IM 系统开发文档

Briquz zgy@bz-inc.com

2015 年 9 月 24 日

version v1

目录

插图

表格

Part I

Swoole

Chapter 1

Introduction

Swoole 是一种基于 PHP 核心开发的高性能网络通信框架[1]，提供了 PHP 语言的异步多线程服务器，异步 TCP/UDP 网络客户端，异步 MySQL，数据库连接池，AsyncTask，消息队列，毫秒定时器，异步文件读写，异步 DNS 查询。

- swoole_server，高并发高性能功能强大的异步并行 TCP/UDP Server。
- swoole_client，支持同步/异步/并发的 socket 客户端实现[2]。
- swoole_event，基于 epoll/kqueue 的全自动 IO 事件发生器。
- swoole_task，基于进程池实现的异步任务处理器。

其中，swoole_event 比 libevent 更简单，仅需 add/set/del 几个操作即可，这样使用者就可以将原有 PHP 代码中的 streams/fsockopen/sockets 代码加入到 swoole 实现异步化，而且利用 swoole_event 还可以实现真正的 PHP 异步 MySQL。

用户可以使用 swoole_task 实现 PHP 的数据库连接池，慢操作异步化[3]，可以说 Swoole 开始将多线程、异步、阻塞引入 PHP 应用开发。

实际上，Swoole 底层内置了异步非阻塞、多线程的网络 IO 服务器，这样仅需处理事件回调[4]即可，无需关心底层。

与 Nginx/Tornado/Node.js 等全异步的框架不同，Swoole 既支持全异步[5]，也支持同步（或者半异步半同步）。

1.1 Overview

最初，PHP 在每个请求进来后都需要重新初始化资源，并且在请求执行完毕后全部丢弃，不过在 CLI 模式下可以不释放资源。

[1] Swoole 本质是 PHP 的一种异步并行扩展，因此要应用 Swoole，首先需要有 PHP 环境。

[2] Swoole 内置了 Socket 客户端的实现，但是采用的是同步 + 并行方式来执行。PHP 本身虽然也提供了 socket 的功能，但是某些函数存在 bug，而且比较复杂，因此使用 Swoole 内置的客户端类更加安全和简化。

[3] Swoole 使用了传统 Linux 下半同步半异步多 Worker 的实现方式，业务代码可以按照更简单的同步方式编写，只有慢操作才考虑使用异步。

[4] 如果在使用 Node.js 等进行开发时的代码太复杂，就会产生嵌套多层回调，使代码丧失可读性，程序流程变得很乱。

[5] Node.js 支持全异步回调，而且内置了异步高性能的 Socket Server/Client 实现，在此基础上提供了内置的 Web 服务器。

1.2 FastCGI

1.3 Multithread

同步和多线程的关系如下：

1. 没有多线程环境就不需要同步。
2. 即使有多线程环境也不一定需要同步。

1.3.1 Block

一旦一个线程处于一个标记为 synchronized 的方法中，那么在这个线程从该方法中返回之前，其他所有要调用类中任何标记为 synchronized 方法的线程都会被阻塞。

每个对象都有一个单一的锁，这个锁本身就是对象的一部分。

当在对象上调用其任意 synchronized 方法的时候，此对象都被加锁，这时该对象上的其他 synchronized 方法只有等到前一个方法调用完毕并释放了锁之后才能被调用。

原子操作不需要进行同步（对除 double 和 long 以外的其他基本类型的进行读取或赋值的操作也是原子操作），然而只要给 long 或 double 加上 volatile，操作就是原子操作了。

只能在同步控制方法或同步块中调用 wait()、notify() 和 notifyAll()。如果在非同步的方法里调用这些方法，在运行时会抛出 java.lang.IllegalMonitorStateException 异常。

- 在调用 wait 的时候，线程自动释放其占有的对象锁，同时不会去申请对象锁。
- 当线程被唤醒的时候，它才再次获得了去获得对象锁的权利。其中：
    1. notify 仅唤醒一个线程并允许它去获得锁；
    2. notifyAll 是唤醒所有等待这个对象的线程并允许它们去获得对象锁。

1.3.2 Dispatch

OOP 的实现相对来说比较复杂，如果低层实现不好，用户的感受只能是难以使用，因此为了保证效率同时避免不必要的运行判定，有以下两个问题一定解决：

- 调度（Dispatch）：相对于静态判定，当调度和重新调度时，在程序执行时动态决定所要调用的子程序。
- 多继承（Multiple Inheritance）：从两个或更多的父类型中继承成员及操作。

1.4 MultiProcess

1.5 Synchronization

1.6 Asynchronization

Chapter 2

Installation

swoole 是标准的 PHP 扩展，而且不依赖 PHP 的 stream、sockets、pcntl、posix、sysvmsg 等扩展，因此只需下载 Swoole 源码包并解压至本地任意目录（保证读写权限），PHP 则只需安装最基本的扩展即可。

按照 PHP 标准扩展构建的构建流程，首先使用 `phpize`[1]来生成 PHP 编译配置，然后执行`./configure` 来做编译配置检测，`make` 进行编译，`make install` 进行安装。

除了手工下载编译外，还可以通过 PHP 官方提供的 pecl 命令来在线安装 swoole：

```
# pecl install swoole
```

2.1 Linux

在安装 PHP 前，需要安装编译环境和 PHP 的相关依赖[2]。

```
$ sudo apt-get install \
  build-essential \
  gcc \
  g++ \
  autoconf \
  libiconv-hook-dev \
  libmcrypt-dev \
  libxml2-dev \
  libmysqlclient-dev \
  libcurl4-openssl-dev \
  libjpeg8-dev \
  libpng12-dev \
  libfreetype6-dev
```

如果在 CentOS/RHEL 环境下编译安装 PHP，则需要预先执行：

1. 编译环境

[1]phpize 命令需要 autoconf 工具，需要预先安装。

[2]swoole 编译为 libswoole.so 作为 C/C++ 库时需要使用 cmake。

```
$ sudo yum -y install \
  gcc \
  gcc-c++ \
  autoconf \
  libjpeg \
  libjpeg-devel \
  libpng \
  libpng-devel \
  freetype \
  freetype-devel \
  libxml2 \
  libxml2-devel \
  zlib \
  zlib-devel \
  glibc \
  glibc-devel \
  glib2 \
  glib2-devel \
  bzip2 \
  bzip2-devel \
  ncurses \
  ncurses-devel \
  curl \
  curl-devel \
  e2fsprogs \
  e2fsprogs-devel \
  krb5 \
  krb5-devel \
  libidn \
  libidn-devel \
  openssl \
  openssl-devel \
  openldap \
  openldap-devel \
  nss_ldap \
  openldap-clients \
  openldap-servers \
  gd \
  gd2 \
  gd-devel \
  gd2-devel \
  perl-CPAN \
  pcre-devel
```

2. 编译 PHP

```
$ cd /usr/local/php-src
$ ./configure \
  --prefix=/usr/local/php \
  --with-config-file-path=/etc/php \
  --enable-fpm \
  --enable-pcntl \
  --enable-mysqlnd \
  --enable-opcache \
  --enable-sockets \
  --enable-sysvmsg \
  --enable-sysvsem \
  --enable-sysvshm \
  --enable-shmop \
  --enable-zip \
  --enable-ftp \
  --enable-soap \
  --enable-xml \
  --enable-mbstring \
  --disable-rpath \
  --disable-debug \
  --disable-fileinfo \
  --with-mysql=mysqlnd \
  --with-mysqli=mysqlnd \
  --with-pdo-mysql=mysqlnd \
  --with-pcre-regex \
  --with-iconv \
  --with-zlib \
  --with-mcrypt \
  --with-gd \
  --with-openssl \
  --with-mhash \
  --with-xmlrpc \
  --with-curl \
  --with-imap-ssl
$ sudo make
$ sudo make install
$ sudo cp php.ini-development /etc/php/
$ cat >> ~/.bashrc
export PATH=/usr/local/php/bin:$PATH
export PATH=/usr/local/php/sbin:$PATH

$ source ~/.bashrc
$ php --version
PHP 5.6.12 (cli) (built: Aug 31 2015 11:09:49)
Copyright (c) 1997-2015 The PHP Group
```

```
Zend Engine v2.6.0, Copyright (c) 1998-2015 Zend Technologies
```

3. 编译 swoole

```
$ cd /usr/local/src/
$ sudo tar zxvf swoole-1.7.19-stable.tar.gz
$ cd swoole-src-swoole-1.7.19-stable
$ sudo phpize
$ sudo ./configure \
  --enable-openssl \
  --enable-swoole-debug \
  --enable-ringbuffer \
  --with-php-config=/usr/local/bin/php-config \
  --enable-async-mysql=/usr/local/mysql \
  --with-swoole \
  --enable-swoole \
  --with-gnu-ld \
  --with-pic
$ sudo make
$ sudo make install
$ sudo echo "extension=swoole.so" >> /usr/local/lib/php.ini
$ php -m
[PHP Modules]
Core
ctype
date
dom
ereg
filter
hash
iconv
json
libxml
mysql
mysqlnd
pcre
PDO
pdo_sqlite
Phar
posix
propro
raphf
Reflection
session
SimpleXML
SPL
```

```
sqlite3
standard
swoole
tokenizer
xml
xmlreader
xmlwriter

[Zend Modules]
```

4. 查看 swoole 信息

```
# php --ri swoole
swoole
swoole support => enabled
Version => 1.7.19
Author => tianfeng.han[email: mikan.tenny@gmail.com]
epoll => enabled
eventfd => enabled
timerfd => enabled
signalfd => enabled
cpu affinity => enabled
spinlock => enabled
rwlock => enabled
sockets => enabled
openssl => enabled
ringbuffer => enabled
Linux Native AIO => enabled
Gcc AIO => enabled
pcre => enabled
zlib => enabled
mutex_timedlock => enabled
pthread_barrier => enabled

Directive => Local Value => Master Value
swoole.aio_thread_num => 2 => 2
swoole.display_errors => On => On
swoole.message_queue_key => 0 => 0
swoole.unixsock_buffer_size => 8388608 => 8388608
```

通过 `php -m`[3]或 `phpinfo()` 来查看是否成功加载了 swoole，如果没有可能是 php.ini 的路径不对，可以使用 `php -i |grep php.ini` 来定位到 php.ini 的绝对路径。

[3]首先查看加载的 php.ini 路径并确认加载了正确的 php.ini，其次可以使用绝对路径指定 swoole 的位置，最后在 php.ini 中打开错误显示来检查是否存在启动时错误。

2.1.1 Error

修改 php.ini，打开错误显示，查看是否存在启动时错误。

```
display_errors => Off => Off
display_startup_errors => Off => Off
html_errors => Off => Off
ignore_repeated_errors => Off => Off
log_errors => On => On
log_errors_max_len => 1024 => 1024
track_errors => Off => Off
xmlrpc_errors => Off => Off
swoole.display_errors => On => On
```

默认情况下，可以在 php.ini 修改错误报告的设置，例如：

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This directive informs PHP of which errors, warnings and notices you would like
; it to take action for. The recommended way of setting values for this
; directive is through the use of the error level constants and bitwise
; operators. The error level constants are below here for convenience as well as
; some common settings and their meanings.
; By default, PHP is set to take action on all errors, notices and warnings EXCEPT
; those related to E_NOTICE and E_STRICT, which together cover best practices and
; recommended coding standards in PHP. For performance reasons, this is the
; recommend error reporting setting. Your production server shouldn't be wasting
; resources complaining about best practices and coding standards. That's what
; development servers and development settings are for.
; Note: The php.ini-development file has this setting as E_ALL. This
; means it pretty much reports everything which is exactly what you want during
; development and early testing.
;
; Error Level Constants:
; E_ALL           - All errors and warnings (includes E_STRICT as of PHP 5.4.0)
; E_ERROR         - fatal run-time errors
; E_RECOVERABLE_ERROR - almost fatal run-time errors
; E_WARNING       - run-time warnings (non-fatal errors)
; E_PARSE         - compile-time parse errors
; E_NOTICE        - run-time notices (these are warnings which often result
;                   from a bug in your code, but it's possible that it was
;                   intentional (e.g., using an uninitialized variable and
;                   relying on the fact it is automatically initialized to an
;                   empty string)
; E_STRICT        - run-time notices, enable to have PHP suggest changes
```

```
;                  to your code which will ensure the best interoperability
;                  and forward compatibility of your code
; E_CORE_ERROR    - fatal errors that occur during PHP's initial startup
; E_CORE_WARNING  - warnings (non-fatal errors) that occur during PHP's
;                  initial startup
; E_COMPILE_ERROR - fatal compile-time errors
; E_COMPILE_WARNING - compile-time warnings (non-fatal errors)
; E_USER_ERROR    - user-generated error message
; E_USER_WARNING  - user-generated warning message
; E_USER_NOTICE   - user-generated notice message
; E_DEPRECATED    - warn about code that will not work in future versions
;                  of PHP
; E_USER_DEPRECATED - user-generated deprecation warnings
;
; Common Values:
;   E_ALL (Show all errors, warnings and notices including coding standards.)
;   E_ALL & ~E_NOTICE (Show all errors, except for notices)
;   E_ALL & ~E_NOTICE & ~E_STRICT (Show all errors, except for notices and coding
    standards warnings.)
;   E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR (Show only errors)
; Default Value: E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED
; Development Value: E_ALL
; Production Value: E_ALL & ~E_DEPRECATED & ~E_STRICT
; http://php.net/error-reporting
error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT

; This directive controls whether or not and where PHP will output errors,
; notices and warnings too. Error output is very useful during development, but
; it could be very dangerous in production environments. Depending on the code
; which is triggering the error, sensitive information could potentially leak
; out of your application such as database usernames and passwords or worse.
; For production environments, we recommend logging errors rather than
; sending them to STDOUT.
; Possible Values:
;   Off = Do not display any errors
;   stderr = Display errors to STDERR (affects only CGI/CLI binaries!)
;   On or stdout = Display errors to STDOUT
; Default Value: On
; Development Value: On
; Production Value: Off
; http://php.net/display-errors
display_errors = Off

; The display of errors which occur during PHP's startup sequence are handled
; separately from display_errors. PHP's default behavior is to suppress those
```

```
; errors from clients. Turning the display of startup errors on can be useful in
; debugging configuration problems. We strongly recommend you
; set this to 'off' for production servers.
; Default Value: Off
; Development Value: On
; Production Value: Off
; http://php.net/display-startup-errors
display_startup_errors = Off

; Besides displaying errors, PHP can also log errors to locations such as a
; server-specific log, STDERR, or a location specified by the error_log
; directive found below. While errors should not be displayed on productions
; servers they should still be monitored and logging is a great way to do that.
; Default Value: Off
; Development Value: On
; Production Value: On
; http://php.net/log-errors
log_errors = On

; Set maximum length of log_errors. In error_log information about the source is
; added. The default is 1024 and 0 allows to not apply any maximum length at all.
; http://php.net/log-errors-max-len
log_errors_max_len = 1024
; Do not log repeated messages. Repeated errors must occur in same file on same
; line unless ignore_repeated_source is set true.
; http://php.net/ignore-repeated-errors
ignore_repeated_errors = Off

; Ignore source of message when ignoring repeated messages. When this setting
; is On you will not log errors with repeated messages from different files or
; source lines.
; http://php.net/ignore-repeated-source
ignore_repeated_source = Off

; If this parameter is set to Off, then memory leaks will not be shown (on
; stdout or in the log). This has only effect in a debug compile, and if
; error reporting includes E_WARNING in the allowed list
; http://php.net/report-memleaks
report_memleaks = On

; This setting is on by default.
;report_zend_debug = 0

; Store the last error/warning message in $php_errormsg (boolean). Setting this
    value
```

```
; to On can assist in debugging and is appropriate for development servers. It
    should
; however be disabled on production servers.
; Default Value: Off
; Development Value: On
; Production Value: Off
; http://php.net/track-errors
track_errors = Off

; Turn off normal error reporting and emit XML-RPC error XML
; http://php.net/xmlrpc-errors
;xmlrpc_errors = 0

; An XML-RPC faultCode
;xmlrpc_error_number = 0

; When PHP displays or logs an error, it has the capability of formatting the
; error message as HTML for easier reading. This directive controls whether
; the error message is formatted as HTML or not.
; Note: This directive is hardcoded to Off for the CLI SAPI
; Default Value: On
; Development Value: On
; Production value: On
; http://php.net/html-errors
html_errors = On
; If html_errors is set to On *and* docref_root is not empty, then PHP
; produces clickable error messages that direct to a page describing the error
; or function causing the error in detail.
; You can download a copy of the PHP manual from http://php.net/docs
; and change docref_root to the base URL of your local copy including the
; leading '/'. You must also specify the file extension being used including
; the dot. PHP's default behavior is to leave these settings empty, in which
; case no links to documentation are generated.
; Note: Never use this feature for production boxes.
; http://php.net/docref-root
; Examples
;docref_root = "/phpmanual/"

; http://php.net/docref-ext
;docref_ext = .html

; String to output before an error message. PHP's default behavior is to leave
; this setting blank.
; http://php.net/error-prepend-string
; Example:
```

```
;error_prepend_string = "<span style='color: #ff0000'>"

; String to output after an error message. PHP's default behavior is to leave
; this setting blank.
; http://php.net/error-append-string
; Example:
;error_append_string = "</span>"

; Log errors to specified file. PHP's default behavior is to leave this value
; empty.
; http://php.net/error-log
; Example:
;error_log = php_errors.log
; Log errors to syslog (Event Log on Windows).
;error_log = syslog

;windows.show_crt_warning
; Default value: 0
; Development value: 0
; Production value: 0
```

2.1.2 Debug

如果编译时使用了`--enable-swoole-debug`，在初始化服务的时候可以 set 是否 debug，这样就可以在开发阶段输出调试数据。

2.1.3 Xdebug

PHP 的 xdebug（或 Zend 的 zend_xdebug[4]）扩展都可能会 swoole 崩溃，运行 swoole 程序时务必去掉 xdebug 扩展。

另外，如果 make 或 make install 无法执行或编译错误，原因可能是 php 版本和编译时使用的 phpize 和 php-config 不对应，需要使用绝对路径来进行编译。使用绝对路径执行 PHP。

```
NOTICE: PHP message: PHP Warning: PHP Startup: swoole: Unable to initialize module
Module compiled with module API=20090626
PHP compiled with module API=20121212
These options need to match
in Unknown on line 0
$ sudo /usr/local/php-5.4.17/bin/phpize \
  ./configure \
  --with-php-config=/usr/local/php-5.4.17/bin/php-config \
  /usr/local/php-5.4.17/bin/php server.php
```

[4]如果出现 `error: too many arguments to function 'zend_exception_error'` 错误，则说明 PHP 版本号低于 5.3.10。

2.1.4 MySQL

如果出现“缺少 mysql 头文件”错误，说明--enable-async-mysql 需要指定路径，需要在编译前执行：

```
# ./configure
  --with-php-config=/usr/local/bin/php-config \
  --enable-sockets \
  --enable-async-mysql=/usr/local/mysql
```

如果出现“缺少 mysqli 头文件”错误，说明没有找到 mysqlclient 的头文件，需要安装 mysqlclient-dev。

```
php_mysqli_structs.h:64:23: fatal error: my_global.h: No such file or directory
```

建议自行编译 PHP，不要使用 Linux 包管理系统自带的 php 版本，或者可以更全面的配置 MySQL 相关信息，如下面示例中所示：

```
# LDFLAGS="-L/usr/local/mysql/lib" \
  CPPFLAGS="-I/usr/local/mysql/include" \
  ./configure \
  --with-php-config=/usr/local/php/bin/php-config \
  --enable-async-mysql=/usr/local/mysql/include \
  --enable-swoole \
  --with-swoole \
  --enable-openssl \
  --enable-ringbuffer \
  --enable-sockets=/usr/local/php/ext/sockets \
  --enable-swoole-debug
```

2.1.5 PCRE

如果出现“缺少 pcre.h 头文件”错误，原因是缺少 pcre，需要安装 libpcre。

```
fatal error: pcre.h: No such file or directory
```

2.2 ARM

swoole 也可以运行在 ARM 平台，在编译 Swoole 时手工修改 Makefile 去掉-O2 编译参数。

2.3 Windows

从 swoole-1.7.7 开始，swoole 增加了对 cygwin 环境的支持，在 Windows 环境下可以直接使用 cygwin + php 来运行 swoole 程序。

- 安装 cygwin，并安装 gcc、make、autoconf、php[5]。

[5]cygwin 模式下需要对 PHP 进行简化，去掉不使用的扩展，避免进程占用内存过大，导致 Fork 操作失败。

- 下载 swoole 源码，在 cygwin-shell 中进行 `phpize/configure/make/make install`。
- 修改 php.ini，加入 swoole.so。

Chapter 3

Configuration

Swoole 和 Yaf 等虽然是标准的基于 Zend API 开发的 PHP C 扩展，实际上与普通的扩展不同。

- 普通的扩展只是提供一个库函数。
- swoole 扩展在运行后会接管 PHP 的控制权，进入事件循环（event-loop）。

当 IO 事件发生后，swoole 会自动回调指定的 PHP 函数。

3.1 Configure options

一般来说，./configure 编译配置的额外参数用于开启某些特性。

3.1.1 --enable-msgqueue

使用消息队列作为 IPC 通信方式，消息队列的好处是 buffer 区域可以很大。

- 1.7.5+ 已经移除了此编译选项，改为由 swoole_server->set 动态设置开启；
- 1.7.9 版本已移除消息队列模式

另外，dispatch_mode=3 时，消息队列天然支持争抢。

使用消息队列作为 IPC 时，worker 进程内将无法使用异步，包括异步 swoole_client，task/finish，swoole_event_add，swoole_timer_add。

3.1.2 --enable-swoole-debug

打开调试日志来允许 swoole 打印各类细节的调试信息，生产环境不要启用。

3.1.3 --enable-sockets

增加对 sockets 资源的支持，依赖 PHP 的 sockets 扩展[1]。

在开启`--enable-sockets`参数后，swoole_event_add 就可以添加 sockets 扩展创建的连接到 swoole 的事件循环中。

[1]PHP 的 sockets 扩展基于流行的 BSD sockets，实现了和 socket 进行通信的底层接口，它可以和客户端一样当做一个 socket 服务器。

3.1.4 --enable-async-mysql

增加异步 MySQL 支持，依赖 mysqli 和 mysqlnd。

- mysqli（MySQL Improved Extension）扩展允许用户使用 MySQL4.1.3 或更新版本中新的高级特性。
- mysqlnd（MySQL Native Driver）扩展是 MySQL Client Library（libmysqlclient）的替代，并且从 PHP5.3 开始进入官方 PHP 源码库。

其中，PHP 的 MySQL 数据库扩展 mysql、mysqli 和 pdo_mysql 以前都需要通过 MySQL Client Library 提供的服务来与 MySQL 服务器通信。

为了兼容 MySQL 提供的客户端-服务器协议（MySQL Client/Server Protocol），MySQL Client Library 被废弃，后续的 PHP 数据库扩展都被编译为使用 MySQL Native Driver 来代替 MySQL Client Library。

3.1.5 --enable-ringbuffer

开启 RingBuffer 内存池（试验性质），主要用于提升性能。

3.1.6 --enable-openssl

启用 SSL 支持。

Chapter 4

Server

4.1 swoole_server

swoole_server 是强大的 TCP/UDP Server 框架，多线程，EventLoop，事件驱动，异步，Worker 进程组，Task 异步任务，毫秒定时器，SSL/TLS 隧道加密。

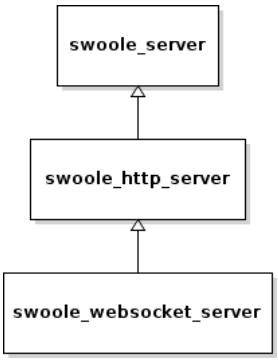

图 4.1: swoole_server 的继承关系

- swoole_http_server 是 swoole_server 的子类，内置了 Http 的支持；
- swoole_websocket_server 是 swoole_http_server 的子类，在 Http 基础上内置了 WebSocket 的支持。

Swoole 的网络 IO 部分基于 epoll/kqueue 事件循环，因此全异步非阻塞执行，而且业务逻辑部分使用多进程同步阻塞方式来运行，从而既保证了 Server 能够应对高并发和大量 TCP 连接，又能够保证业务代码仍然可以简单地编写。

4.1.1 swoole_http_server

4.1.2 swoole_websocket_server

Chapter 5

Client

swoole_client 是 TCP/UDP 客户端，支持同步并发调用，也支持异步事件驱动。

Chapter 6

Event

EventLoop API[1]，让用户可以直接操作底层的事件循环，将 socket，stream，管道等 Linux 文件加入到事件循环中。

6.1 swoole_event

[1]eventloop 接口仅可用于 socket 类型的文件描述符，不能用于磁盘文件读写。

Chapter 7

Async

7.1 swoole_async

Chapter 8

Process

共享内存的性能虽然很好，但是存在安全问题，需要读写时加锁。

- 锁的粒度过大会导致只有一个线程在运行。
- 锁太复杂又会有死锁问题。

8.1 swoole_process

进程管理模块，可以方便的创建子进程，进程间通信，进程管理。

与 Node.js 的网络库本身没有提供多进程/多线程的实现的情况不同，swoole 用户不需要自己手动管理进程的创建与回收，swoole 内核根据配置文件自动完成，Node.j 开发者需要自行创建进程，或者通过 cluster 来利用多核，否则只能使用单线程。

Chapter 9

Buffer

9.1 swoole_buffer

强大的内存区管理工具，像 C 一样进行指针计算，又无需关心内存的申请和释放，而且不用担心内存越界，底层全部做好了。

Chapter 10

Table

10.1 swoole_table

swoole_table 是基于共享内存和自旋锁实现的超高性能内存表[1]，可以彻底解决线程，进程间数据共享，加锁同步等问题。

[1]swoole_table 的性能可以达到单线程每秒读写 50W 次。

Part II

Server

Chapter 11

Overview

首先，软件意义上的服务器就是一个管理资源并为用户提供服务的计算机软件，通常可以分为文件服务器（能使用户在其它计算机访问文件），数据库服务器和应用程序服务器（例如 TCP 服务器、UDP 服务器、HTTP 服务器、WebSocket 服务器以及异步 IO 和 Task 服务器等）。

其次，运行服务器软件的计算机一般称为网络主机（host）[1]，可以通过网络对外提供服务。例如，可以通过 Intranet 对内网提供服务，也可以通过 Internet 对外提供服务。

Web 服务器的定义有时会引起混淆，例如可以指用于网站的计算机，也可能是指 Apache 或 Nginx 等软件，而且运行 Web 服务器软件的计算机可以管理网页组件和回应网页浏览器的请求。

按照服务器软件工作在客户端-服务器还是浏览器-服务器模式，可以有很多形式的服务器，例如：

- 文件服务器（File Server）或网络存储设备（Network Attached Storage），例如 NetWare
- 数据库服务器（Database Server），例如 Oracle，MySQL，PostgreSQL，SQL Server 等
- 邮件服务器（Mail Server），例如 Sendmail、Postfix、Qmail、Microsoft Exchange、Lotus Domino 等
- 网页服务器（Web Server），例如 Apache、httpd 和 IIS 等
- FTP 服务器（FTP Server），例如 Pureftpd、Proftpd、WU-ftpd、Serv-U 等
- 域名服务器（DNS Server），例如 Bind9 等
- 应用程序服务器（Application Server/AP Server），例如 WebLogic、JBoss 和 GlassFish
- 代理服务器（Proxy Server），例如 Squid cache
- NAT 服务器，例如 WINS（Windows Internet Name Service）服务器

11.1 Echo Server

下面是一个基本的基于 swoole 的 echo 服务器实现。

```
<?php
// Server
class Server{
  private $serv;
```

[1]服务器与主机的意义可能不同，其中主机是通过终端给用户使用的，服务器是通过网络给客户端用户使用的。

图 11.1: swoole_server 类

```
public function __construct(){
  $this->serv = new swoole_server("0.0.0.0",9501);
  $this->serv->set(array(
    'worker_num' => 8, // worker进程数
    'daemonize' => 1, // uint32_t, 是否守护进程化
    'max_request' => 10000, // worker进程的最大任务数
    'dispatch_mode' => 2, // 数据包分发策略, 默认为2 (固定模式)
    'debug_mode' => 1 // 无效参数, 可以传入, 不会执行
  ));

  $this->serv->on('Start',array($this,'onStart'));
  $this->serv->on('Connect',array($this,'onConnect'));
  $this->serv->on('Receive',array($this,'onReceive'));
  $this->serv->on('Close',array($this,'onClose'));

  $this->serv->start();
}
// onStart回调在server运行前被调用
public function onStart($serv){
  echo 'Start\n';
}
// onConnect在有新客户端连接过来时被调用
public function onConnect($serv,$fd,$from_id){
  $serv->send( $fd,"Hello {$fd}!" );
```

```
    }
    // onReceive函数在有数据发送到server时被调用
    public function onReceive(swoole_server $serv,$fd,$from_id,$data){
      echo "Get Message From Client {$fd}:{$data}\n";
    }
    // onClose在有客户端断开连接时被调用
    public function onClose($serv, $fd, $from_id){
      echo "Client {$fd} close connection\n";
    }
}

// 启动服务器
$server = new Server();
?>
```

创建一个 swoole_server 基本分为如下三步：

- 通过构造函数创建 swoole_server 对象；
- 调用 set 函数设置 swoole_server 的相关配置选项；
- 调用 on 函数设置相关回调函数。

这里，on 方法的作用是注册 swoole_server 的事件回调函数。

- 在 onStart 处注册 Start 回调函数来启动 swoole_server 实例。
- 在 onConnect 处注册 Connect 回调函数来监听新的连接。
  如果有数据传入，则调用 send 函数将处理结果发送出去。
- 在 onReceive 处注册 Receive 回调函数来接收数据并处理。
- 在 onClose 处注册 Close 回调函数处理客户端下线的事件。

在后台运行 echo server 的代码如下：

```
$ php server.php
$ sudo ps aux | grep server.php | grep -v grep
devops  31368 0.0  0.0 531596 5584 ?     Ssl 22:03  0:00 php server.php
devops  31369 0.0  0.0 235568 4832 ?     S   22:03  0:00 php server.php
devops  31377 0.0  0.0 221788 5012 ?     S   22:03  0:00 php server.php
devops  31378 0.0  0.0 221788 5012 ?     S   22:03  0:00 php server.php
devops  31379 0.0  0.0 221788 5480 ?     S   22:03  0:00 php server.php
devops  31380 0.0  0.0 221788 5012 ?     S   22:03  0:00 php server.php
devops  31381 0.0  0.0 221788 5012 ?     S   22:03  0:00 php server.php
devops  31382 0.0  0.0 221788 5012 ?     S   22:03  0:00 php server.php
devops  31383 0.0  0.0 221788 5012 ?     S   22:03  0:00 php server.php
devops  31384 0.0  0.0 221788 5012 ?     S   22:03  0:00 php server.php
```

11.2 Echo Client

下面使用 swoole_client 创建一个基于 TCP 的客户端实例，并调用 connect 方法向指定的 IP 和端口发起连接请求。

如果连接成功，接下来就可以通过 recv() 和 send() 方法来接收和发送请求。

- 如果使用同步阻塞客户端（默认），recv 和 send 操作都会产生网络阻塞。
- 如果使用异步传输客户端，connect 会立即返回 true，但是实际上连接并未建立。

```
<?php
// Client
class Client{
  private $client;

  public function __construct(){
    // 创建tcp socket
    $this->client = new swoole_client(SWOOLE_SOCK_TCP);
  }

  public function connect(){
    if(!$this->client->connect("127.0.0.1",9501,1)){
      echo "Error: {$fp->errMsg}[{$fp->errCode}]\n";
    }
    $message = $this->client->recv();
    echo "Get Message From Server: {$message}\n";

    fwrite(STDOUT,"请输入消息：  ");
    $msg = trim(fgets(STDIN));
    $this->client->send($msg);
  }
}
$client = new Client();
$client->connect();
?>
```

在使用非阻塞 socket 时，不能在 connect 后使用 send/recv，通过 isConnected() 判断也是 false。只有当连接成功后，系统会自动回调 onConnect，这时才可以使用 send/recv。

在命令行中运行 echo client 的代码如下：

```
$ php client.php
Get Message From Server: Hello 1!
请输入消息：
```

Chapter 12

TCP

12.1 TCP Server

```
$serv = new swoole_server("127.0.0.1", 9501);
$serv->set(array(
    'worker_num' => 8, //工作进程数量
    'daemonize' => true, //是否作为守护进程
));
$serv->on('connect', function ($serv, $fd){
    echo "Client:Connect.\n";
});
$serv->on('receive', function ($serv, $fd, $from_id, $data) {
    $serv->send($fd, 'Swoole: '.$data);
    $serv->close($fd);
});
$serv->on('close', function ($serv, $fd) {
    echo "Client: Close.\n";
});
$serv->start();
```

12.2 TCP Client

```
$client = new swoole_client(SWOOLE_SOCK_TCP, SWOOLE_SOCK_ASYNC);
//设置事件回调函数
$client->on("connect", function($cli) {
    $cli->send("hello world\n");
});
$client->on("receive", function($cli, $data){
    echo "Received: ".$data."\n";
});
$client->on("error", function($cli){
```

```
    echo "Connect failed\n";
});
$client->on("close", function($cli){
    echo "Connection close\n";
});
//发起网络连接
$client->connect('127.0.0.1', 9501, 0.5);
```

Chapter 13

UDP Server

Swoole 支持 CPU Affinity 设置，守护进程化，并且混合 UDP/TCP 多端口监听，多定时器等。

Chapter 14

Web Server

14.1 HTTP Server

```
$serv = new swoole_http_server("127.0.0.1", 9502);

$serv->on('Request', function($request, $response) {
    var_dump($request->get);
    var_dump($request->post);
    var_dump($request->cookie);
    var_dump($request->files);
    var_dump($request->header);
    var_dump($request->server);

    $response->cookie("User", "Swoole");
    $response->header("X-Server", "Swoole");
    $response->end("<h1>Hello Swoole!</h1>");
});

$serv->start();
```

14.2 HTTPS Server

Chapter 15

WebSocket Server

```php
<?php
/**
 * 创建WebSocket Server
 */
$serv = new swoole_websocket_server("127.0.0.1", 9502);

/**
 * 注册Server的事件回调函数open
 */
$serv->on('Open', function($server, $req) {
    echo "connection open: ".$req->fd;
});
/**
 * 注册Server的事件回调函数message
 */
$serv->on('Message', function($server, $frame) {
    echo "message: ".$frame->data;
    $server->push($frame->fd, json_encode(["hello", "world"]));
});
/**
 * 注册Server的事件回调函数close
 */
$serv->on('Close', function($server, $fd) {
    echo "connection close: ".$fd;
});

/**
 * 启动WebSocket Server
 */
$serv->start();
?>
```

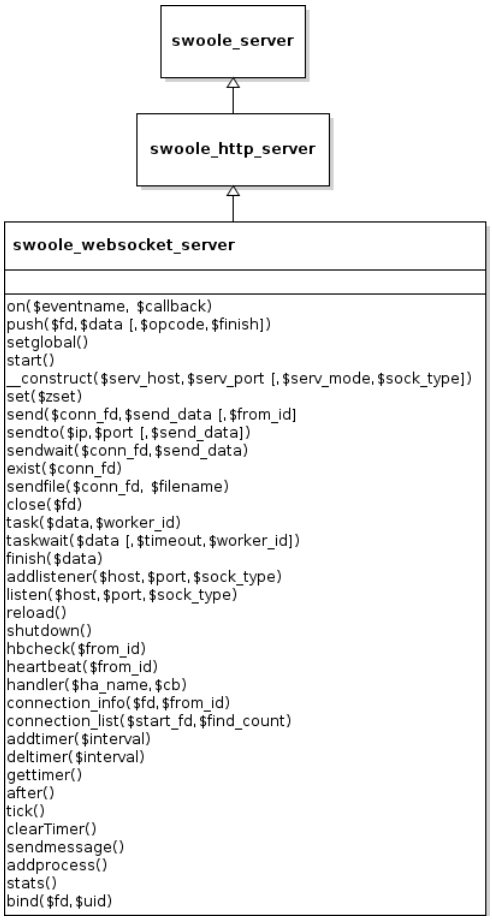

图 15.1: swoole_websocket_server 类

- `$fd`：TCP 连接的文件描述符（file description），在 swoole_server 中是客户端的唯一标识符。
- `$from_id`：来自于哪个 reactor 线程。
- `$conn_fd`：网络字节序（long 类型字段，IPv4 的第 4 字节最小为 1）。

15.1 WS Server

```php
<?php
/**
 * 创建WebSocket Server
 */
$wsserver = new swoole_websocket_server("127.0.0.1", 9502);

/**
 * 注册Server的事件回调函数open
 */
$wsserver->on('Open', function($server, $req) {
    echo "connection open: ".$req->fd;
});

/**
 * 注册Server的事件回调函数message
 */
$wsserver->on('Message', function($server, $frame) {
    echo "message: ".$frame->data;
    $server->push($frame->fd, json_encode(["hello", "world"]));
});

/**
 * 注册Server的事件回调函数close
 */
$wsserver->on('Close', function($server, $fd) {
    echo "connection close: ".$fd;
});

/**
 * 启动WebSocket Server
 */
$wsserver->start();
?>
```

15.2 WSS Server

```
<?php
$wssserver = new swoole_websocket_server("0.0.0.0",9527,SWOOLE_PROCESS,);
?>
```

Chapter 16

WebSocket Client

Chapter 17

Async-IO

```
$fp = stream_socket_client("tcp://127.0.0.1:80", $code, $msg, 3);
$http_request = "GET /index.html HTTP/1.1\r\n\r\n";
fwrite($fp, $http_request);
swoole_event_add($fp, function($fp){
    echo fread($fp, 8192);
    swoole_event_del($fp);
    fclose($fp);
});
swoole_timer_after(2000, function() {
    echo "2000ms timeout\n";
});
swoole_timer_tick(1000, function() {
    echo "1000ms interval\n";
});
```

Chapter 18

Task

```
$serv = new swoole_server("127.0.0.1", 9502);
$serv->set(array('task_worker_num' => 4));
$serv->on('Receive', function($serv, $fd, $from_id, $data) {
    $task_id = $serv->task("Async");
    echo "Dispath AsyncTask: id=$task_id\n";
});
$serv->on('Task', function ($serv, $task_id, $from_id, $data) {
    echo "New AsyncTask[id=$task_id]".PHP_EOL;
    $serv->finish("$data -> OK");
});
$serv->on('Finish', function ($serv, $task_id, $data) {
    echo "AsyncTask[$task_id] Finish: $data".PHP_EOL;
});
$serv->start();
```