Swoole 4.0 协程

全新的PHP编程模式

@hantianfeng Rango-韩天峰

分享内容

一．Go + Chan 全新协程编程模式

二．Swoole 4.0 新版本协程实现原理

三．Swoft 协程框架的使用

01 Swoole 4.0

```php
function fun1() {
  sleep(1);
  echo "hello";
}

function fun2() {
  sleep(1);
  echo "swoole";
}


fun1();
fun2();
echo "done.";
```

- 串行编程
- fun1 要等待 fun2 执行完毕
- 总耗时 2 秒

- 如何实现并发编程?

	多进程	多线程
创建	fork	pthread_create
回收	wait	pthread_join
通信方式	IPC 进程间通信	数据同步/锁
资源消耗	进程切换开销	进程切换开销
并发能力	数百	数千
编程难度	困难	非常困难

Coroutine

	多进程	多线程	协程
创建	fork	pthread_create	go
回收	wait	pthread_join	-
通信方式	IPC 进程间通信	数据同步/锁	array/chan
资源消耗	进程切换开销	进程切换开销	非常低
并发能力	数百	数千	50万
编程难度	困难	非常困难	容易

```php
go(function () {
    co::sleep(1);
    echo "hello";
});

go(function () {
    co::sleep(1);
    echo "swoole";
});
```

```php
go('fun1');
go('fun2');
go([$this, 'fun']);
```

- 并发编程
- fun1 和 fun2 并发执行
- 总耗时 1 秒

```php
<?php
$socket = stream_socket_server("tcp://0.0.0.0:8000",
    $errno, $errstr);
while ($conn = stream_socket_accept($socket)) {
    if (pcntl_fork() == 0) {
        $request = fread($conn);
        fwrite($conn, "hello world\n");
        fclose($conn);
        exit;
    }
}
```

```php
$socket = new Co\Socket(AF_INET, SOCK_STREAM, 0);
$socket->bind('127.0.0.1', 8000);
$socket->listen(128);

go(function () use ($socket) {
    while(true) {
        $client = $socket->accept();
        go(function () use ($client) {
                $data = $client->recv();
                $client->send("Server: $data");
            }
        });
    }
});
```

创建 Socket
绑定端口并监听

Accept 连接
创建新的协程处理

接收数据并响应
协程退出

协程

1. 用户态线程，遇到 IO 主动让出
2. PHP 代码依然是串行执行的，无需加锁
3. 开销极低，仅占用内存，不存在进程/线程切换开销
4. 并发量大，单个进程可开启 50W 个协程
5. 随时随地，只要你想并发，就调用 go 创建新协程

```php
for($i = 0; $i < 10; $i++) {
    go(function () {
        co::sleep(1000);
    });
}
```

```php
$chan = new chan;

go(function() use ($chan) {
    $retval = [1, 2, 3, 4, 5];
    $chan->push($retval);
});


go(function() use ($chan) {
    $retval = "hello world";
    $chan->push($retval);
});

go(function () use ($chan) {
    for($i = 0; $i < 2; $i++) {
        $chan->pop();
    }
    echo "done.\n";
});
```

SplQueue	Chan
new SplQueue	new chan()
-	缓存/无缓存
$queue->push	$chan->push
$queue->pop	$chan->pop
push 永远可用，持续写内存	push 容量不足是挂起协程
pop 无可用数据时返回 false	pop 无可用数据时挂起协程

通道

1. 数据流转
2. 协程管理
3. 并发依赖管理
4. 多个 chan 可以使用 chan::select 进行读写判断

```php
$server = new Swoole\Http\Server('127.0.0.1', 8000);

$server->on('Request', function ($req, $resp) {
    $chan = new chan();

    go(function () use ($chan) {
        $c = new Co\Http\Client('www.baidu.com', 443, true);
        $c->get('/index.php');
        $chan->push($c->body);
    });

    go(function () use ($chan) {
        $c = new Co\Http\Client('www.taobao.com', 443, true);
        $c->get('/index.php');
        $chan->push($c->body);
    })

    for($i = 0; $i < 2; $i++) {
        $resp->write($chan->pop());
    }
    $resp->end();
});

$server->start();
```

Http 请求

Http 请求

发送响应

协程组件	说明	同步阻塞 API
Co\Socket	Socket 的封装	Sockets/Stream
Co\Client	TCP/UDP/UnixSocket客户端	Sockets/Stream
Co\Http\Client	Http和WebSocket客户端	CURL/file_get_contents
Co\Http2\Client	Http2客户端	CURL/GRPC
Co\MySQL	MySQL客户端	mysqli/PDO
Co\Redis	Redis客户端	redis
Co::sleep	睡眠	usleep/sleep
Co::readFile/Writefile	读写文件	fread/fwrite

02 协程实现

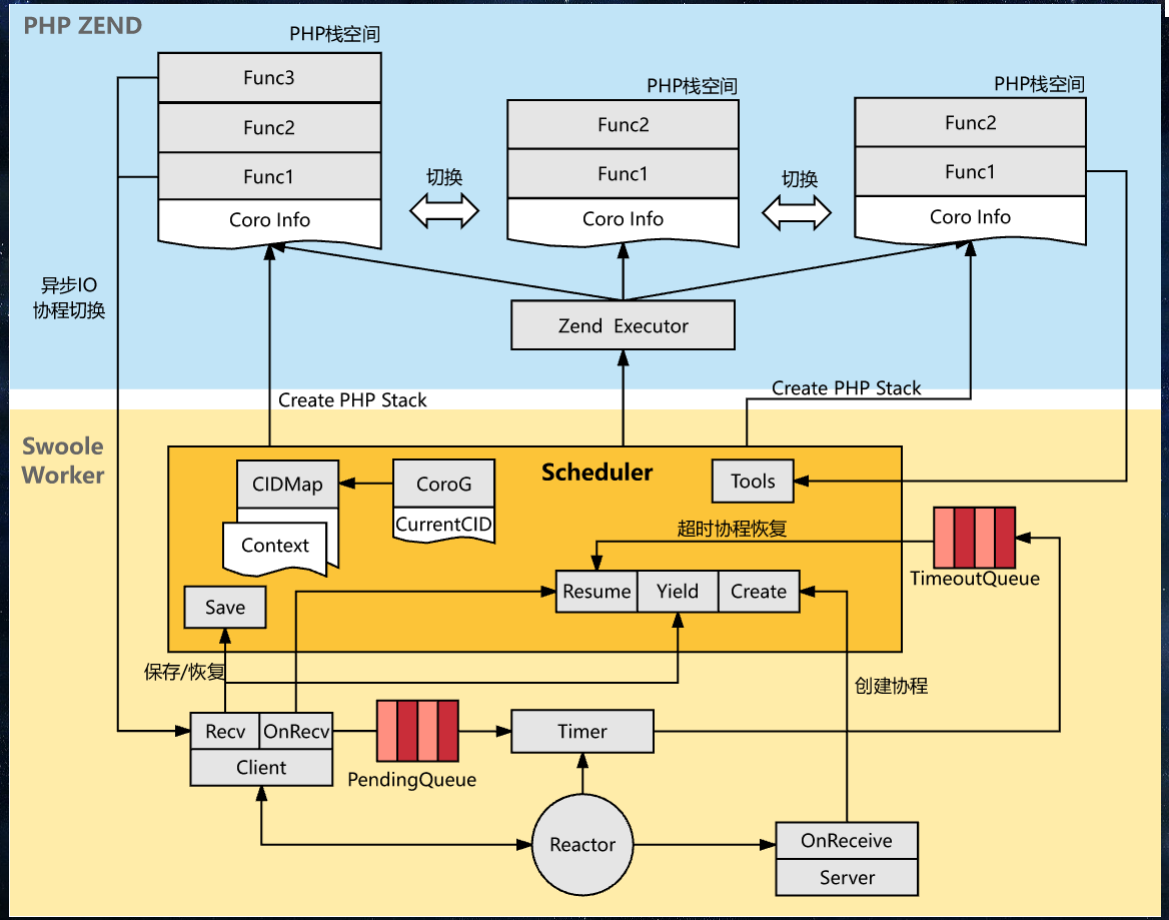
- 2.0 setjmp/longjmp
- 3.0 EG(vm_interrupt)
- 4.0 libco

Swoole 2.0 和 3.0 的局限性

1. 不支持 call_user_func, array_map, 魔术方法，反射方法
2. 不支持扩展 zend_call_function
3. 很多第三方库，使用了复杂的设计模式，不可控

Swoole 4.0

1. 100% 支持所有 PHP 语法，包括魔术方法、反射、call_func
2. 完整的 C 栈（基于微信开源的 libco）+ PHP 栈
3. 前置 SysCall Hook，可将 PHP 的同步客户端，如 mysqli/pdo/file_get_contents/sleep/fread 无缝切换为协程模式（Swoole 5.0）



swoole_coroutine.cc

- co_create
- co_yield
- co_resume
- co_close

```
EG(vm_stack) = current_coro->origin_coro->stack;
EG(vm_stack_top) = current_coro->origin_coro->vm_stack_top;
EG(vm_stack_end) = current_coro->origin_coro->vm_stack_end;
```

```
static int libco_yield()
{
    co_yield_ct();
    return 0;
}
```

```
static int libco_resume(stCoRoutine_t *co)
{
    co_resume(co);
    if (co->cEnd) {
        resume_php_stack(co);
        libco_release(co);
    }
    return 0;
}
```

03 Swoft 框架

Swoft

1. 完全基于 Swoole 的纯协程框架
2. Composer 组件化，完全遵循 PSR 规范
3. 依赖注入，容器，组件，连接池，AOP（面向切面编程）
4. 支持 Web 开发、微服务治理
5. Docker 支持

```
composer create-project swoft/swoft swoft
```

```
composer require swoft/db
```

Swoft 支持的服务器

1. swoft-http-srever ：高并发纯协程 Web 应用程序
2. swoft-websocket-server ：长连接通信服务器
3. swoft-rpc-server ：微服务治理

服务启动

此服务启动指的是单独的RPC服务启动，因为HTTP Server启动伴随着RPC服务启动方式，是不需要手动启动。

```
[root@0dd3950e175b swoft]# php bin/swoft rpc:start
                    Information Panel
*********************************************************
* tcp | Host: 0.0.0.0, port: 8099, Model: 3, type: 1
*********************************************************
```

- php bin/swoft rpc:start , 启动服务，根据 .env 配置决定是否是守护进程
- php bin/swoft rpc:start -d , 守护进程启动，覆盖 .env 守护进程(DAEMONIZE)的配置
- php bin/swoft rpc:restart , 重启
- php bin/swoft rpc:reload , 重新加载
- php bin/swoft rpc:stop , 关闭服务

快速创建控制器

```
// Gen DemoController class to `@app/Controllers`
php bin/swoft gen:controller demo --prefix /demo -y

// Gen UserController class to `@app/Controllers`(RESTFul type)
php bin/swoft gen:controller user --prefix /users --rest
```

```php
/**
 * action demo
 *
 * @Controller(prefix="/route")
 */
class RouteController
{
    /**
     * @RequestMapping()
     */
    public function index()
    {
        return 'index';
    }

    /**
     * @RequestMapping(route="user/{uid}/book/{bid}/{bool}/{name}")
     *
     * @param bool    $bool
     * @param Request  $request
     * @param int      $bid
     * @param string   $name
     * @param int      $uid
     * @param Response $response
     *
     * @return array
     */
    public function funcArgs(bool $bool, Request $request, int $bid, string $name, int $uid
sponse $response)
    {
        return [$bid, $uid, $bool, $name, \get_class($request), \get_class($response)];
    }
    ...
}
```

控制器与 URL 映射

URL 路由

GET 参数映射

Swoft 微服务

1. 服务熔断
2. 服务降级
3. 服务注册与发现（基于 consul）
4. 负载均衡