

基于 **Swift** 协程框架实现 PHP 微服务治理

@hantianfeng Rango-韩天峰

关于我

- **Swoole、PHP-X 开源项目创始人**
- **车轮互联总架构师**
- **微博：@hantianfeng**
- **Github： <https://github.com/matyhtf>**

分享内容

- 一. 基于 Swoole 4.0 全新的 PHP 编程模式
- 二. 协程框架 Swoft 的介绍
- 三. 基于 Swoft 协程框架进行 PHP 微服务治理



01

Swoole 4.0


```
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

The background of the slide is a deep space image. It features a prominent blue-tinted nebula with wispy, cloud-like structures. Scattered throughout the dark space are numerous stars of varying brightness. Some stars show diffraction spikes, giving them a starburst appearance. The overall color palette is dominated by dark blues, blacks, and a few hints of lighter blue and white from the stars and nebula.

	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


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

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

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

- 并发编程
- fun1 和 fun2 并发执行
- 总耗时 1 秒


```
1 <?php
2 $socket = stream_socket_server("tcp://0.0.0.0:8000",
3     $errno, $errstr);
4 while ($conn = stream_socket_accept($socket)) {
5     if (pcntl_fork() == 0) {
6         $request = fread($conn);
7         fwrite($conn, "hello world\n");
8         fclose($conn);
9         exit;
10    }
11 }
```



```
$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 Go

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


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

```
go(function () {  
    $res = co::readFile('/tmp/data');  
    go(function () use ($res) {  
        co::writeFile('/tmp/data2');  
        go(function () use ($res) {  
            var_dump($res);  
        });  
    });  
});
```

```
$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";  
});
```



```
$data = array();

go(function() use (&$array) {
    $array['hello'] = 'swoole';
});

go(function() use (&$array) {
    echo $array['hello'];
    $array['hello'] = 'world';
});
```

```
$lock = new Mutex;

pthread_create(function() use (&$array, $lock) {
    $lock->lock();
    $array['hello'] = 'swoole';
    $lock->unlock();
});

pthread_create(function() use (&$array, $lock) {
    $lock->lock();
    echo $array['hello'];
    $array['hello'] = 'world';
    $lock->unlock();
});
```


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 Chan

1. 数据流转，协程管理，并发依赖管理
2. 仅占用内存，无锁，不存在数据同步问题
3. 可以 push 任意 PHP 变量
4. 基于引用计数管理，零拷贝
5. 多个 chan 可以使用 `chan::select` 进行读写判断


```
$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

Swift 框架

Swift

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 swift/swift swift
```

```
composer require swift/db
```


Swift 支持的服务器

1. swift-http-srever : 高并发**纯协程** Web 应用程序
2. swift-websocket-server : 长连接通信服务器
3. swift-rpc-server : RPC 服务

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 swift]# php bin/swift rpc:start
Information Panel
*****
* tcp | Host: 0.0.0.0, port: 8099, Model: 3, type: 1
*****
```

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

快速创建控制器

```
// Gen DemoController class to `@app/Controllers`  
php bin/swift gen:controller demo --prefix /demo -y  
  
// Gen UserController class to `@app/Controllers` (RESTful type)  
php bin/swift gen:controller user --prefix /users --rest
```



```

/**
 * 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, Response $response)
    {
        return [$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



03

微服务治理

Swift 微服务

1. 服务注册与发现 (基于 **consul**)
2. 接口多版本
3. 服务熔断
4. 服务降级
5. 负载均衡

配置文件: `./config/beans/base.php`

```
return [  
    // .....  
    'providerSelector' => [  
        'class' => \Swift\Sg\ProviderSelector::class,  
        'provider' => 'consul',  
        'providers' => [  
            'consul' => \Swift\Sg\Provider\ConsulProvider::class  
        ]  
    ],  
    // .....  
];
```

- provider 配置服务提供方, 默认consul
- providers 配置自定义服务, 通过KEY名称使用


```
/**
 * @Reference("user")
 *
 * @var DemoInterface
 */
private $demoService;

/**
 * @Reference(name="user", version="1.0.1")
 *
 * @var DemoInterface
 */
private $demoServiceV2;
```

```
/**
 * @RequestMapping(route="call")
 * @return array
 */
public function call()
{
    $version = $this->demoService->getUser('11');
    $version2 = $this->demoServiceV2->getUser('11');

    return [
        'version' => $version,
        'version2' => $version2,
    ];
}
```



```

/**
 * the breaker of user
 *
 * @Breaker("user")
 */
class UserBreaker extends CircuitBreaker
{
    /**
     * The number of successive failures
     * If the arrival, the state switch to open
     *
     * @Value(name="${config.breaker.user.failCount}", env="${USER_BREAKER_FAIL_COUNT}")
     * @var int
     */
    protected $switchToFailCount = 3;

    /**
     * The number of successive successes
     * If the arrival, the state switch to close
     *
     * @Value(name="${config.breaker.user.successCount}", env="${USER_BREAKER_SUCCESS_COUNT}")
     * @var int
     */
    protected $switchToSuccessCount = 3;

    /**
     * Switch close to open delay time
     * The unit is milliseconds
     *
     * @Value(name="${config.breaker.user.delayTime}", env="${USER_BREAKER_DELAY_TIME}")
     * @var int
     */
    protected $delaySwitchTimer = 500;
}

```

失败超过次数，断开

成功超过次数，连接

N 毫秒重试


```
\breaker('name')->call($handler, $params, $fallback);
```



```
/**
 * @Reference(name="user", fallback="demoFallback")
 *
 * @var DemoInterface
 */

/**
 * @return array
 */
public function fallback()
{
    $result1 = $this->demoService->getUser('11');
    $result2 = $this->demoService->getUsers(['1','2']);
    $result3 = $this->demoService->getUserByCond(1, 2, 'boy', 1.6);

    return [
        $result1,
        $result2,
        $result3,
    ];
}
```


Swift + 腾讯 Tars

- 多语言：C++、Java、Node.js、Go、PHP
- IDL：基于 WUP 结构定义文件，自动生成接口骨架代码
- 完整成熟的服务治理方案
- 自带发布、运维、监控、弹性伸缩体系



THANK YOU

- 期待再见 -

Q & A