代码解耦之道

PHPCON www.phpconchina.com

黄朝晖

PHPCon 历年完整 PPT 下载站：

https://github.com/ThinkDevelopers/PHPConChina

Hyperf

- 启动于 2018 年 8月，于 2019 年 6 月 20 日以 MIT 协议对外公开
- 每周发布一个兼容的迭代版本，每个变更都有 Changelog
- Github 获得 842 个 star，共 2,517 commits，256 Pull Requests，22 Contributors
- 60 个组件，560 tests，1,870 assertions

性能数据

- ab -k -c 100 -n 10000
- ≈90883 requests / s
- time per request
  - 1.100 ms (C)
  - 0.011 ms (S)
- % of the requests
  - 80% < 1 ms
  - 99% < 3 ms

```
Server Software:        Hyperf
Server Hostname:        127.0.0.1
Server Port:            9501

Document Path:          /
Document Length:        1 bytes

Concurrency Level:      100
Time taken for tests:   0.110 seconds
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    10000
Total transferred:      1420000 bytes
HTML transferred:       10000 bytes
Requests per second:    90883.48 [#/sec] (mean)
Time per request:       1.100 [ms] (mean)
Time per request:       0.011 [ms] (mean, across all concurrent requests)
Transfer rate:          12602.98 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       2
Processing:     0    1   0.4      1       3
Waiting:        0    1   0.4      1       3
Total:          0    1   0.4      1       4

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      1
  75%      1
  80%      1
  90%      2
  95%      2
  98%      2
  99%      3
 100%      4 (longest request)
➜  hyperf-course
```

初衷

- 尽管现在基于 PHP 语言开发的框架处于一个百花争鸣的时代，但仍旧未能看到一个优雅的设计与超高性能的共存的完美框架，亦没有看到一个真正为 PHP 微服务铺路的框架，此为 Hyperf 及其团队成员的初衷，我们将持续投入并为此付出努力，也欢迎大家加入我们参与开源建设
- 各种 Swoole 框架开发无法沉淀和复用，设计上强耦合，而 Hyperf 每个组件从设计之初就考虑如何在其它框架上复用

Hyperf

- MySQL Client、Redis Client，Eloquent ORM、JSON RPC Server/Client、gRPC Server/Client、WebSocket Server/Client、Zipkin (OpenTracing) Client、Guzzle HTTP、Elasticsearch Client、Consul Client、ETCD Client、AMQP、Redis Queue、Apollo / 阿里云 ACM / ETCD 配置中心、基于令牌桶算法的限流器、通用连接池、熔断器、Swagger、Swoole Tracker (Enterprise)、Blade / Smarty View Engine
- Hyperf 还提供了 依赖注入、注解、AOP 面向切面编程、中间件、自定义进程、事件管理器、自动模型缓存、Crontab 定时任务

Hyperf 的优势

- 高性能、CSP 协程编程、DI、注解、AOP、极其开放
- Laravel 用户无缝转移，无需重新学习的 Eloquent ORM
- 协程组件库丰富且完备，不重新发明轮子，更稳定
- 可单体、可微服务、可 API、可视图、可注解，可配置
- 全面实现标准的 PSR 协议，3 (Logger), 7 (HTTP Message), 11 (Container), 14 (Event Dispatcher), 15 (HTTP Handlers), 16 (Simple Cache)
- 非常详细的文档，变更记录清晰
- 代码风格统一、Pull Request 及 提交记录规范清晰

何为耦合?

- 指一个程序中，模块与模块之间信息或参数依赖的程度

耦合度的计算方法

di：输入数据参数的个数
ci：输入控制参数的个数
do：输出数据参数的个数
co：输出控制参数的个数

gd：用来存储数据的全局变量
gc：用来控制的全局变量
w：此模块调用的模块个数
r：调用此模块的模块个数

$$\text{Coupling(C)} = 1 - \frac{1}{d_i + 2 \times c_i + d_o + 2 \times c_o + g_d + 2 \times g_c + r + w}$$

Coupling(C) 数值越大表示耦合越严重，
数值一般会界于0.67（低度耦合）到1.0（高度耦合）之间

耦合度的计算方法

di：输入数据参数的个数
ci：输入控制参数的个数
do：输出数据参数的个数
co：输出控制参数的个数

数据和控制流的耦合 ⇒ 对参数解耦

gd：用来存储数据的全局变量
gc：用来控制的全局变量

全局耦合 ⇒ 对内容解耦

w：此模块调用的模块个数
r：调用此模块的模块个数

环境耦合 ⇒ 对依赖解耦

为什么要解耦合?

- 解耦某种程度上是防御性编程，防御产品经理的脑洞
- 程序员最大的敌人不是需求，而是需求变更
- 解耦合是为了更好的应对需求变更

如何解耦?

- **单一原则**
- **依赖倒置原则**
- **接口隔离原则**
- 里氏替换原则
- 开闭原则
- 最少原则（迪米特法则）

单一原则

- 一个类只负责一项职责

```
class Human
{
    public function 教书();
    public function 讲课();
    public function 听课();
    public function 写作业();
}
```

```
class Teacher
{
    public function 教书();
    public function 讲课();
}
```

```
class Student
{
    public function 听课();
    public function 写作业();
}
```

依赖倒置

- 高层模块不应该依赖低层模块，二者都应该依赖其抽象
- 抽象不依赖细节，细节依赖抽象

依赖倒置

```php
class Teacher
{
    public function teach()
    {
        $english = new English();
        return $english->teach();
    }
}

class English
{
    public function teach()
    {
        return 'Hello everyone, I am English teacher.';
    }
}
```

依赖倒置

```php
class Teacher
{
    public function teach()
    {
        $chinese = new Chinese();
        return $chinese->teach();
    }
}

class Chinese
{
    public function teach()
    {
        return '同学们好，我是语文老师';
    }
}
```

依赖倒置

简单来讲，就是

面向接口编程

```php
class Teacher
{
    public function teach(CourseInterface $course)
    {
        return $course->teach();
    }
}

interface CourseInterface
{
    public function teach();
}

class English implements CourseInterface
{
    public function teach()
    {
        return 'Hello everyone, I am English teacher.';
    }
}

$teacher = new Teacher();
$teacher->teach(new English());
```

接口隔离

- 实现不应该依赖它不需要的接口，一个类对另一个类的依赖应该建立在最小的接口上

```
class Human implements Teach, Study
{
    public function 教书();
    public function 讲课();
    public function 听课();
    public function 写作业();
}
```

```
interface Teach
{
    public function 教书();
    public function 讲课();
}
```

```
interface Study
{
    public function 听课();
    public function 写作业();
}
```

策略模式

- 指对象有某个行为，但在不同的场景中，该行为有不同的实现算法

```php
class Teacher
{
    private $course;
    public function __construct(CourseInterface $course)
    {
        $this->course = $course;
    }
    public function teach()
    {
        return $this->course->teach();
    }
}

interface CourseInterface
{
    public function teach();
}

class English implements CourseInterface
{
    public function teach()
    {
        return 'Hello everyone, I am English teacher.';
    }
}

$teacher = new Teacher(new English());
$teacher->teach();
```

IoC 与 DI

- IoC，即控制反转，把对象的调用权交给容器，通过容器来实现对象的装配和管理
- DI，即依赖注入，对象之间依赖关系由容器在运行期决定，由容器动态的将依赖关系注入到对象之中
- DI 是对 IoC 更完善的描述

IoC 与 DI

- IoC，即控制反转，把对象的调用权交给容器，通过容器来实现对象的装配和管理
- 谁控制谁？ IoC 容器控制对象
- 控制了什么？控制了对象的外部对象获取
- 为什么要反转？IoC 容器帮助我们获取对象及注入依赖
- 反转了什么？获取依赖对象的过程被反转了

IoC 与 DI

- DI，即依赖注入，对象之间依赖关系由容器在运行期决定，由容器动态的将依赖关系注入到对象之中
- 谁依赖谁？对象实例化依赖容器
- 为什么要依赖？对象实例化通过容器自动得到外部依赖
- 谁注入谁？容器注入对象的依赖到对象中
- 注入了什么？注入了对象的外部依赖

IoC 与 DI

```php
class Teacher implements TeacherInterface
{
    private $course;
    public function __construct(CourseInterface $course)
    {
        $this->course = $course;
    }
    public function teach()
    {
        return $course->teach();
    }
}

interface TeacherInterface {}

interface EnglishTeacher extends TeacherInterface {}

$this->app->bind(EnglishTeacher::class, function ($app) {
    return new Teacher(new English());
});
$teacher = $this->app->make(EnglishTeacher::class);
$teacher->teach();
```

IoC 与 DI – 自动注入

```php
class TeachController
{
    private $teacher;
    public function __construct(EnglishTeacher $teacher)
    {
        $this->teacher = $teacher;
    }
    public function teach()
    {
        $this->teacher->teach();
    }
}
```

观察者模式

- 一种对象行为模式
- 它定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都能得到通知
- PSR-14 Event Dispatcher 事件调度器

观察者模式

```php
class Teacher
{
    private $eventDispatcher;
    public function __construct(EventDispatcher $eventDispatcher)
    {
        $this->eventDispatcher = $eventDispatcher;
    }
    public function teach(CourseInterface $course)
    {
        $this->eventDispatcher->dispatch(new BeforeTeach($course->isPublic));
        return $course->teach();
    }
}
```

观察者模式

- 一个简单的基于 PSR-14 实现的事件调度器

```php
class ListnerProvider implements ListenerProviderInterface
{
    private $listeners = [];
    public function getListenersForEvent(object $event): iterable
    {
        $listeners = [];
        foreach ($this->listeners as $listenerData) {
            if ($event instanceof $listenerData[0]) {
                $listeners[] = $listenerData[1];
            }
        }
        return $listeners;
    }
    public function on(object $event, $listener): void
    {
        $this->listeners[] = [$event, $listener];
    }
}
```

观察者模式

```php
class EventDispatcher implements EventDispatcherInterface
{
    private $listenerProvider;
    public function __construct(ListenerProviderInterface $listenerProvider)
    {
        $this->listenerProvider = $listenerProvider;
    }
    public function dispatch(object $event)
    {
        foreach ($this->listenerProvider->getListenersForEvent($event) as $listener) {
            $listener($event);
            if ($event instanceof StoppableEventInterface
                && $event->isPropagationStopped()
               ) {
                break;
            }
        }
        return $event;
    }
}
```

观察者模式

```php
class BeforeTeach
{
    public $isPublic;
    public function __constuct($public = false)
    {
        $this->isPublic = $public;
    }
}

class PublicListener
{
    public function process(BeforeTeach $event)
    {
        if ($event->isPublic === true) {
            $this->openCamera();
        }
    }
}

$listenerProvider = new ListenerProvider();
$listenerProvider->on(new BeforeTeach(), [new PublicListener(), 'process']);
$dispatcher->dispatch(new BeforeTeach(true));
```

AOP

- 面向切面编程，通过预编译方式或运行期动态代理实现程序功能的统一维护的一种技术
- AOP 在字面上与 OOP 很相似，但设计思想在目标上有着本质的差异
- OOP 是针对业务处理过程的实体及其属性和行为进行抽象封装，以获得更加清晰高效的逻辑单元划分
- 而 AOP 则是针对业务处理过程中的切面进行提取，它所面对的是处理过程中的某个步骤或阶段，以获得逻辑过程中各部分之间低耦合性的隔离效果

AOP

- 应用场景：参数校验、日志、无侵入埋点、安全控制、性能统计、事务处理、异常处理、缓存、监控、资源池、连接池管理等

- PHP 下的实现：
  - Hyperf Swoft ESD 框架（CLI）
  - AOP扩展（FPM）- XAOP https://gitee.com/josinli/xaop

AOP – Hyperf

```php
class Teacher
{
    private $course;
    public function __construct(CourseInterface $course)
    {
        $this->course = $course;
    }
    public function teach()
    {
        return $this->course->teach();
    }
}
```

AOP – Hyperf

```php
/**
 * @Aspect
 */
class PublicClassAspect extends AbstractAspect
{
    public $classes = [
        Teacher::class,
        // 'Teacher::teach',
        // 'Teacher::*',
    ];
    public function process(ProceedingJoinPoint $proceedingJoinPoint)
    {
        $this->openCamera();
        $result = $proceedingJoinPoint->process();
        $this->closeCamera();
        return $result;
    }
}
```

PHPCON 官方渠道：

官网：http://www.phpconchina.com

公众号：PHPCon

纪念品购买渠道： https://k.weidian.com/H3=4lVho

微信交流群：

添加个人微信号「PHPConChina」自动通过后，输入加群密码：11643 稍等自动拉群

感谢聆听