vSwoole 开发手册

微服务器开发框架

目　录

命令帮助

序言

vSwoole

vSwoole是对Swoole扩展的超轻量级封装框架，主要用于开发基于PHP的微服务，用于快速方便构建适合自己业务的定制化服务。支持服务单机部署和分布式部署。

开发环境要求：

- PHP 7.0以上
- swoole扩展2.0以上
- rekafka扩展（用于使用kafka客户端，不使用kafka，可以不安装）
- hiredis扩展（用于异步redis，不使用异步redis，可以不安装）
- Inotify扩展（用于监控代码修改，服务自动重载进程，不使用监控，可以不安装）

基础

目录结构

```
vSwoole（服务框架目录）
├─application                    框架应用根目录
│   │
│   ├─client                     客户端应用目录
│   │   ├─logic                  客户端应用逻辑目录
│   │   ├─ClientName.php         客户端应用基础层
│   │   └─...
│   └─server                     服务端应用目录
│       ├─logic                  服务端应用逻辑目录
│       ├─ServerName.php         服务端应用基础层
│       └─...
│
├─configs                        框架配置根目录
│   │
│   ├─const.php                  服务注册
│   ├─config.php                 基础配置
│   ├─redis.php                  缓存配置
│   ├─db.php                     数据库配置
│   ├─server.php                 基础服务配置
│   └─...
│
├─core                           服务核心根目录
│   │
│   ├─client                     客户端核心目录
│   │   ├─ServerClient.php       客户端核心类
│   │   └─...
│   └─server                     服务端核心目录
│       ├─ServerServer.php       服务端核心类
│       └─...
│
├─data                           框架数据根目录
│   │
│   ├─pid                        服务进程PID目录
│   │   ├─Server_Manager.pid     服务管理进程PID
│   │   ├─Server_Master.pid      服务主进程PID
│   │   └─...
│   └─cache                      缓存文件目录
│       ├─cache.php              缓存文件
│       └─...
│
├─library                        框架核心目录
│   │
│   ├─client                     客户端核心目录
│   │   └─Client.php             客户端底层抽象模型
│   │─server                     服务端核心目录
│   │   └─Server.php             服务端底层抽象模型
```

```
|    ├─common                    核心工具类目录
|    |    ├─cache                服务缓存方法目录
|    |    ├─exception            服务异常处理类目录
|    |    ├─kafka                服务kafka类目录
|    |    ├─reflection           反射类目录
|    |    ├─Build.php            服务核心层构建类
|    |    ├─Command.php          服务管理命令类
|    |    ├─Config.php           服务配置类
|    |    ├─File.php             服务文件处理类
|    |    ├─Log.php              服务日志处理类
|    |    ├─Process.php          服务进程管理类
|    |    ├─Redis.php            服务缓存处理类
|    |    ├─Request.php          服务客户端请求处理类
|    |    ├─Respinse.php         服务客户端响应处理类
|    |    ├─Task.php             服务异步任务处理类
|    |    ├─Utils.php            服务其他工具类
|    |    └─...
|    ├─conf                      框架核心配置目录
|    |    ├─config.php           框架核心配置
|    |    └─const.php            框架核心常量配置
|    └─Init.php                  框架初始化引导类
|
├─log                            框架日志目录
|    |
|    ├─client                    客户端日志目录
|    |    └─...
|    └─server                    服务端日志目录
|         └─...
|
└─public                         框架公共目录
     |
     ├─static                    框架静态文件目录
     |    └─...
     ├─index.php                 第三方框架接入文件
     ├─client.php                客户端接口入口文件
     └─swoole.php                服务端管理入口文件
```

框架特点

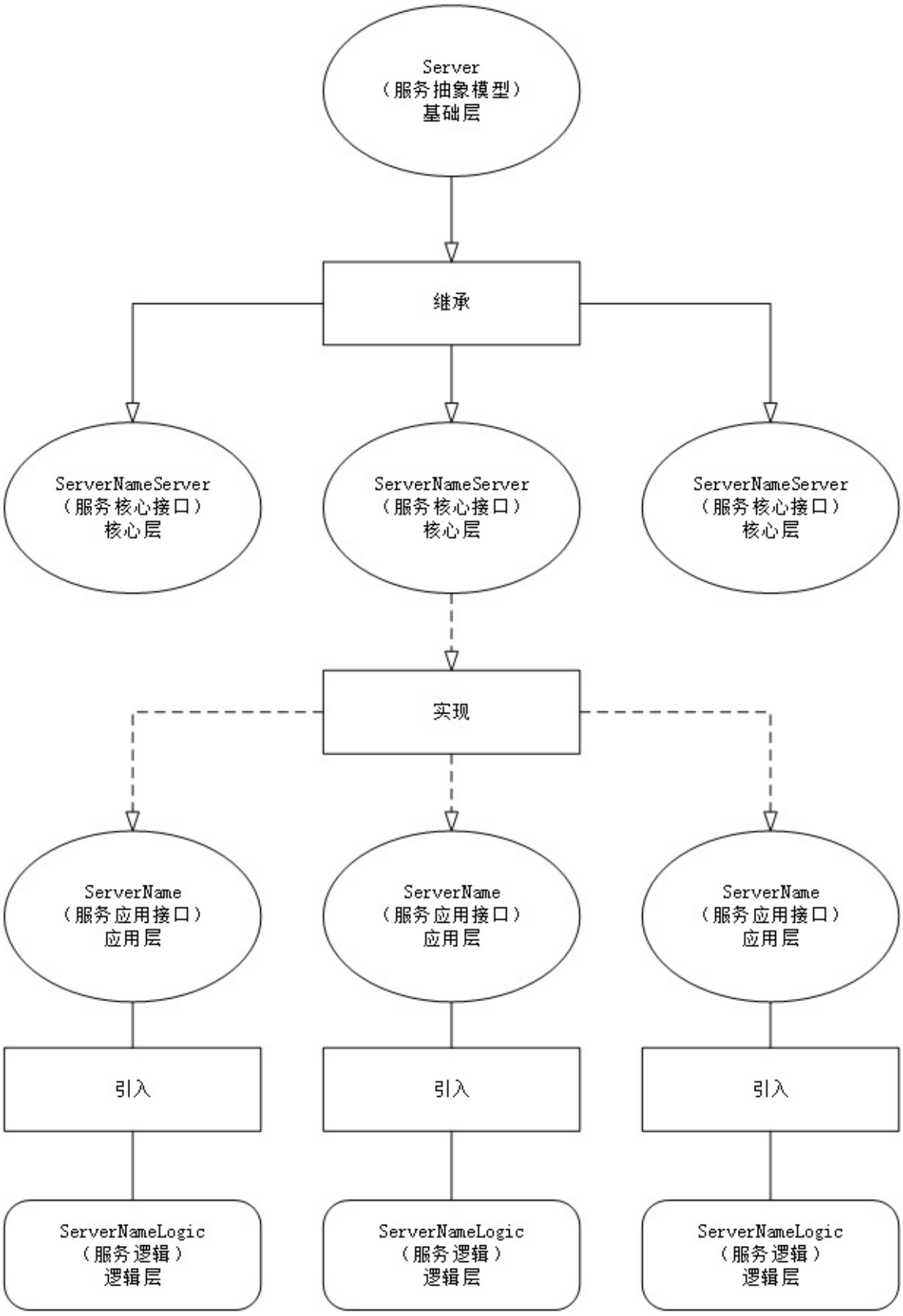
- 客户端与服务端模块化分离，减少框架耦合性
- 服务与框架分离，易于框架升级
- 配置文件模块化，增强可维护性
- 服务进程号统一管理
- 服务模块核心层，基础层，逻辑层分离，增强可开发管理性
- 框架功能组件化，易于统一调用
- 框架配置缓存化，加速配置读取
- 组件对象静态化，加速组件调用
- 组件对象单例化，减少内存消耗
- 框架异常提示优化，反射异常源代码，直观显示错误异常
- 管理客户端与用户客户端端口区分连接，服务接口也以端口区分，加强连接安全性
- 文件IO同步与异步调用组件化，统一调用
- 缓存同步与异步操作组件化，以参数区别调用
- 服务支持分布式部署
- Timer定时器服务，异步任务毫秒数定时处理
- Task，Process等扩展底层调用方法封装成组件，便于使用
- 服务管理，如重启，关闭，启动等，可通过管理客户端调用服务接口，也可通过命令行管理
- 支持命令行构建服务核心类文件，节省开发时间
- 支持Inotify监听代码改动，自动重启工作进程
- 服务对外以接口方式提供服务和管理
- 服务配置参数框架底层自动优化
- 框架可独立使用，可作为组件被其他框架引入
- 命令行提供：install（安装目录），clear（清理日志），build（构建服务核心文件），start（启动指定服务），reload（重启指定服务），shutdown（关闭指定服务），help（帮助）等命令

开发规范

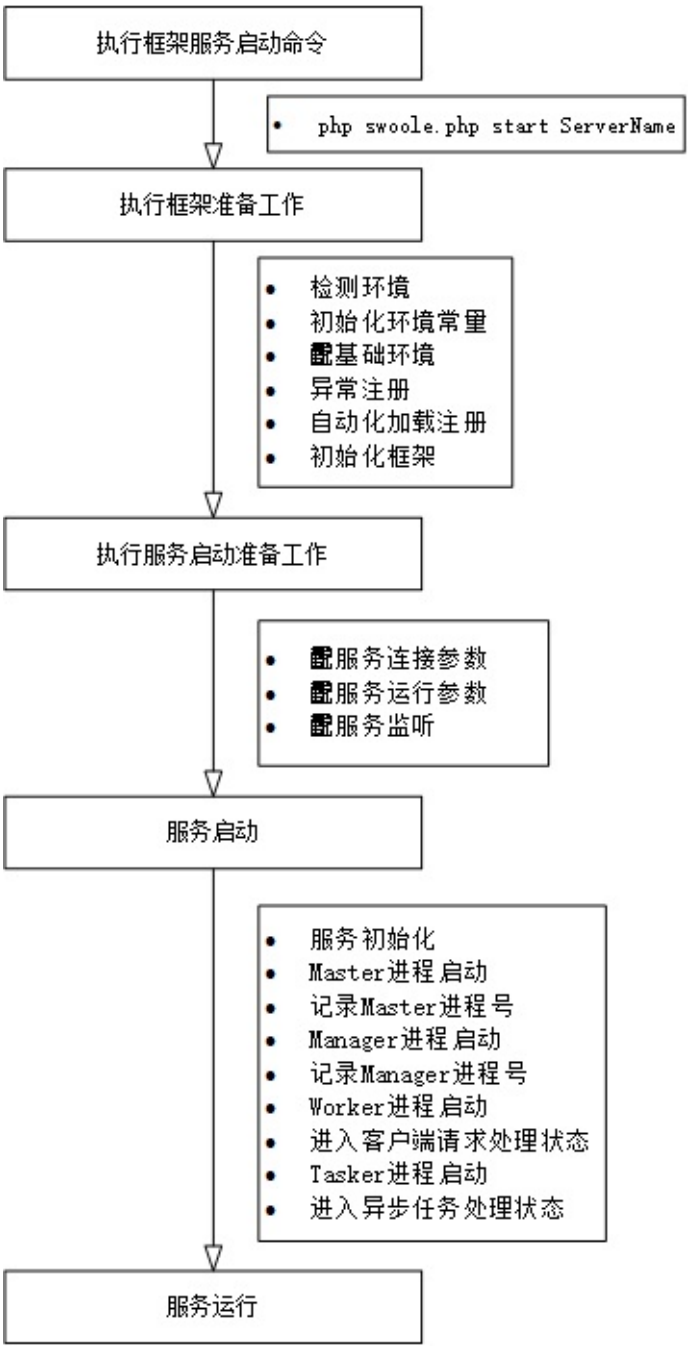
- 类名和类文件名保持一致，以大驼峰方式命名。
- 类命名空间从框架根目录开始命名，且需与路径完全一致（否则第三方接入后，自动引入类文件会冲突失效）。
- 框架初始目录名不可以改变，目录名为小写。
- 常量以全字母大写命名，以下划线'_'分隔，以VSWOOLE开头。

架构

服务架构


Server
（服务抽象模型）
基础层
维承
ServerNameServer
（服务核心接口）
核心层
ServerNameServer
（服务核心接口）
核心层
ServerNameServer
（服务核心接口）
核心层
实现
ServerName
（服务应用接口）
应用层
ServerName
（服务应用接口）
应用层
ServerName
（服务应用接口）
应用层
引入
引入
引入
ServerNameLogic
（服务逻辑）
逻辑层
ServerNameLogic
（服务逻辑）
逻辑层
ServerNameLogic
（服务逻辑）
逻辑层

服务周期


执行框架服务启动命令
php swoole.php start ServerName
执行框架准备工作
检测环境
初始化环境常量
基础环境
异常注册
自动化加载注册
初始化框架
执行服务启动准备工作
服务连接参数
服务运行参数
服务监听
服务启动
服务初始化
Master进程启动
记录Master进程号
Manager进程启动
记录Manager进程号
Worker进程启动
进入客户端请求处理状态
Tasker进程启动
进入异步任务处理状态
服务运行

核心

Cache

Cache

Cache用于缓存管理，提供三种缓存方式：

- Redis redis缓存，需要安装hiredis扩展
- File 文件缓存
- Table 内存表缓存

Redis

同步：

```
/**
 * 同步reids
 * @param array $options 连接配置
 * @param is_sync true
 * /
$redis = Redis::getInstance(array $options = [], true);

/**
 * 获取同步客户端的缓存前缀
 * @return string
 */
$redis->getRedisPrefix();

/**
 * redis操作魔术方法，自动映射
 */
$redis->__call($name, $arguments);
```

异步：

基于swoole_redis，依赖于hiredis扩展

```
/**
 * 异步reids
 * @param array $options 连接配置
 * @param bool $is_sync false
 * @param callable $callback 回调函数
 * /
Redis::getInstance(array $options = [], false,callable $callback);

/**
 * 异步redis的操作必须在回调函数中写，支持类方法，函数，匿名函数等回调函数写法
```

```
 * 回调函数提供两个参数
 * @param $redis redis操作句柄
 * @param $get_redis_key 获取redis实际缓存key函数，会返回一个拼接配置中缓存前缀的实际缓存key值
 */
Redis::getInstance(['host'=>'127.0.0.1','port'=>6379,'password'=>''], false,function ($redis,$get_redis_key) {
    $redis->get($get_redis_key('vswoole'));
});
```

File

缓存文件位于根目录下的data/cache/目录下。

```
/**
 * 设置缓存
 * @param string $key 缓存key
 * @param null $value 缓存value
 * @param string $prefix 缓存前缀
 * @param int $expire 缓存有效期
 * @return bool|int 返回设置状态
 */
 File::set(string $key, $value = null, string $prefix = '', int $expire = 0);

//缓存设置
//支持所有PHP基本数据类型，除匿名函数以及带匿名函数的类
//设置字符串
File::set('a',1);
//设置数组
File::set('b',[1,2,3]);
//删除设置
File::set('a',null);
//带缓存前缀
File::set('c',1,'vswoole_');
//带有效期，单位为s，0为永久生效，负值为立即失效
File::set('d',1'vswoole',60);

//缓存读取
File::get(string $key, $prefix = '')

//如果缓存未过期，返回对应的数据
//如果缓存key过期或缓存key不存在，返回null
File::get('a');
File::get('c','vswoole_');
```

Table

Table缓存基于swoole_table，是将数据存储在内存中，服务进程关闭，数据将会销毁，适用于非重要或无需物

理物理存储的数据存储。

优点：Table缓存直接操作内存，因此IO效率很高。

缺点：需要在服务启动前创建缓存表，不能随时创建。

```
/**
 * 实例化缓存Table对象
 * @param int $table_size 数据可存储条数
 */
$table = new Table(int $table_size = 1024);

/**
 * 创建缓存表
 * @param array $column 缓存表结构字段
 * @param string $field 表字段名称
 * @param string $field_type 表字段类型[string/\swoole_table::TYPE_STRING, int/\s
woole_table::TYPE_INT, float/\swoole_table::TYPE_FLOAT]
 * @param int $field_length 表字段最大长度
 */
 $table->create([$field=>[$field_type,'$filed_legth]]);

 //举例
 $table->create(['name'=>['string',16],'age'=>['int',2]]);

/**
 * 显示已设置的内存表结构
 * @return string
 */
$table->show();

/**
 * 获取内存表物理大小
 * @return int
 */
$table->getTableSize();

/**
 * 获取内存表内存大小
 * @return mixed
 */
$table->getTableMemorySize();

/**
 * 获取内存表所有数据
 * @return array
 */
$table->getAll();

/**
```

```
 * 删除内存表所有数据
 * @return int
 */
 $table->deleteAll();

/**
 * 获取内存表对象实例
 * @return null|\swoole_table
 */
$table->getTable();

/**
 * 魔术方法，执行原生方法
 * @param $name
 * @param $arguments
 * @return mixed
 */
$table->__call($name, $arguments);
```

Exception

Exception

Exception用于异常处理，提供三种异常状态：

- ClassNotFoundException：类文件未找到异常类
- ErrorException：错误异常类
- Exception：异常处理基类

其他异常类继承异常处理基类，异常处理基类继承PHP核心异常处理基类，代码中的异常可以用原生异常类和框架提供的异常类抛出。
Exception类提供三个对外方法：

```
/**
 * 报告异常
 * @param Throwable 对象
 */
Exception::reportException($e);

/**
 * 报告错误
 * @param Throwable 对象
 */
Exception::reportError($e);
```

- 开启debug模式，这两个方法会向浏览器或命令行输出格式化后的异常错误信息；
- 开启log模式，这两个方法会记录异常错误信息到日志文件；
- 不同点在于，reportException不会中断脚本，reportError会立即中断脚本

```
/**
 * 获取异常错误信息
 * @param Throwable 对象
 * @return string
 */
Exception::getException($e);
```

Kafka

Kafka

Kafka用于Kafka服务的管理，提供消费者和成产者客户端。服务器需要安装librdkafka，PHP需要安装rdkafka扩展。

生产者

```
/**
 * 创建生产者对象
 * /
$producer = new Producer([
    //kafka实例地址, 多个地址用逗号间隔
    'broker_list' => '127.0.0.1:9092',
    //kafka日志级别
    'log_level'   => LOG_DEBUG
]);

/**
 * 创建话题
 */
$topic = $producer->topic(string $topic = '', TopicConf $topicConf = null);

/**
 * 生产消息
 */
$topic->produce(string $message = '');
```

低级消费者

```
/**
 * 创建消费者者对象
 * /
$consumer = new Consumer([
         //kafka实例地址, 多个地址用逗号间隔
        'broker_list' => '127.0.0.1:9092',
        //topic分页
        'partition'   => 0,
        // kafka日志级别
        'log_level'   => LOG_DEBUG
   ]);

/**
 * 消费者者选取消费话题
```

```
 * /
 $topic = $consumer->topic(string $topic = '', TopicConf $topicConf = null);


/**
 * 消费者消费消息
 * /
 $topic->consume(callable $callback = null, int $timeout = 120e3);
```

高级消费者

```
/**
 * 创建消费者者对象
 * /
$consumer = new GroupConsumer([
         //kafka实例地址，多个地址用逗号间隔
        'broker_list' => '127.0.0.1:9092',
        //消费者分组
        'group_id'    => 0,
         // 消费等待时间
        'timeout'     => 120e3
   ]);

/**
 * 消费者消费消息
 * /
 $consumer ->consume(array $topics = [], callable $callback = null);
```

Reflection

Reflection

Reflection用于反射管理，提供三种反射类型：

- ReflectionClass：反射类
- ReflectionFile：反射文件
- ReflectionFunction：反射函数

ReflectionClass可反射一个类的相关信息，使用方式：

```
//实例化反射类
$reflectionClass = new ReflectionClass($className);

/**
 * 获取类文件名
 * @return string
 */
$reflectionClass->getFileName();

/**
 * 获取类源代码数组
 * @return array|bool
 */
$reflectionClass->getSource();

/**
 * 获取反射类源代码
 * @param int $start_line 源码起始行数
 * @param int $end_line 源码结束行数
 * @param bool $show_line 是否显示对应行号
 * @return string
 */
$reflectionClass->getSourceCode(int $start_line = 0, int $end_line = 0, bool $s
how_line = false);

/**
 * 获取反射类开始行号
 * @return int
 */
$reflectionClass->getStartLine();


/**
```

```
 * 获取反射类结束行号
 * @return int
 */
 $reflectionClass->getEndLine();
```

ReflectionFile可反射一个文件的相关信息，使用方式：

```
//实例化反射类
$reflectionFile = new ReflectionFile($fileName);

/**
 * 获取反射文件的文件名
 * @return string
 */
$reflectionFile->getFileName();

/**
 * 获取反射文件源码数组
 * @return array|bool
 */
$reflectionFile->getSource();

/**
 * 获取反射文件源码
 * @param int $start_line 源码起始行数
 * @param int $end_line 源码结束行数
 * @param bool $show_line 是否显示对应行号
 * @return string
 */
$reflectionFile->getSourceCode(int $start_line = 0, int $end_line = 0, bool $sh
ow_line = false);

/**
 * 获取反射文件开始行号
 * @return int
 */
$reflectionFile->getStartLine();


/**
 * 获取反射文件结束行号
 * @return int
 */
$reflectionFile->getEndLine();
```

ReflectionFunction可反射一个函数的相关信息，使用方式：

```
//实例化反射类
$reflectionFunction = new ReflectionFunction($fileName);

/**
 * 获取反射函数文件名
 * @return string
 */
$reflectionFunction->getFileName();

/**
 * 获取反射函数源码数组
 * @return array|bool
 */
$reflectionFunction->getSource();

/**
 * 获取反射函数源码
 * @param int $start_line 源码起始行数
 * @param int $end_line 源码结束行数
 * @param bool $show_line 是否显示对应行号
 * @return string
 */
$reflectionFunction->getSourceCode(int $start_line = 0, int $end_line = 0, bool
 $show_line = false);

/**
 * 获取反射函数开始行号
 * @return int
 */
$reflectionFunction->getStartLine();


/**
 * 获取反射函数结束行号
 * @return int
 */
$reflectionFunction->getEndLine();
```

Build

Build

Build用于服务构建管理。

```
/**
 * @param $serverName 服务名称
 * @param $serverPort 服务绑定端口
 * @param $serverType 服务类型
 */
Build::build(string $serverName, int $serverPort = 9501, string $serverType = '
common');
```

Command

Command

Command用于提供服务命令管理。

```
/**
 * 获取命令对象实例
 * @param $server 服务对象
 */
$command = Command::getInstance(\swoole_server $server);

/**
 * 重置服务
 * @param $server_name 服务名
 * @param $callbak 执行成功后回调函数
 */
$command->reload(string $server_name, callable $callback);

/**
 * 关闭服务
 * @param $server_name 服务名
 * @param $callbak 执行成功后回调函数
 */
$command->shutdown(string $server_name, callable $callback);

/**
 * 重载日志服务
 * @param $server_name 服务名
 * @param $callbak 执行成功后回调函数
 */
$command->reloadLog(string $server_name, callable $callback);
```

Config

Config

Config用于服务配置管理。

```
/**
 * 装载配置文件
 * @param string $config_file
 * @param bool $is_force
 * @return null|Config
 */
$config = Config::loadConfig(string $config_file = 'config', bool $is_force = f
alse);

/**
 * 获取配置
 * @param string|null $configKey
 * @return array|mixed|null
 */
$config->get(string $configKey = null);

/**
 * 获取缓存的配置
 * @return array
 */
Config::getCacheConfig();

/**
 * 缓存所有配置文件
 */
Config::cacheConfig();
```

举例

```
//redis.php
return [
    //缓存连接配置（正式）
    'redis_master' => [
        'host'       => '127.0.0.1',
        'port'       => 6379,
        'password'   => '',
        'select'     => 0,
        'timeout'    => 0,
        'expire'     => 0,
        'persistent' => true,
```

```
        'prefix'     => 'VSwoole-',
    ],
];

//获取redis文件所有配置
$redis = $config = Config::loadConfig('redis');

//获取redis的redis_master节配置
$redis_master = $redis->get('redis_master');

//获取redis_master的host配置
$redis_host = $redis->get('redis_master.host');

//加载所有配置
$redis = $config = Config::loadConfig('*');

//获取redis_master的host配置
$redis_host = $redis->get('redis.redis_master.host');
```

Curl

Curl

Curl 用于发送网络请求资源。

```
    /**
     * 创建Curl对象
     */
    $curl = new Curl([
        CURLOPT_TIMEOUT        => 30,
        CURLOPT_HEADER         => false,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_SSL_VERIFYHOST => false,
        CURLOPT_HTTPHEADER     => array("Content-type: text/html; charset=utf-8"
)
    ]);

    /**
     * GET方式访问
     * @param string $url
     * @param array $param
     * @return mixed
     */
    $curl->get(string $url, array $param = []);

    /**
     * POST方式访问
     * @param string $url
     * @param array $param
     * @return mixed
     */
    $curl->post(string $url, array $param = []);

    /**
     * 关闭CURL
     */
    $curl->close();
```

File

File

File用于文件相关操作处理。

```
    /**
     * 异步读取文件
     * @param string $filename
     * @param callable $callback
     * @throws \ReflectionException
     */
    File::read(string $filename, callable $callback);

    /**
     * 异步写入文件
     * @param string $filename
     * @param string $content
     * @param int $mode
     * @param callable|null $callback
     */
    File::write(string $filename, string $content = '', int $mode = 0, callable
 $callback = null);

    /**
     * 同步读取文件内容
     * @param string $filename
     * @param callable|null $callback
     */
    File::get(string $filename, callable $callback = null);

    /**
     * 同步写入文件内容
     * @param string $filename
     * @param string $content
     * @param int $mode
     * @param callable|null $callback
     */
    File::put(string $filename, string $content = '', int $mode = 0, callable $
callback = null);

    /**
     * 异步执行命令
     * @param string $command
     * @param $callback
     * @param array $arguments
     * @throws \ReflectionException
```

```
     */
    File::exec(string $command, callable $callback = null);
```

Http

Http

Http 是基于Swoole的异步网络资源请求方式。

```
    /**
     * 创建Http对象
     */
    $http = new Http([
        //是否启动dns解析
        'enable_dns_lookup' => false,
        //请求服务器主机
        'request_domain'    => '',
        //请求服务器端口
        'request_port'      => 80,
        //请求回调
        'request_callback'  => null,
        //如果端口是443，需要配置ssl证书
        'ssl'               => [
            'ssl_cert_file' => '',
            'ssl_key_file'  => ''
        ]
    ]);

     /**
      * 连接请求
      */
    $http->connect();
```

支持魔术方法，即可调用swoole_http_client的所有接口。

Inotify

Inotify

inotify用于文件监控管理：

依赖扩展：inotify。

```
/**
 * 初始化Inotify实例
 * @param int|null $mask
 * @return null|Inotify
 */
$inotify = Inotify::getInstance(int $mask = IN_CREATE | IN_DELETE | IN_MODIFY);


/**
 * 启用监听
 * @param array $pathname 需要监控的文件名或目录，多个以数组形式传入
 * @param callable|null $callback 触发监听事件的回调函数
 */
$inotify->watch($pathname = [], callable $callback = null);

//举例
$inotify->watch([__DIR__.'/a.php',__DIR__.'/b'], function () {
    echo '文件发生了变化';
});
```

Log

Log

Log用于读写日志管理。

```
/**
 * 异步写日志
 * @param string $content 日志内容
 * @param string $fileName 日志文件名
 * @param int $mode 是否追加
 * @param callable|null $callback 回调函数
 */
Log::write(string $content = '', string $fileName = 'vSwoole.log', int $mode =
FILE_APPEND, callable $callback = null);

//举例
Log::write('测试','test.log', 0, function () {
    echo '日志写入成功';
});
```

```
/**
 * 同步记录日志
 * @param string $content 日志内容
 * @param string $fileName 日志文件名
 * @param int $mode 是否追加
 * @return bool|int 记录状态
 */
Log::save($content = '', string $fileName = 'vSwoole.log', int $mode = FILE_APP
END);

//举例
Log::save('测试','test.log', FILE_APPEND);
```

Process

Process

Process用于对Swoole的进程管理。

```
    /**
     * 设置进程配置参数，获取进程管理实例
     * @param array $options
     * @return null|static
     */
    $process = Process::getInstance([
        //进程内是否允许标准输入输出
        'redirect_stdin_stdout'     => true,
        //是否将标准输出转入管道
        'create_pipe'               => true,
        //是否启用内存保护
        'enable_memory_security'    => true,
        //内存保护阈值
        'memory_security_threshold' => 204800
    ]);

    /**
     * 添加进程回调函数，并创建子进程
     * @param $callback
     * @param array $arguments
     * @return bool|int
     */
    $process->add($callback, array $arguments = []);

     /**
     * 获取已创建子进程
     * @param int $pid
     * @return mixed|null
     */
    $process->getProcess(int $pid = -1);

    /**
     * 获取已创建的子进程列表
     * @return array
     */
     $process->getProcessList();

    /**
     *  监听子进程状态，子进程退出后，释放子进程
     * @param bool $is_blocking
  true 为同步阻塞 false为异步监听
```

```
     */
    Process::signalProcess(bool $is_blocking = true);

    /**
     * 终止指定子进程
     * @param int $pid
     */
    Process::killProcess($pid = -1);
```

Request

Request

Request用于获取请求参数。

```
    /**
     * 单例模式获取请求实例
     * @return static
     */
    $request = Request::getInstance();

    /**
     * 获取参数
     * @param string $name
     * @param null $default
     * @param string $filter
     * @return array|null
     */
   $request->param(string $name = '', $default = null, $filter = '');
```

Response

Response

Response用于响应请求。

```
    /**
     * 输出指定格式内容
     * @param string $data
     * @param string $format
     */
    Response::return($data = '', string $format = 'json');
```

Task

Task

Task用于对Swoole的Task进程管理。

```
    /**
     * 执行异步任务投递
     * @param \swoole_server $server
     * @param $callback
     * @param array $arguments
     * @param int $dst_worker_id
     * @throws \ReflectionException
     */
    Task::task(\swoole_server $server, $callback, array $arguments = [], int $d
st_worker_id = -1);

    /**
     * 执行异步任务处理
     * @param \swoole_server $server
     * @param array $data
     */
   Task::execute(\swoole_server $server, array $data = []);

    /**
     * 执行异步任务执行完成回调
     * @param array $data
     */
    Task::finish(array $data = []);
```

Timer

Timer

Timer用于对Swoole定时器管理。

```
    /**
     * 间隔时钟定时器
     * @param int $ms
     * @param callable|null $callback
     */
    Timer::tick(int $ms = 0, callable $callback = null);

    /**
     * 延迟时钟定时器
     * @param int $ms
     * @param callable|null $callback
     */
   Timer::after(int $ms = 0, callable $callback = null);

    /**
     * 清除定时器
     * @param int $timer_id
     */
    Timer::clear(int $timer_id);
```

Utils

Utils

Utils用于提供框架各种便捷方法。

```
    /**
     * 获取本机服务器Ip地址
     * @return string
     */
    Utils::getServerIp();

    /**
     * 获取客户端的连接端口
     * @param \swoole_server $server
     * @param $fd
     * @return string
     */
   Utils::getServerPort(\swoole_server $server, $fd);

    /**
     * 获取客户端的连接IP
     * @param \swoole_server $server
     * @param $fd
     * @return string
     */
    Utils::getClientIp(\swoole_server $server, $fd);

    /**
     * 获取客户端连接时间
     * @param \swoole_server $server
     * @param $fd
     * @return int
     */
    Utils::getClientConnectTime(\swoole_server $server, $fd);

    /**
     * 异步记录服务主进程PID
     * @param int $pid
     * @param string $pid_name
     */
     Utils::writePid(int $pid = 0, string $pid_name = '');

    /**
     * 获取指定服务器运行状态
     * @param string $host
     * @param int $port
```

```
     * @param int $timeout
     * @param int $flag
     * @return bool
     */
    Utils::getServerStatus(string $host, int $port, int $timeout = 3, int $flag =
0);

    /**
     * 设置进程别名
     * @param string $process_name
     */
    Utils::setProcessName(string $process_name = '');

    /**
     * 字节转换
     * @param int $size
     * @return string
     */
     Utils::byteConvert(int $size = 0);
```

服务

Crontab

Crontab

Crontab是用于执行定时任务的毫秒级服务。

运行模式：

1. 服务向外暴露任务添加接口。
2. 客户端以管理模式连接服务，并以固定格式字段向服务添加任务。
3. 服务在启动时会扫描任务池（在调用接口指令时，会跳过扫描），从任务池中取出已开启的任务，并分析任务-校验任务-解析任务-重组任务，最后将重组后的任务写入服务内存表。
4. 服务以特定Task进程每分钟定时读取内存表中的任务，并解析执行时间。(run task)
5. run task进程将解析后任务以进程轮询的方式发送到其余Task进程。(execute task)
6. execute task进程解析任务执行参数，根据参数创建子进程模拟多线程并发执行任务。

Http

Http

Http是基于Swoole实现的异步Http服务，可以替代PHP-FPM或PHP-CGI，充当Web服务器。

WebSocket

WebSocket

WebSocket是基于Swoole实现的PHP WebSocket服务。

命令

构建服务

命令构建

```
php path/vswoole/public/swoole.php build ServerName [ServerPort:9501] [ServerType:common]
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

build：构建服务指令

ServerName：指定需要构建的服务名称（必要）

*服务已存在或服务相关文件存在将不会创建新文件

ServerPort：指定需要构建的服务端口（非必要）

*客户端默认9501，管理端默认8501

ServerType：指定需要构建的服务类型（非必要）

*默认common，可选common，websocket，http，udp

命令说明

执行命令后，框架会做以下事：

1. 检测同名服务是否注册，若未注册将在vSwoole\configs\const.php中注册服务
2. 检测同名服务核心文件是否存在，若不存在将创建服务核心文件 vSwoole\core\server\ServerNameServer.php
3. 检测同名服务配置文件是否存在，若不存在将创建服务配置文件vSwoole\configs\servername.php
4. 检测同名服务客户端核心文件是否存在，若不存在将创建客户端核心文件 vSwoole\core\client\ServerNameClient.php

手动构建

如果使用手动构建，需要严格按照框架开发规范创建服务文件，并在服务启动前，在vSwoole\configs\const.php文件中，对服务进行注册。

启动服务

```
php path/vswoole/public/swoole.php start ServerName
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

start：启动服务指令

ServerName：指定需要启动的服务名称（必要）

命令说明

服务不存在或服务已经启动或服务端口已被绑定，执行命令将会启动失败。

关闭服务

```
php path/vswoole/public/swoole.php shutdown ServerName
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

shutdown：关闭服务指令

ServerName：指定需要关闭的服务名称（必要）

命令说明

服务不存在或服务未启动或服务master PID文件丢失，执行命令将会关闭失败。

重载服务

```
php path/vswoole/public/swoole.php reload ServerName
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

reload：重载服务指令

ServerName：指定需要重载的服务名称（必要）

命令说明

服务不存在或服务未启动或服务manage PID文件丢失，执行命令将会重载失败。

服务重载，仅会在work进程和Task进程处于空闲状态时，由管理进程重新创建work进程和Task，以达到服务热更新，所以建议将所有逻辑代码或业务代码放在work进程中加载。

重载日志

```
php path/vswoole/public/swoole.php log
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

log：重载日志指令

命令说明

日志文件在异步服务中使用，可能会造成文件缓冲冲突或其他异常，可以使用该指令重新加载对应服务指定的日志文件。

清理日志

```
php path/vswoole/public/swoole.php clear
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

clear：清理日志指令

命令说明

该命令将会清除日志目录下的所有日志文件，不区分服务。

安装目录

```
php path/vswoole/public/swoole.php install
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

install：安装目录指令

命令说明

该命令将会自动创建框架所有必要的目录。

命令帮助

```
php path/vswoole/public/swoole.php help
```

命令解析

php：PHP服务命令

swoole.php：vSwoole框架服务脚本

help：获取帮助指令

命令说明

该命令将会获取所有指令的使用说明。