_server::__construct

用途

创建一个swoole server资源对象.

适用范围及版本限制

- Swoole 任何版本
- PHP-FPM内请勿使用

函数原型

```
$serv = new swoole_server(string $host, int $port, int $mode = SWOOLE
    int $sock_type = SWOOLE_SOCK_TCP);
```

参数

参数	类型	必填	默认值	用途及注意事项
host	String	Y	无	服务器监听IP地址，用0.0.0.0或0:0:0:0:0:0:0:0表示监听所有地址
port	Int	Y	无	服务器监听端口，取值范围为1~65535
mode	Int	N	SWOOLE_PROCESS	服务器运行模式，目前有三种运行模式
sock_type	Int	N	SWOOLE_SOCK_TCP	Socket类型支持：TCP/UDP、TCP6/UDP6、UnixSock Stream/Dgram 6种

注意事项

- 监听端口小于1024需要Root权限，不推荐使用低于1024端口
- 如果期望监听的端口被其它服务占用server->start会失败
- 1.7.11后增加了对Unix Socket的支持
- 构造函数中的参数与swoole_server::addlistener中是完全相同
- Swoole1.6版本之后PHP版本去掉了线程模式，原因是php的内存管理器在多线程下容易发生错误
- SWOOLE_BASE模式没有进程管理进程，如果使用了Process需要自行Kill
- 线程模式仅供C++中使用
- BASE模式在1.6.4版本之后也可是使用多进程，设置worker_num来启用

代码样例

```
//创建一个server，监听9560端口，多进程模式，提供TCP协议通讯服务
$server = new swoole_server("0.0.0.0",9560,SWOOLE_PROCESS,SWOOLE_SOCK
$server->start();
```

__construct

其它相关知识

- 启用加密通讯请参考：SSL启用
- Unix Socket使用介绍：Unix Socket支持
- 高并发性能服务必须优化Linux内核：Linux内核优化
- mode属性介绍：服务端三种运行模式介绍
- 通讯协议的设计及场景

set

设置服务器配置 swoole_server->set

用途

swoole_server->set函数用于设置swoole_server运行时的各项参数。服务器启动后通过$serv->setting来访问set函数设置的参数数组

适用范围及版本限制

- Swoole 服务端 任何版本

函数原型

```
function swoole_server->set(array $setting);
```

注意事项

- 此函数必须在server->start之前调用
- 多端口的时候set必须针对不同listner返回的对象进行设置

代码样例

```
//创建一个server，监听9560端口，多进程模式，提供TCP协议通讯服务
$server = new swoole_server("0.0.0.0",9560,SWOOLE_PROCESS,SWOOLE_SOCI
//设置配置参数
$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,
));
$server->start();
```

参数说明

参数值都为key=>value方式组织 具体的参数用途及说明请参考以下其他相关知识连接

其它相关知识

- 多端口监听相关设置限制：多端口监听
- 服务器配置介绍服务器配置

set

on

注册事件回调 swoole_server->on

用途

注册Server的事件回调函数。

适用范围及版本限制

- 适用于任何版本的swoole
- php-fpm 方式并不支持server

函数原型

```
bool swoole_server->on(string $event, mixed $callback);
```

参数

参数	类型	必填	默认值	用途及注意事项
event	string	yes	--	回调的名称, 大小写不敏感，具体内容参考回调函数列表，事件名称字符串不要加on
callback	mixed	yes	--	回调的PHP函数，可以是函数名的字符串，类静态方法，对象方法数组，匿名函数。

注意事项

- 回调的名称大小写不敏感，不要加on
- 请在server->start之前执行

代码样例

```
$serv = new swoole_server("127.0.0.1", 9501);
$serv->on('connect', function ($serv, $fd){
    echo "Client:Connect.\n";
});
$serv->on('receive', function ($serv, $fd, $from_id, $data) {
    $serv->send($fd, 'Swoole: '.$data);
    $serv->close($fd);//非长连接请求可以返回结果后直接关闭，长连接请求close
});
$serv->on('close', function ($serv, $fd) {
    echo "Client: Close.\n";
});
$serv->start();
```

其它相关知识

on

- 相关事件回调列表及介绍

addListener

addListener

addProcess

addProcess

listen

listen

start

start

reload

reload

3. Client

3. Client

3.1 函数列表

3.1 函数列表

__construct

构造函数swoole_client::__construct

用途

创建一个swoole client资源对象.

适用范围及版本限制

- Swoole 任何版本
- PHP-FPM / CLI 均可使用

函数原型

```
swoole_client->__construct(int $sock_type, int $is_sync = SWOOLE_SOCI
```

参数

参数	类型	必填	默认值	用途及注意事项
sock_type	Int	Y	无	表示socket的类型，如TCP/UDPCP6/UDP6、UnixSock Stream/Dgram
is_sync	Int	N	SWOOLE_SOCK_SYNC	表示同步阻塞还是异步非阻塞，默认为同步阻塞
key	String	N	IP:PORT	用于长连接的Key，默认使用IP:PORT作为key。相同key的连接会被复用

代码样例

```
//使用同步阻塞 创建一个TCP的client
$client = new swoole_client(SWOOLE_SOCK_TCP);$client->connect("192.1(
```

注意事项

- 如果server启用了SSL加密，那么new client的时候 sock_type参数需要 | SWOOLE_SSL 来启用SSL加密。
- 可以使用swoole提供的宏来之指定类型，请参考 Swoole常量定义

在php-fpm/apache中创建长连接

```
$cli = new swoole_client(SWOOLE_TCP | SWOOLE_KEEP);
```

sock_type参数加入SWOOLE_KEEP标志后，创建的TCP连接在PHP请求结束或者调用$cli->close时并不会关闭。下一次执行connect调用时会复用上一次创建的连接。长连接保存的方式默认是以ServerHost:ServerPort为key的。可以再第3个参数内指定key。

注意事项

- SWOOLE_KEEP 只允许用于同步客户端
- swoole_client在unset时会自动调用close方法关闭socket
- 异步模式unset时会自动关闭socket并从epoll事件轮询中移除SWOOLE_KEEP
- 长连接模式在1.6.12后可用，长连接的$key参数在1.7.5后增加

在swoole_server中使用swoole_client

- 必须在事件回调函数中使用swoole_client，不能在swoole_server->start前创建 如在onWorkerStart、onRecv等回调函数中。
- swoole_server可以用任何语言编写的 socket client来连接。同样swoole_client也可以去连接任何语言编写的socket server。

set

设置客户端配置 swoole_client->set

用途

swoole_client->set函数用于设置客户端参数，必须在connect前执行。swoole-1.7.17为客户端提供了类似swoole_server的自动协议处理功能。通过设置一个参数即可完成TCP的自动分包。

适用范围及版本限制

- Swoole 客户端 任何版本

函数原型

```
function swoole_client->set(array $setting);
```

示例**1** 结束符检测

```
$client->set(
    array(
        'open_eof_check'     => true,
        'package_eof'        => "\r\n\r\n",
        'package_max_length' => 1024 * 1024 * 2,
    )
);
```

示例**2** 长度检测

```
$client->set(
    array(
        'open_length_check'       => 1,
        'package_length_type'     => 'N',
        'package_length_offset'   => 0,
        'package_body_offset'     => 4,
        'package_max_length'      => 2000000,
    )
);
```

示例**3** 设置**Socket**缓存区尺寸

```
$client->set(
     array(
        'socket_buffer_size'   => 1024*1024*2, //2M缓存区
    )
);
```

包括socket底层操作系统缓存区、应用层接收数据内存缓存区、应用层发送数据内存缓冲区

示例**4** 关闭**Nagle**合并算法

```
$client->set(
    array(
        'open_tcp_nodelay' => true,
    )
);
```

示例**5 SSL/TLS**证书

```
$client->set(
     array(
        'ssl_cert_file' => $your_ssl_cert_file_path,
        'ssl_key_file'  => $your_ssl_key_file_path,
     )
);
```

示例**6** 绑定**IP**和端口

```
$client->set(
    array(
        'bind_address' => '192.168.1.100',
        'bind_port'    => 36002,
    )
);
```

使用说明

- 目前支持open_length_check和open_eof_check2种自动协议处理功能，参考swoole_server中的配置选项
- 启用了自动协议后，同步阻塞客户端recv方法将不接受长度参数，每次必然返回一个完整的数据包
- 启用了自动协议后，异步非阻塞客户端onReceive每次必然返回一个完整的数据包

参数说明

参数值都为key=>value方式组织

具体的参数用途及说明请参考以下其他相关知识连接

其它相关知识

- 客户端配置介绍客户端配置

set

connect

构造函数swoole_client::connect

send

recv

客户端接收服务器数据 swoole_client->recv

用途

4. Process

swoole-1.7.2增加了一个进程管理模块，用来替代PHP的pcntl扩展。

PHP自带的pcntl，存在很多不足，如

- pcntl没有提供进程间通信的功能
- pcntl不支持重定向标准输入和输出
- pcntl只提供了fork这样原始的接口，容易使用错误
- swoole_process提供了比pcntl更强大的功能，更易用的API，使PHP在多进程编程方面更加轻松。

swoole_process提供了如下特性：

- swoole_process提供了基于unixsock的进程间通信，使用很简单只需调用write\/read或者push\/pop即可
- swoole_process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
- swoole_process允许用于fpm\/apache的Web请求中
- 配合swoole_event模块，创建的PHP子进程可以异步的事件驱动模式
- swoole_process提供了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

1.8.0或更高swoole_process只能在cli（命令行）环境中使用

swoole_process::__construct

创建子进程

```
int swoole_process::__construct(mixed $function, $redirect_stdin_std
```

- $function，子进程创建成功后要执行的函数
- $redirect_stdin_stdout，重定向子进程的标准输入和输出。 启用此选项后，在进程内echo将不是打印屏幕，而是写入到管道。读取键盘输入将变为从管道中读取数据。 默认为阻塞读取。
- $create_pipe，是否创建管道，启用$redirect_stdin_stdout后，此选项将忽略用户参数，强制为true 如果子进程内没有进程间通信，可以设置为false

$process对象在销毁时会自动关闭管道，子进程内如果监听了管道会收到CLOSE事件

1.7.22或更高版本允许设置管道的类型，默认为SOCK_STREAM流式

参数$create_pipe为2时，管道类型将设置为SOCK_DGRAM

在子进程中创建swoole_server

可以在swoole_process创建的子进程中swoole_server服务器程序，但为了安全必须在$process->start创建进程后，调用$worker->exec执行server的代码。

```
<?php
$process = new swoole_process('callback_function', true);
$pid = $process->start();

function callback_function(swoole_process $worker)
{
    $worker->exec('/usr/local/bin/php', array(__DIR__.'/swoole_serve
}

swoole_process::wait();
```

swoole_process->start

swoole_process->start

执行fork系统调用，启动进程。

```
int swoole_process->start();
```

创建成功返回子进程的PID，创建失败返回false。可使用swoole_errno和swoole_strerror得到错误码和错误信息。

- $process->pid 属性为子进程的PID
- $process->pipe 属性为管道的文件描述符

执行后子进程会保持父进程的内存和资源，如父进程内创建了一个redis连接，那么在子进程会保留此对象，所有操作都是对同一个连接进行的。

注意事项

因为子进程会继承父进程的内存和IO句柄，所以如果父进程要创建多个子进程，务必要等待创建完毕后再使用swoole_event_add\/异步swoole_client\/定时器\/信号等异步IO函数。

错误的代码

```
$workers = [];
$worker_num = 3;//创建的进程数

for($i=0;$i<$worker_num ; $i++){
    $process = new swoole_process('process');
    $pid = $process->start();
    $workers[$pid] = $process;
    //子进程也会包含此事件
    swoole_event_add($process->pipe, function ($pipe) use($process){
    $data = $process->read();
        echo "RECV: " . $data.PHP_EOL;
    });
}


function process(swoole_process $process){// 第一个处理
    $process->write($process->pid);
    echo $process->pid,"\t",$process->callback .PHP_EOL;
}
```

正确的代码：

```
$workers = [];
$worker_num = 3;//创建的进程数
```

```
for($i=0;$i<$worker_num ; $i++){
    $process = new swoole_process('process');
    $pid = $process->start();
    $workers[$pid] = $process;
}

foreach($workers as $process){
    //子进程也会包含此事件
    swoole_event_add($process->pipe, function ($pipe) use($process){
    $data = $process->read();
        echo "RECV: " . $data.PHP_EOL;
    });
}

function process(swoole_process $process){// 第一个处理
    $process->write($process->pid);
    echo $process->pid,"\t",$process->callback .PHP_EOL;
}
```

swoole_process->name

swoole_process->name

修改进程名称。此函数是swoole_set_process_name的别名。

```
bool swoole_process::name(string $new_process_name);
$process->name("php server.php: worker");
```

此方法在swoole-1.7.9以上版本可用

name方法应当在start之后的子进程回调函数中使用

swoole_process->exec

swoole_process->exec

执行一个外部程序，此函数是exec系统调用的封装。

```
bool swoole_process->exec(string $execfile, array $args)
```

- $execfile指定可执行文件的绝对路径，如 "\/usr\/bin\/python"
- $args是一个数组，是exec的参数列表，如 array('test.py', 123)，相当与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。子进程脱变成另外一套程序。父进程与当前进程仍然是父子进程关系。

父进程与新进程之间可以通过可以通过标准输入输出进行通信，必须启用标准输入输出重定向。

$execfile必须使用绝对路径，否则会报文件不存在错误

由于exec系统调用会使用指定的程序覆盖当前程序，子进程需要读写标准输出与父进程进行通信

如果未指定redirect_stdin_stdout = true，执行exec后子进程与父进程无法通信

swoole_process->write

swoole_process->write

向管道内写入数据。

```
int swoole_process->write(string $data);
```

- $data的长度在Linux系统下最大不超过8K，MacOS\/FreeBSD下最大不超过2K
- 在子进程内调用write，主进程会收到数据
- 在主进程内调用write，子进程会收到数据

swoole底层使用Unix Socket实现通信，UnixSocket是内核实现的全内存通信，无任何IO消耗。在1进程write，1进程read，每次读写1024字节数据的测试中，100万次通信仅需1.02秒。

管道通信默认的方式是流式，`write`写入的数据在`read`可能会被底层合并。可以设置`swoole_process`构造函数的第三个参数为`2`改变为数据报式。

MacOS\/FreeBSD可以设置`net.local.dgram.maxdgram`内核参数修改最大长度

异步模式

如果进程内使用了异步IO，比如`swoole_event_add`，进程内执行write操作将变为异步模式。swoole底层会监听可写事件，自动完成管道写入。

异步模式下如果SOCKET缓存区已满，Swoole的处理逻辑请参考 swoole_event_write

同步模式

进程内未使用任何异步IO，当前管道为同步阻塞模式，如果缓存区已满，将阻塞等待直到write操作完成。

swoole_process->read

swoole_process->read

从管道中读取数据。

```
int swoole_process->read(int $buffer_size=8192);
```

- $buffer_size是缓冲区的大小，默认为8192，最大不超过64K
- 默认read操作为流式的，write\/read的大小并不一致

这里是同步阻塞读取的，可以使用swoole_event_add将管道加入到事件循环中，变为异步模式

示例：

```
function callback_function_async(swoole_process $worker)
{
    $GLOBALS['worker'] = $worker;
    swoole_event_add($worker->pipe, function($pipe) {
        $worker = $GLOBALS['worker'];
        $recv = $worker->read();

        echo "From Master: $recv\n";

        //send data to master
        $worker->write("hello master\n");

        sleep(2);

        $worker->exit(0);
    });
}
```

swoole_process->useQueue

swoole_process->useQueue

启用消息队列作为进程间通信。

```
bool swoole_process->useQueue(int $msgkey = 0, int $mode = 2);
```

useQueue方法接受2个可选参数。

- $msgkey是消息队列的key，默认会使用ftok(**FILE**)
- $mode通信模式，默认为2，表示争抢模式，所有创建的子进程都会从队列中取数据
- 如果创建消息队列失败，会返回false。可使用swoole_strerror(swoole_errno()) 得到错误码和错误信息。

使用模式2后，创建的子进程无法进行单独通信，比如发给特定子进程。

$process对象并未执行start，也可以执行push\/pop向队列推送\/提取数据

消息队列通信方式与管道不可公用。消息队列不支持EventLoop，使用消息队列后只能使用同步阻塞模式

swoole_process->statQueue

swoole_process->statQueue

查看消息队列状态。

```
array swoole_process->statQueue();
```

- 返回一个数组，包括2项信息
- queue_num 队列中的任务数量
- queue_bytes 队列数据的总字节数

```
array(
  "queue_num" => 10,
  "queue_bytes" => 161,
);
```

swoole_process->freeQueue

swoole_process->freeQueue

删除队列。此方法与useQueue成对使用，useQueue创建队列，使用freeQueue销毁队列。销毁队列后队列中的数据会被清空。

```
function swoole_process->freeQueue();
```

swoole_process->push

swoole_process->push

投递数据到消息队列中。

```
bool swoole_process->push(string $data);
```

- $data要投递的数据，长度受限与操作系统内核参数的限制。默认为8192，最大不超过65536
- 操作失败会返回false，成功返回true

示例

```
$workers = [];
$worker_num = 2;

for($i = 0; $i < $worker_num; $i++)
{
    $process = new swoole_process('callback_function', false, false);
    $process->useQueue();
    $pid = $process->start();
    $workers[$pid] = $process;
    //echo "Master: new worker, PID=".$pid."\n";
}

function callback_function(swoole_process $worker)
{
    //echo "Worker: start. PID=".$worker->pid."\n";
    //recv data from master
    $recv = $worker->pop();

    echo "From Master: $recv\n";

    sleep(2);
    $worker->exit(0);
}

foreach($workers as $pid => $process)
{
    $process->push("hello worker[$pid]\n");
}

for($i = 0; $i < $worker_num; $i++)
{
    $ret = swoole_process::wait();
    $pid = $ret['pid'];
    unset($workers[$pid]);
    echo "Worker Exit, PID=".$pid.PHP_EOL;
}
```

swoole_process->push

swoole_process->pop

swoole_process->pop

从队列中提取数据。

```
string swoole_process->pop(int $maxsize = 8192);
```

- $maxsize表示获取数据的最大尺寸，默认为8192
- 操作成功会返回提取到的数据内容，失败返回false
- 如果队列中没有数据，pop()方法会阻塞等待

swoole_process->close

swoole_process->close

用于关闭创建的好的管道。

```
bool swoole_process->close();
```

有一些特殊的情况swoole_process对象无法释放，如果持续创建进程会导致连接泄漏。调用此函数就可以直接关闭管道，释放资源。

swoole_process->exit

swoole_process->exit

退出子进程

```
int swoole_process->exit(int $status=0);
```

$status是退出进程的状态码，如果为0表示正常结束，会继续执行PHP的shutdown_function，其他扩展的清理工作。

如果$status不为0，表示异常退出，会立即终止进程。不再执行PHP的shutdown_function，其他扩展的清理工作。

在父进程中，执行swoole_process::wait可以得到子进程退出的事件和状态码。

swoole_process::kill

swoole_process::kill

向子进程发送信号

```
int swoole_process::kill($pid, $signo = SIGTERM);
```

- 默认的信号为SIGTERM，表示终止进程
- $signo=0，可以检测进程是否存在，不会发送信号

僵尸进程

子进程退出后，父进程务必要执行swoole_process::wait进行回收，否则这个子进程就会变为僵尸进程（孤儿进程）。会浪费操作系统的进程资源。

父进程可以设置SIGCHLD信号，收到信号后执行swoole_process::wait回收退出的子进程。

swoole_process::wait

swoole_process::wait

回收结束运行的子进程。

```
array swoole_process::wait(bool $blocking = true);
$result = array('code' => 0, 'pid' => 15001, 'signal' => 15);
```

- $blocking 参数可以指定是否阻塞等待，默认为阻塞
- 操作成功会返回返回一个数组包含子进程的PID、退出状态码、被哪种信号KILL
- 失败返回false

子进程结束必须要执行wait进行回收，否则子进程会变成僵尸进程

$blocking 仅在1.7.10以上版本可用

在异步信号回调中执行**wait**

```
swoole_process::signal(SIGCHLD, function($sig) {
  //必须为false，非阻塞模式
  while($ret =  swoole_process::wait(false)) {
      echo "PID={$ret['pid']}\n";
  }
});
```

- 信号发生时可能同时有多个子进程退出
- 必须循环执行wait直到返回false

swoole_process::daemon

swoole_process::daemon

使当前进程脱变为一个守护进程。

```
bool swoole_process::daemon(bool $nochdir = false, bool $noclose = fa
```

- $nochdir，为true表示不修改当前目录。默认false表示将当前目录切换到“\/”
- $noclose，默认false表示将标准输入和输出重定向到\/dev\/null

此函数在1.7.5-stable版本后可用

swoole_process::signal

swoole_process::signal

设置异步信号监听。

```
bool swoole_process::signal(int $signo, mixed $callback);
```

- 此方法基于signalfd和eventloop是异步IO，不能用于同步程序中
- 同步阻塞的程序可以使用pcntl扩展提供的pcntl_signal
- $callback如果为null，表示移除信号监听

使用举例：

```
swoole_process::signal(SIGTERM, function($signo) {
     echo "shutdown.";
});
```

swoole_server中不能设置SIGTERM和SIGALAM信号

swoole_process::signal在swoole-1.7.9以上版本可用

信号移除特性仅在1.7.21或更高版本可用

swoole_process::setaffinity

swoole_process::setaffinity

设置CPU亲和性，可以将进程绑定到特定的CPU核上。

```
function swoole_process::setaffinity(array $cpu_set);
```

- 接受一个数组参数表示绑定哪些CPU核，如array(0,2,3)表示绑定CPU0\/CPU2\/CPU3
- 成功返回true，失败返回false

$cpu_set内的元素不能超过CPU核数

CPU-ID不得超过（CPU核数 - 1）

使用SWOOLE_CPU_NUM常量可以得到当前服务器的CPU核数

setaffinity函数在1.7.18以上版本可用

此函数的作用是让进程只在某几个CPU核上运行，让出某些CPU资源执行更重要的程序。

5. AsyncIO

6. Memory

7. HttpServer

swoole_http_server

 # swoole_http_server

swoole_http_server->setGlobal

常见问题

8. WebSocket

8.2 回调函数

回调函数

swoole_websocket_server 继承自 swoole_http_server,所以websocket_server 不仅拥有http_server的回调函数onRequest。还有 onHandShake[可选] onOpen onMessage 等回调函数，下面一一来介绍。。当然，websocket_server也可以当Http服务器来用。

```
$server = new swoole_websocket_server("0.0.0.0", 9501);

$server->on('open', function (swoole_websocket_server $server, $requ
    echo "server: handshake success with fd{$request->fd}\n";
});

$server->on('message', function (swoole_websocket_server $server, $fr
    echo "receive from {$frame->fd}:{$frame->data},opcode:{$frame->op
    $server->push($frame->fd, "this is server");
});

$server->on('close', function ($ser, $fd) {
    echo "client {$fd} closed\n";
});

$server->start();
```

短短十来行代码就可以实现一个高性能的websocket server。

onHandShake

函数原型：

```
function onHandShake(swoole_http_request $request, swoole_http_respo
```

参数	描述
$request	swoole_websocket_server对象
$response	是一个Http请求对象，包含了客户端发来的握手请求信息

说明：

- 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。
- onHandShake函数必须返回true表示握手成功，返回其他值表示握手失败
- onHandShake 事件回调是可选的。

如果设置onHandShake回调函数后将不会再触发onOpen事件，需要应用代码自行处理，（1.8.1或更高版本可以使用server->defer调用onOpen逻辑）。

---

onOpen

函数原型：

```
function onOpen(swoole_websocket_server $server, swoole_http_request
```

参数	描述
$server	swoole_websocket_server对象
$request	是一个Http请求对象，包含了客户端发来的握手请求信息

说明：

- 当有新的WebSocket客户端与本服务建立连接并完成握手后会回调此函数。
- onOpen事件函数中可以调用push向客户端发送数据或者调用close关闭连接。
- onOpen事件回调是可选的。

如果在onConnect里有代码，会先执行onConnect里的代码。

---

onMessage

函数原型：

```
function onMessage(swoole_websocket_server $server, swoole_websocket_
```

参数	描述
$server	swoole_websocket_server对象
$frame	是swoole_websocket_frame对象，包含了客户端发来的数据帧信息

说明：

- 当服务器收到来自客户端的数据帧时会回调此函数。
- onMessage回调必须被设置，未设置服务器将无法启动 。

swoole_websocket_frame

这个对象共有4个属性，分别是

属性名	描述
fd	客户端的socket id,要推给那个客户端就靠它了。
data	客户端传的数据内容，可以是文本也可以是二进制数据，可以通过opcode的值来判断
opcode	WebSocket的OpCode类型，
finish	表示数据帧是否完整，一个WebSocket请求可能会分成多个数据帧进行发送

最常用的应该就是 $frame->fd 和 $frame->data。

> $data 如果是文本类型，编码格式必然是UTF-8，这是WebSocket协议规定的

opcode与数据类型

- WEBSOCKET_OPCODE_TEXT = 0x1 ，文本数据
- WEBSOCKET_OPCODE_BINARY = 0x2 ，二进制数据

点击查看聊天室完整代码样例：

8.3 预定义常量

预定义常量

WebSocket数据帧类型

- WEBSOCKET_OPCODE_TEXT = 0x1，UTF-8文本字符数据
- WEBSOCKET_OPCODE_BINARY = 0x2，二进制数据

WebSocket连接状态

- WEBSOCKET_STATUS_CONNECTION = 1，连接进入等待握手
- WEBSOCKET_STATUS_HANDSHAKE = 2，正在握手
- WEBSOCKET_STATUS_FRAME = 3，已握手成功等待浏览器发送数据帧

8.4 常见问题

常见问题

如何判断连接是否为WebSocket客户端

通过使用$server->connection_info获取连接信息，返回的数组中有一项为 websocket_status，根据此状态可以判断是否为WebSocket客户端。

```
var_dump($server->connection_info($fd));
```

- WEBSOCKET_STATUS_CONNECTION = 1，连接进入等待握手
- WEBSOCKET_STATUS_HANDSHAKE = 2，正在握手
- WEBSOCKET_STATUS_FRAME = 3，已握手成功等待浏览器发送数据帧

9.1 swoole的实现

9.2 Reactor线程

9.3 Manager进程

9.4 Worker进程