PHP反射在设计模式中的应用

【PHP高级特性】之反射

在策略模式中的应用

在依赖注入模式中的应用

在代理模式中的应用

在装饰器模式中的应用

【PHP高级特性】之反射

什么是反射：

反射在每个面向对象的编程语言中都存在，它的主要目的就是在运行时分析类或者对象的状态，导出或提取出关于类、方法、属性、参数等的详细信息，包括注释。

有什么用：

在偏向底层一些的代码中，比如依赖注入、对象池、动态代理、自动获取插件列表、自动生成文档以及一些设计模式等等，都会大量运用到反射技术。

PHP 的反射 API 很多，但是常用的一般都是 `ReflectionClass` 和 `ReflectionMethod`

【PHP高级特性】之反射

ReflectionClass::getMethods　获取方法的数组

ReflectionClass::getMethod　获取指定方法

ReflectionClass::getName　获取类名

ReflectionClass::hasMethod　检查方法是否已定义

ReflectionClass::newInstance　newInstanceArgs　创建一个新的类实例

ReflectionMethod::invoke　执行

ReflectionMethod::invokeArgs　带参数执行

ReflectionMethod::isConstructor 判断方法是否是构造方法

ReflectionMethod::isPublic　判断方法是否是公开方法

ReflectionMethod::setAccessible 设置方法是否访问

【PHP高级特性】之反射

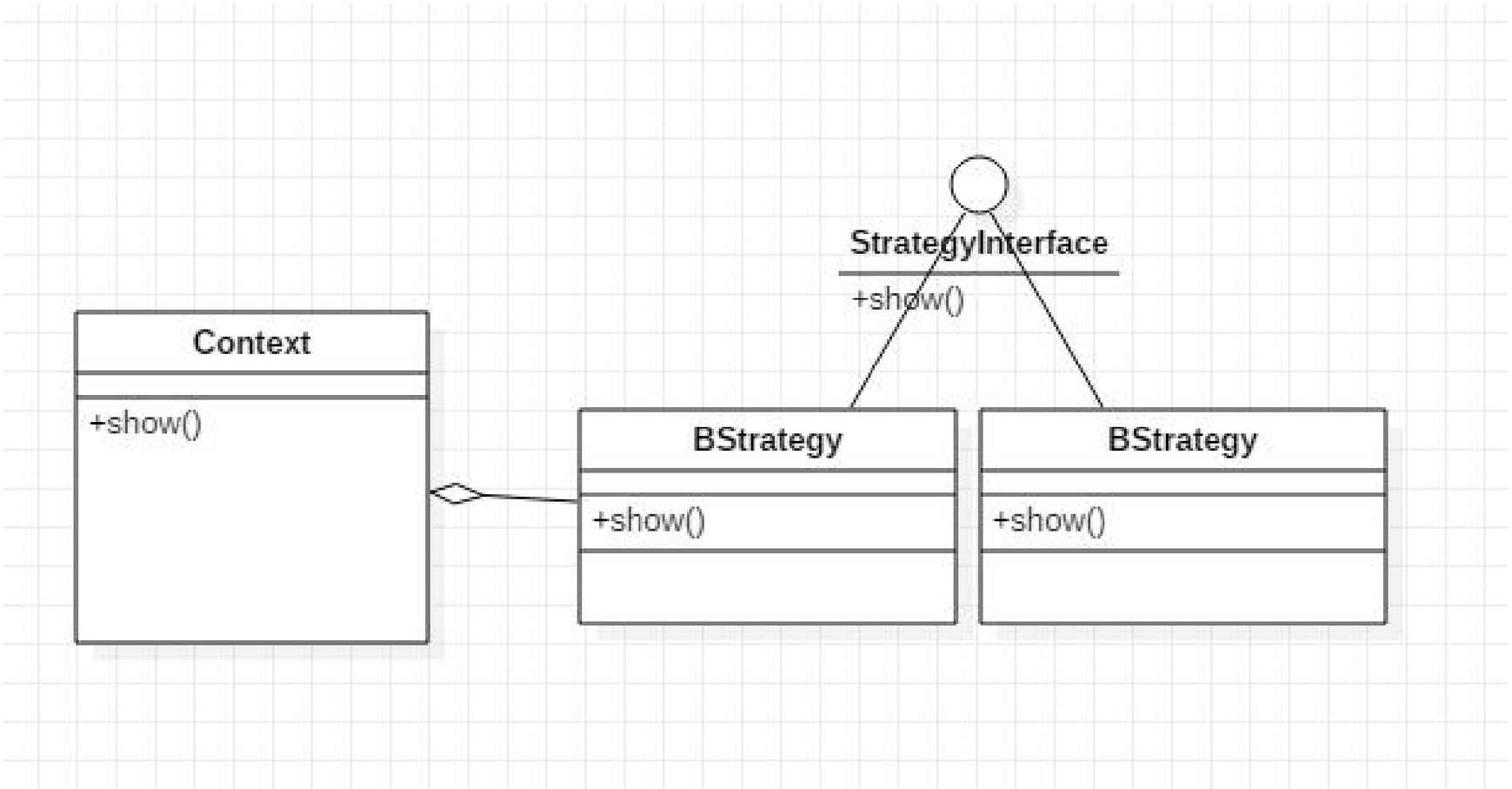
```
class User
{
    private $name;

    public function setName($name)
    {
        $this->name = $name;
    }
    protected function getName()
    {
        return $this->name;
    }
}

$user = new User();
$ref = new ReflectionClass(User::class);
$method = $ref->getMethod('setName');
$method->invoke($user, 'zhangshan');
//执行私有方法
$method = $ref->getMethod('getName');
$method->setAccessible(true);
$data = $method->invoke($user);
var_dump($data);
```

反射在策略模式中的应用

定义：

策略模式，顾名思义，就是提供多个策略的模式，用户在不同的情况下可以选择不同的策略，比如商场的打折策略（不同节假日不同的折扣方式），旅游出行的方式（提供飞行，或者火车，或者大巴的方式）。再进一步讲，就是把这些同一个系列的不同的算法封装起来，让它们能够被客户自由地使用。

UML：



反射在策略模式中的应用

使用场景：

在一个条件语句中又包含了多个条件语句就会使得代码变得臃肿，维护的成本也会加大

多个类只区别在表现行为不同，在运行时动态选择具体要执行的行为。

需要在不同情况下使用不同的策略(算法)，或者策略还可能在未来用其它方式来实现

(1)

```
$a = 'A';
if ($a == 'A') {
        echo 'a';
} else if ($a == 'B') {
        echo 'b';
} else {
        echo '暂无';
}
```

反射在策略模式中的应用

(2)

```
$context = new Context();
if ($a == 'A') {
      $context->setStrategy(new AStrategy());
} elseif ($a == 'B') {
    $context->setStrategy(new BStrategy());
} else {
    throw new UserException('暂无');
}
echo $context->show();
```

(3)

```
$strategy = Context::getInstance('app\\components\\strategy\\' . $a . 'Strategy');
echo $strategy->show()
```

反射在策略模式中的应用

优点：

1）良好的扩展性。增加一种策略，只要实现接口，写上具体逻辑就可以了。当旧策略不需要时，直接剔除就行。

2）良好的封装性。策略的入口封装在 Context 封装类中，客户端只要知道使用哪种策略就传哪种策略对象就可以了

3）避免了像简单工厂模式这样的多重条件判断。

缺点：

1）客户端必须了解策略组的各个策略，并且决定使用哪一个策略，也就是各个策略需要暴露给客户端。

2）如果策略增多，策略类的数量就会增加。

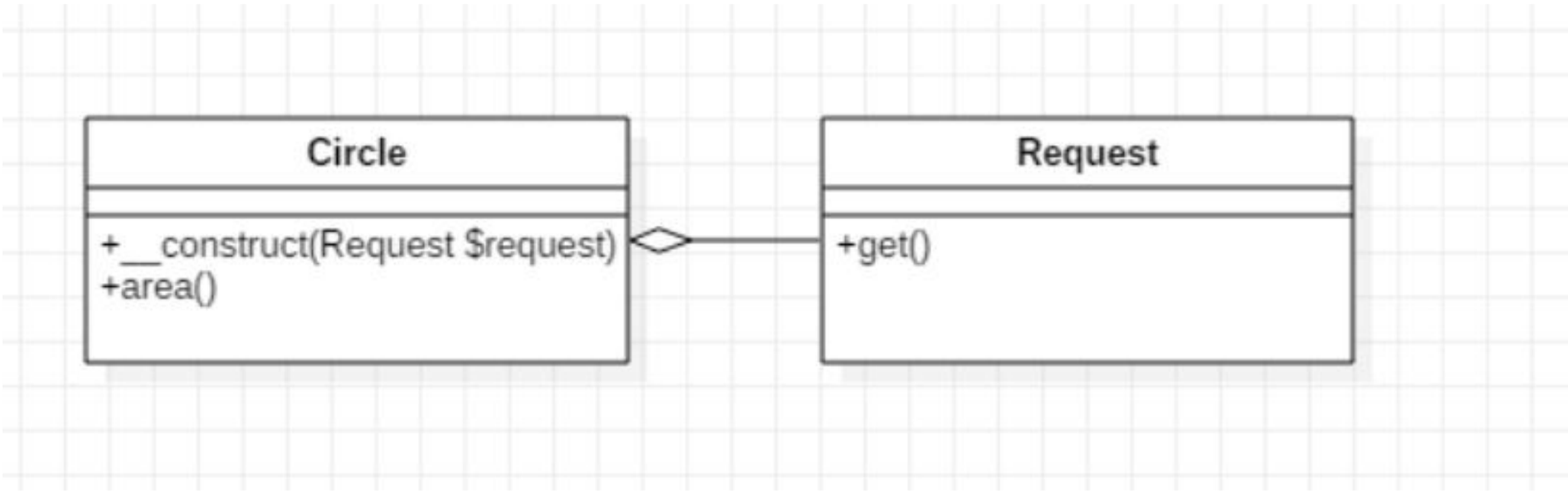
反射在依赖注入模式中的应用

定义：

通过以**构造函数**参数，**设值方法**或**属性字段**等方式将具体组件传递给依赖方

依赖注入不是目的，它是一系列工具和手段，最终的目的是帮助我们开发出松散耦合、可维护、可测试的代码和程序

UML：



反射在依赖注入模式中的应用

（1）

```
$circle = new Circle(new Request());
$circle->area();
```

（2）

```
$circle = Application::make(Circle::class);
$circle->area();
```

优点：

提供系统解耦的能力

可以明确的了解到组件之间的依赖关系

缺点：

需要我们自己管理注入的对象。

在实际应用中，我们通常需要实现一个容器去管理和实现依赖对象的注入。

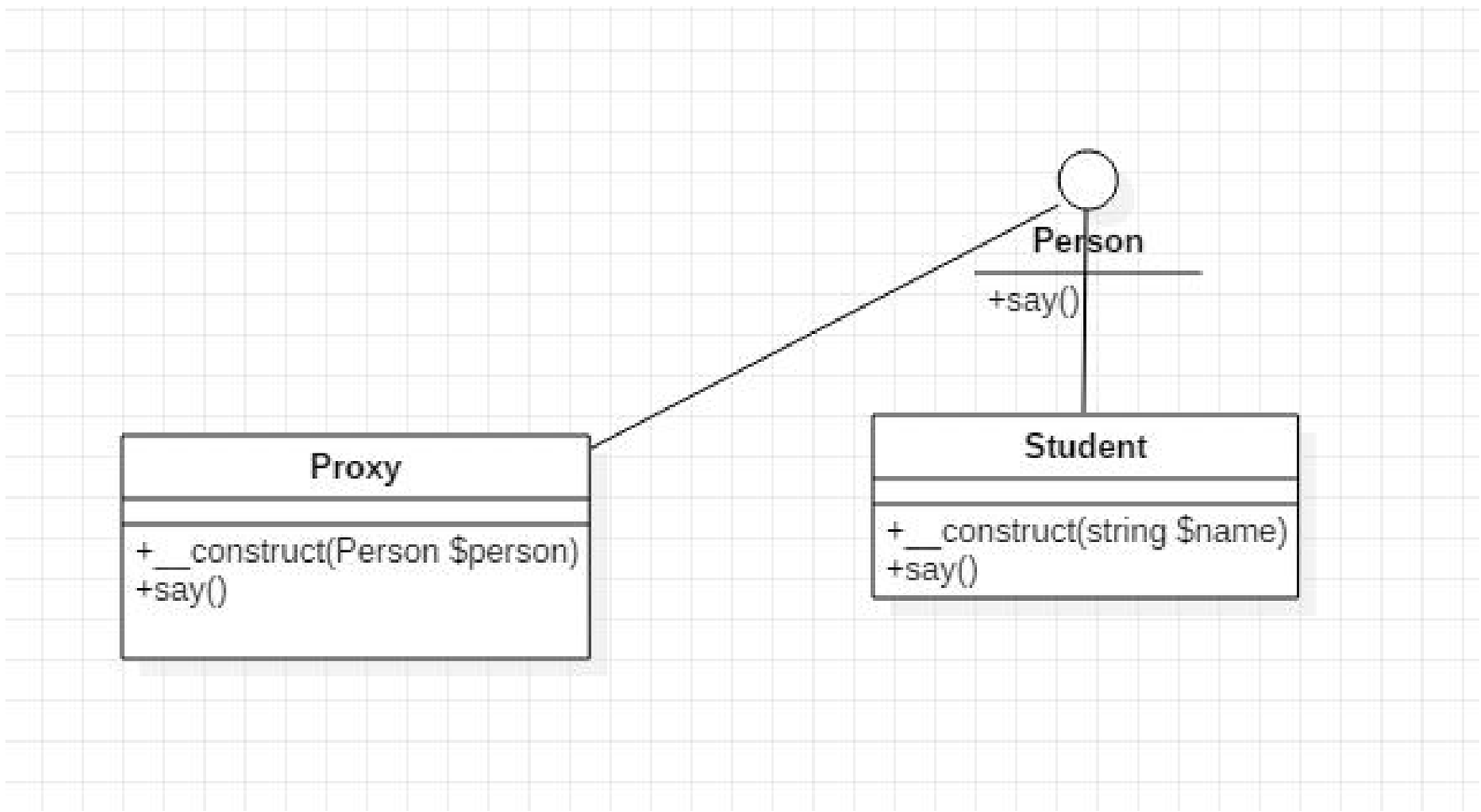
实际上，PHP 的常用 Web 框架中都是这么做的。

反射在代理模式中的应用

定义：

一种对象结构型模式。给某一个对象提供一个代理，并由代理对象控制对原对象的引用。

UML：



反射在代理模式中的应用

(1)

```
$subject = new Student("张三");
$proxy = new Proxy($subject);
$proxy->say();
```

(2)

```
$productService = new \app\components\proxy\Request(ProductService::class);
$list = $productService->list();
```

优点：

1）代理模式能够协调调用者和被调用者，在一定程度上降低了系统的耦合度。

2）远程代理使客户端可以访问在远程机器上的对象，远程机器 可能具有更好的计算性能与处理速度，可以快速响应并处理客户端请求。

3）虚拟代理通过使用一个小对象来代表一个大对象，可以减少系统资源的消耗，对系统进行优化并提高运行速度。

4）保护代理可以控制对真实对象的使用权限。

缺点：

1）由于在客户端和真实主题之间增加了代理对象，因此 有些类型的代理模式可能会造成请求的处理速度变慢。

2）实现代理模式需要额外的工作，有些代理模式的实现非常复杂。

3）让代理对象控制目标对象的访问，并且可以在不改变目标对象的情况下添加一些额外的功能。

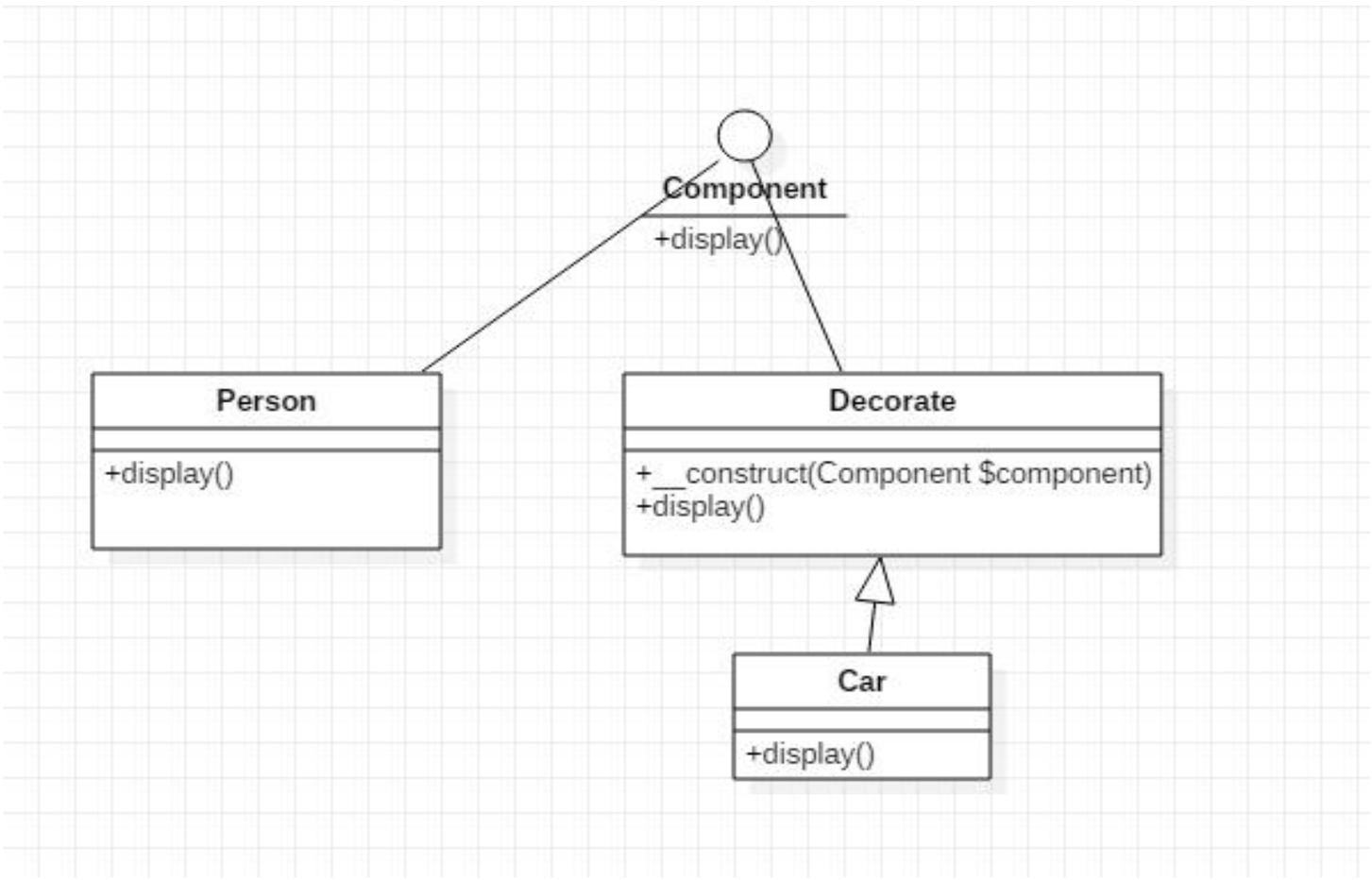
反射在装饰器模式中的应用

定义：

允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。

这种模式创建了一个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。

UML：



反射在装饰器模式中的应用

(1)

```
$person = new Person();
$car = new Car($person);
$car->display();
```

(2) 利用注解反射

```
/**
* @\app\components\annotations\Car()
*/
```

优点：

1）和继承的共同特点就是扩展对象的功能，而装饰器模式比继承更加灵活，可以在不改变原类文件和使用继承的情况下，动态的扩展一个对象的功能

2）通过使用不同的具体装饰器类，及其不同的排列组合，可以产生出大量不同的组合

缺点：

1）比继承更加复杂.

2）会出现一些小类，过度使用会使程序变得复杂

反射在装饰器模式中的应用

装饰器模式和代理模式的区别：

代理模式，注重对对象某一功能的流程把控和辅助。它可以控制对象做某些事，重心是为了借用对象的功能完成某一流程，而非对象功能如何。

装饰模式，注重对对象功能的扩展，它不关心外界如何调用，只注重对对象功能的加强，装饰后还是对象本身。

对于代理类，如何调用对象的某一功能是思考重点，而不需要兼顾对象的所有功能；

对于装饰类，如何扩展对象的某一功能是思考重点，同时也需要兼顾对象的其它功能，因为再怎么装饰，本质也是对象本身，要担负起对象应有的职责。