php设计模式之装饰模式应用案例详解 更新时间：2019年06月17日 11:34:52 作者：赵筱蕊

..这篇文章主要介绍了php设计模式之装饰模式,结合具体应用案例形式详细分析了php装饰模式的概念、原理、用法及相关操作注意事项,需要的朋友可以参考下

. .本文实例讲述了php设计模式之装饰模式。分享给大家供大家参考，具体如下：

介绍

•装饰者模式（Decorator Pattern）允许你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。

主要角色

•抽象构件(Component)角色：定义一个独享接口，以规范准备接收附加职责的对象，从而可以给这些对象动态的添加职责。

•具体构件(Concrete Component)角色：定义一个将要接收附加职责的类。

•装饰(Decorator)角色：持有一个指向Component对象的指针，并定义一个与Component接口一致的接口。

•具体装饰(Concrete Decorator)角色：负责给构件对象增加附加的职责。

下面是使用装饰模式的一个简单实现:

?

1234567891011121314151617 class RequestHelper{} abstract class ProcessRequest{ abstract function process(RequestHelper $req); } class MainProcess extends ProcessRequest{ function process(RequestHelper $req) { print \_\_CLASS\_\_.": doing something useful with request\n"; } } abstract class DecorateProcess extends ProcessRequest{ protected $processRequest; function \_\_construct(ProcessRequest $pr) { $this->processRequest = $pr; } }

class RequestHelper{}

abstract class ProcessRequest{

abstract function process(RequestHelper $req);

}

class MainProcess extends ProcessRequest{

function process(RequestHelper $req)

{

print \_\_CLASS\_\_.": doing something useful with request\n";

}

}

abstract class DecorateProcess extends ProcessRequest{

protected $processRequest;

function \_\_construct(ProcessRequest $pr)

{

$this->processRequest = $pr;

}

}和之前一样，我们定义了一个抽象基类(ProcessRequest)、一个具体的组件(MainProcess)和一个抽象装饰类(DecorateProcess)。 MainProcess::process()方法仅仅报告方法被调用，并没有其他功能。DecorateProcess为他的子类保存了一个ProcessRequest对象。下面是一些简单的具体装饰类：

?

123456789101112131415161718192021 class LogRequest extends DecorateProcess{ function process(RequestHelper $req) { print \_\_CLASS\_\_.": logging request\n"; $this->processRequest->process($req); } } class AuthenticateRequest extends DecorateProcess{ function process(RequestHelper $req) { print \_\_CLASS\_\_.": authenticating request\n"; $this->processRequest->process($req); } } class StructureRequest extends DecorateProcess{ function process(RequestHelper $req) { print \_\_CLASS\_\_.": structuring request\n"; $this->processRequest->process($req); } }

class LogRequest extends DecorateProcess{

function process(RequestHelper $req)

{

print \_\_CLASS\_\_.": logging request\n";

$this->processRequest->process($req);

}

}

class AuthenticateRequest extends DecorateProcess{

function process(RequestHelper $req)

{

print \_\_CLASS\_\_.": authenticating request\n";

$this->processRequest->process($req);

}

}

class StructureRequest extends DecorateProcess{

function process(RequestHelper $req)

{

print \_\_CLASS\_\_.": structuring request\n";

$this->processRequest->process($req);

}

}装饰类的每一个process()方法在调用引用的processRequest对象的Process()方法前输出一条信息。

现在我们可以在运行时合并这些类的对象，创建过滤器来对每一个请求按不同的顺序执行不同操作。下面的代码将所有具体类的对象组合成为一个过滤器：

?

12345 $process = new AuthenticateRequest(new StructureRequest( new LogRequest( new MainProcess() ))); $process->process(new RequestHelper());

$process = new AuthenticateRequest(new StructureRequest(

new LogRequest(

new MainProcess()

)));

$process->process(new RequestHelper());执行代码会得到下面的输出结果：

Authenticate

Request: authenticating request

StructureRequest: structuring request

LogRequest: logging request

MainProcess: doing something useful with request

优点：

装饰类和被装饰类可以独立发展，不会相互耦合，装饰模式是继承的一个代替模式，装饰模式可以动态扩展一个实现类的功能。

缺点：

多层装饰比较负责。