

限时福利：打包购买优惠多多，欲购从速哦~ 查看详情 (/buy)

php之依赖注入、控制反转和依赖倒置原则

更新于 2017年09月06日 by 白狼 被浏览了 2355 次

☆ 2 ♡ 10

今天我们继续补充php的相关内容。

开篇我们先做个声明：此篇文章没有难度，也是我们为后面所补充的基础，不要被标题给吓到了。

以下是正文内容：

思考一个问题：假如公司让你评价各开发成员的代码质量，好坏如何区分？

跟你玩的好的最好还是跟你有过过节的最差呢？这都不好说。

要评判好坏，我们是不是需要有一种标准？

相信不少同学都听说过“高内聚，低耦合”，即类的内聚性是不是很高，耦合度是不是很低。这一原则可以作为我们评判软件设计的好坏。

通俗的说，我们就需要尽量让写出的程序易于维护，减少程序与程序之间的复杂性、耦合度。

在php中，为了解决此类问题，有很多优秀的设计模式，比如单例模式、工厂模式等。

提到设计模式，有些同学头就懵了。很多时候，你可能不经意间就已经完成了某模式的实现，尽管你头脑中并没有此类概念。这都无所谓。

但是，在yii2的核心源码实现上，却有很多优秀的设计模式的体现。比如我们接下来要介绍的控制反转。

所谓控制反转，英文 Inversion of Control，简称 IOC，字面上可以理解为对xx的控制进行了一个反转，换句话说就是对xx的控制的另一种实现，是一种思路，一种逻辑思想。这个很容易理解。

既然是一种思路，那总有一种解决方案可以实现吧。这就是我们接下来要介绍的依赖注入。

依赖注入，英文 Dependency Injection，简称 DI，是 IOC 的一种典型实现。

那什么是依赖注入呢？

简单的说，依赖注入就是把对象A所依赖的其他对象B呀C呀的，以属性或者构造函数的方式传递到对象A的内部，而不是直接在对象A内实例化。

其目的就是为了让对象A和其依赖的其他对象解耦，减少二者的依赖。即通过“注入”的方式去解决依赖问题。

听不懂？没关系，我们来看一个用户注册成功后发送邮件的例子，撇开依赖注入，通常我们可以这样实现

```
class EmailSenderByQq
{
    public function send()
    {
    }
}

class User
{
    public function register()
    {
        // other code

        $email = new EmailSenderByQq;
        $email->send();
    }
}
```

调用User类的register方法注册成功后，实例化邮件类 EmailSenderByQq 并调用邮件方法发送邮件。

从程序上我们可以看出，User类依赖邮件对象 EmailSenderByQq，没有 EmailSenderByQq 就发不了邮件。这一切看起来都很正常。

突然有一天，QQ 的邮箱服务器故障了，怎么办？

我们需要更换邮件类，比如我们要把所有依赖 EmailSenderByQq 对象的代码全部更新为163的邮件对象 EmailSenderBy163。

但是，我们的项目很庞大，很多地方都实例化了 EmailSenderByQq，一个一个找到这个类，然后再一个一个的改掉，自己写的代码哭着也要说没事，万一是其他同事写的，心中瞬间有一万只@#*%**& 飞过...，项目的可维护性就不言而喻了。

有好的解决方法吗？

想一下我们刚刚讲过的控制反转，如果我们对二者进行解耦，会不会好一些？

我们把User类依赖的EmailSenderByQq对象，以构造函数的参数传递进去，降低User类对EmailSenderByQq对象的依赖。

```
class User
{
    private $_emailSenderObject;

    public function __construct($emailSenderObject)
    {
        $this->_emailSenderObject = $emailSenderObject;
    }
    public function register()
    {
        // other code

        $this->_emailSenderObject->send();
    }
}

$emailSenderObject = new EmailSenderByQq;
$user = new User($emailSenderObject);
$user->register();
```

以属性的方式同样也可以实现，代码参考如下

```
class EmailSenderBy163
{
    public function send()
    {
    }
}

class User
{
    public $emailSenderClass;

    public function register()
    {
        // other code

        $this->emailSenderClass->send();
    }
}

$user = new User;
$user->emailSenderClass = new EmailSenderBy163;
$user->register();
```

第一种通过构造参数传递对象和第二种通过属性传递对象的过程，其实就是依赖注入的体现。

“注入”，就是把一个实例传递到另一个实例的内部。

明白了以上，我们再继续对上面的代码优化一下

通过 EmailSenderByQq 类和EmailSenderBy163类，我们提炼一个 interface 接口,让 User类的register方法依赖interface接口的对象看起来更合适。

以构造函数“注入”实例为例，代码我们整理如下

```

interface EmailSender
{
    public function send();
}

class EmailSenderByQq implements EmailSender
{
    public function send()
    {
    }
}

class EmailSenderBy163 implements EmailSender
{
    public function send()
    {
    }
}

class User
{
    public $emailSenderClass;

    public function __construct(EmailSender $emailSenderObject)
    {
        $this->emailSenderClass = $emailSenderObject;
    }

    public function register()
    {
        // other code

        $this->emailSenderClass->send();
    }
}

$user = new User(new EmailSenderBy163);
$user->register();

```

如此，我们便实现了User类与EmailSender解耦的目的。

最后我们再来看看要讲解的依赖倒置原则。

依赖倒置原则（Dependence Inversion Principle, DIP），是一种软件设计思想。传统软件设计中，上层代码依赖于下层代码，当下层出现变动时，上层代码也要相应变化，维护成本较高。而DIP的核心思想是上层定义接口，下层实现这个接口，从而使得下层依赖于上层，降低耦合度，提高整个系统的弹性。这是一种经实践证明的有效策略。

回过神来我们发现，刚刚优化的代码是不是就是DIP原则的一种体现？

正如开篇所说，本文要讲解的内容不难，对吧？如果在学习的过程中有任何不理解的地方，下面尽管留言。

最后希望各位跟我一样，在成长的过程中，也学会慢慢优化代码，让你的代码更健壮一些，尽量不要再天马行空啦。

思考：尽管我们通过注入的方式解决了依赖问题，但是，假如A依赖B和C，B依赖D，C依赖E，D依赖F等等，那我们实例化A的时候，是不是更加复杂？

下一节我们就来说一说怎么解决这个问题。

提示：不熟悉反射相关知识的同学可以先去补补课啦。

评论区

共19条评论



说点什么吧...

发布



清风

里边有关于 restful 接口的吗

2018-04-28 14:52 回复



墨奇

是映射吗....还是反射????? - - 我搜到了光

2018-01-17 20:25 回复



3861398**@qq.com

之前用过 `laravel` 一个月，现在看这篇文章，还真是简单易懂。

2017-12-22 15:38 回复



starry-gala**@foxmail.com

干货，这教程费花的值。

2017-11-09 11:40 回复



庄欠林

反射在哪篇文章讲过？

2017-10-22 11:00 回复



白狼

反射基本上是一系列相关的api，可以参考这里
<http://php.net/manual/zh/book.reflection.php> 作为了解

2017-10-22 19:18 回复



欢乐豆

狼哥，最后那段代码，构造函数

```
public function __construct(EmailSender $emailSenderObject){ }
```

中参数的写法是指\$emailSenderObject必须是实现EmailSender接口的子类吗？

跟捕获异常那个try{ } catch(\Exception \$e){ }是一个道理么？

2017-09-27 17:40 回复



白狼

php5新增的“类型约束”，具体参考php手册

<http://php.net/manual/zh/language.oop5.typehinting.php>

2017-09-27 17:51 回复



3861398**@qq.com

PHP7 中新增了标量『类型约

束』，<http://php.net/manual/en/functions.arguments.php#functions.arguments.type-declaration>

2017-12-22 15:36 回复



8254554**@qq.com

虽然我会laravel,而yii2一点都不会,但是我能看懂,写的不错

2017-09-24 19:35 回复



Ali

@白狼 这个模块什么时候能够讲完呢？大概 时间周期是？

2017-09-11 13:34 回复



白狼

预计是2-3个月

2017-09-11 15:50

[回复](#)



hellohongti**@163.com

下节课：容器，嘿嘿嘿

2017-09-11 10:27

[回复](#)



白狼

哈哈，聪明

2017-09-11 10:48

[回复](#)



hello

很详细，通俗易懂，点赞！期待更多优质文章

2017-09-10 21:00

[回复](#)

1

[2 \(/comment/more?id=58&type=2&page=2\)](/comment/more?id=58&type=2&page=2)

[下一页 \(/comment/more?id=58&type=2&page=2\)](/comment/more?id=58&type=2&page=2)

友情链接

Bootstrap中文网 (<http://www.bootcss.com>)

开放CDN服务 (<http://www.bootcdn.cn>)

Yii2.0官方文档 (<http://www.yiiframework.com>)

阿里云 (<http://www.aliyun.com/>) 码云 (<http://git.oschina.net/>)

Yii中文社区 (<http://www.yiichina.com/>)

一点PHP (<https://www.yidianphp.com>) 站点地图 (</sitemap.xml>)


联系我

email: bailangzhan@qq.com

QQ群: 147533993 (已满) , 群2: 659817663

服务商

 又拍云 (<https://www.upyun.com>)

浙ICP备16006462号-1 (<http://www.miibeian.gov.cn/>)  浙公网安备 33010202000482号
(<http://www.beian.gov.cn/portal/registerSystemInfo?recordcode=33010202000482>) Copyright ©
白狼栈 (/) 2016-2018