第四章 PHP5 接口与多态

目录

4.1 接口的定义与规范

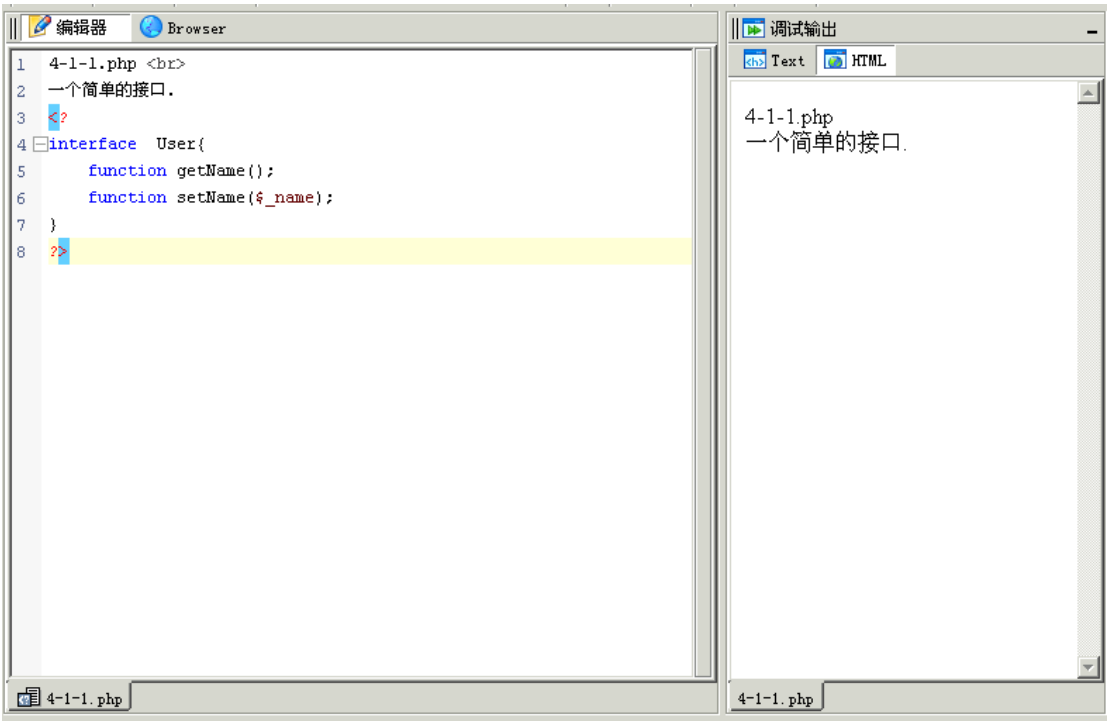
- 接口(interface)是**抽象方法**和**静态常量**定义的集合。
- 接口是一种特殊的抽象类，这种抽象类中只包含抽象方法和静态常量。
- 接口中没有其它类型的内容。

4.1.1 接口的定义

我们先写接口的定义，后面几节再介绍接口的意义。
下面的例子是接口的一个简单写法。

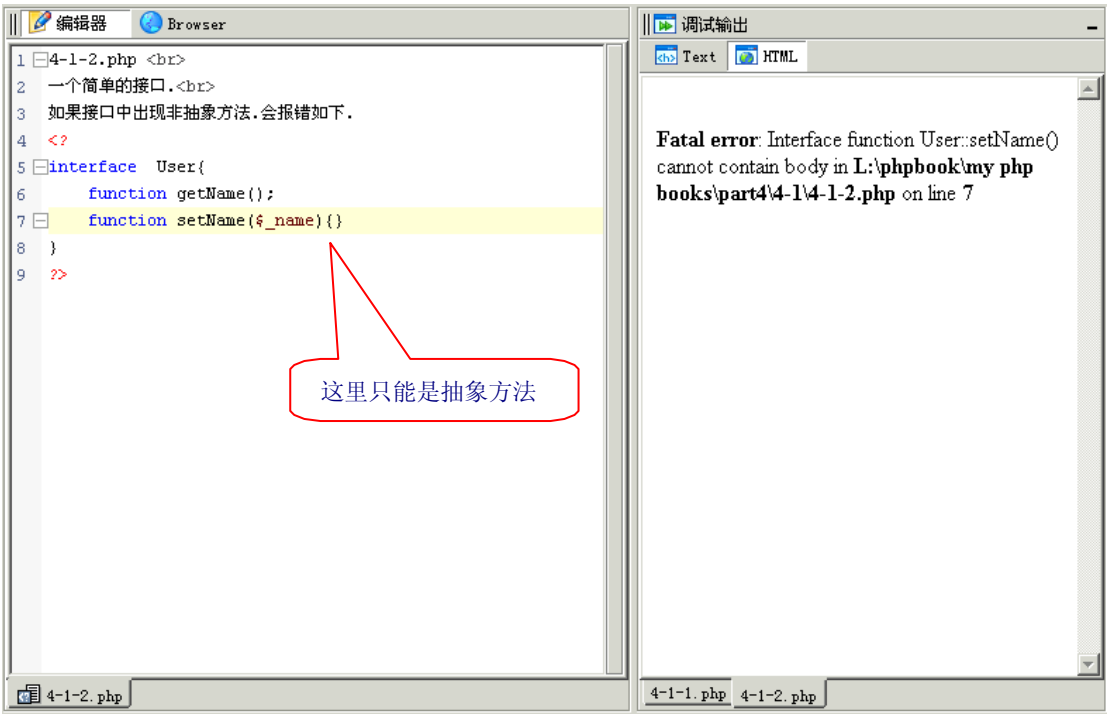
```
interface 接口名{
}
```

下面的例子定义了一个接口 User ，这个接口中有两个抽象方法，getName() 和 setName()。能看到接口的写法和类很相似。
例：4-1-1.php


编辑器
Browser
4-1-1.php
一个简单的接口.
<?
interface User{
function getName();
function setName($_name);
}
?>
4-1-1.php
调试输出
Text
HTML
4-1-1.php
一个简单的接口.
4-1-1.php

4.1.2 接口中的抽象方法

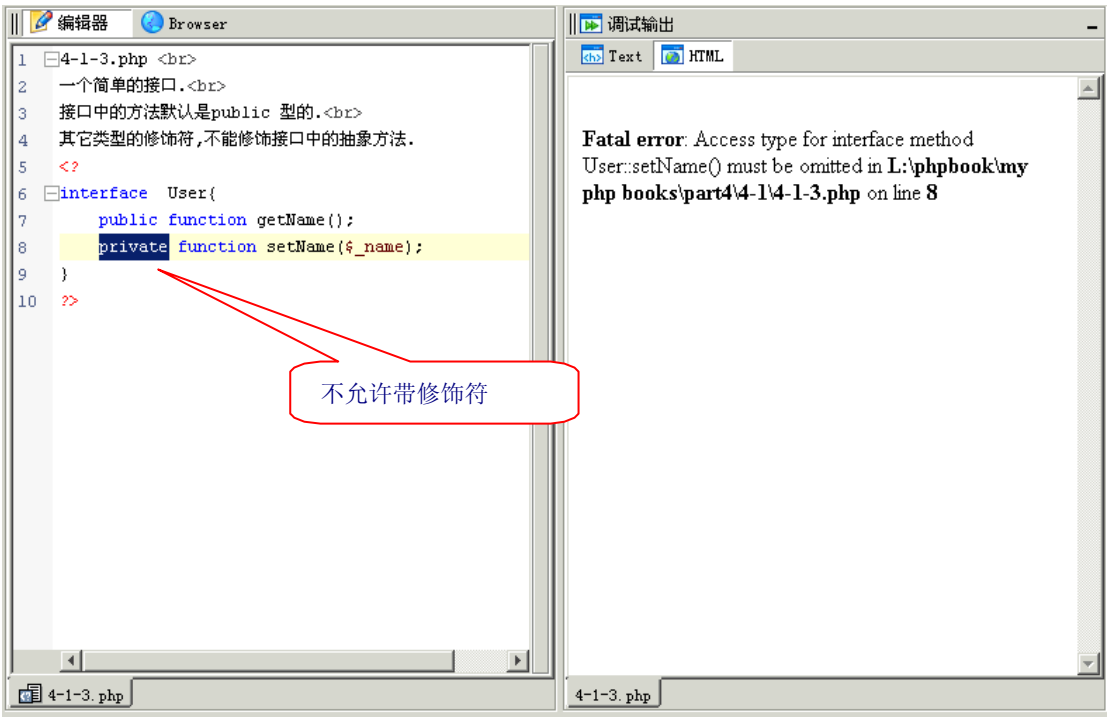
注意，在接口中只能有抽象方法。如果在接口中出现了非抽象方法，会报错如下：
Interface function User::setName() cannot contain body in …….
例 4-1-2.php



4.2.3 接口中抽象方法的修饰和访问权限

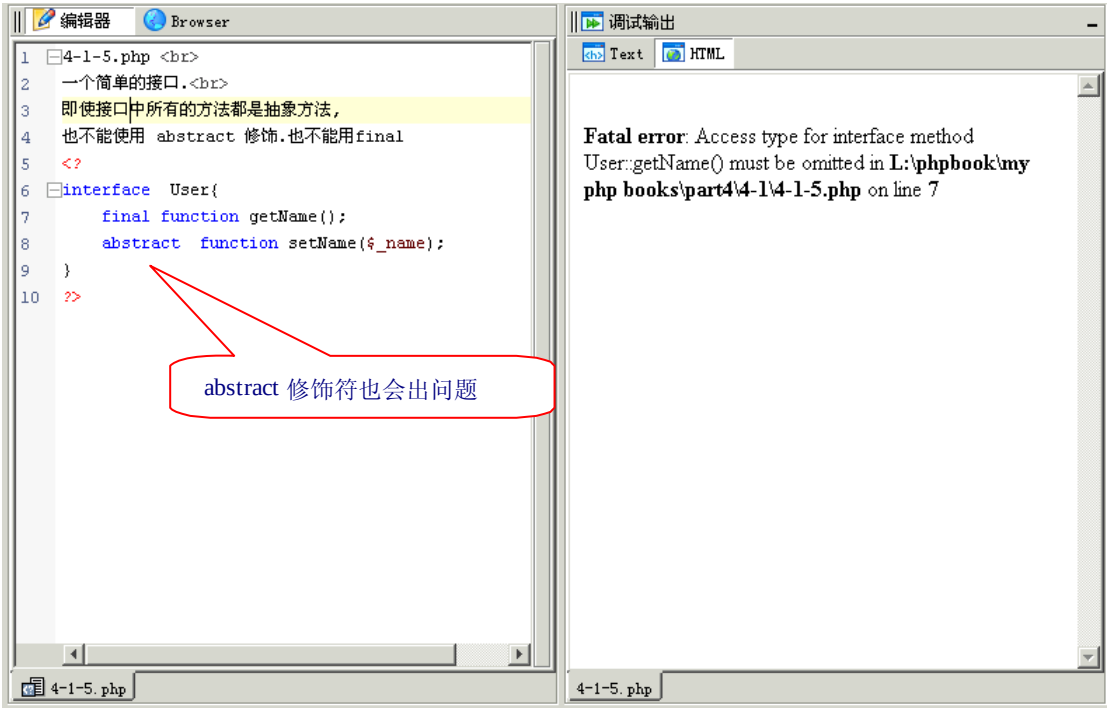
在接口中的抽象方法只能是 public 的，默认也是 public 权限。
并且不能设置成 private 或者 protected 类型。
否则会报错如下：
Access type for interface method User::setName() must be omitted in —on line —
（在接口中，访问类型必须忽略。）

例：4-1-3.php


编辑器
Browser
1 4-1-3.php
2 一个简单的接口.
3 接口中的方法默认是public 型的.
4 其它类型的修饰符,不能修饰接口中的抽象方法.
5 <?
6 interface User{
7 public function getName();
8 private function setName($_name);
9 }
10 ?>
不允许带修饰符
4-1-3.php
调试输出
Text
HTML
Fatal error: Access type for interface method User::setName() must be omitted in L:\phpbook\my php books\part4\4-1\4-1-3.php on line 8
4-1-3.php

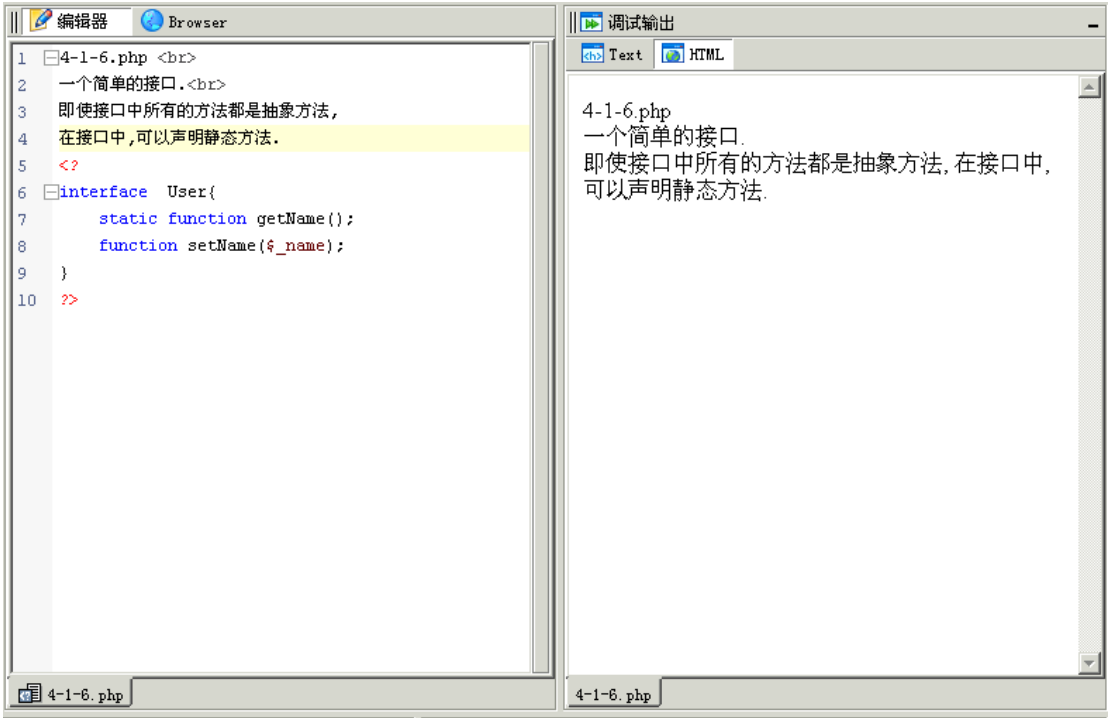
例：4-1-4.php
protected 访问权限也会有问题

例：4-1-5.php
即使 abstract 和 final 修饰符不能修饰接口中的抽象方法。



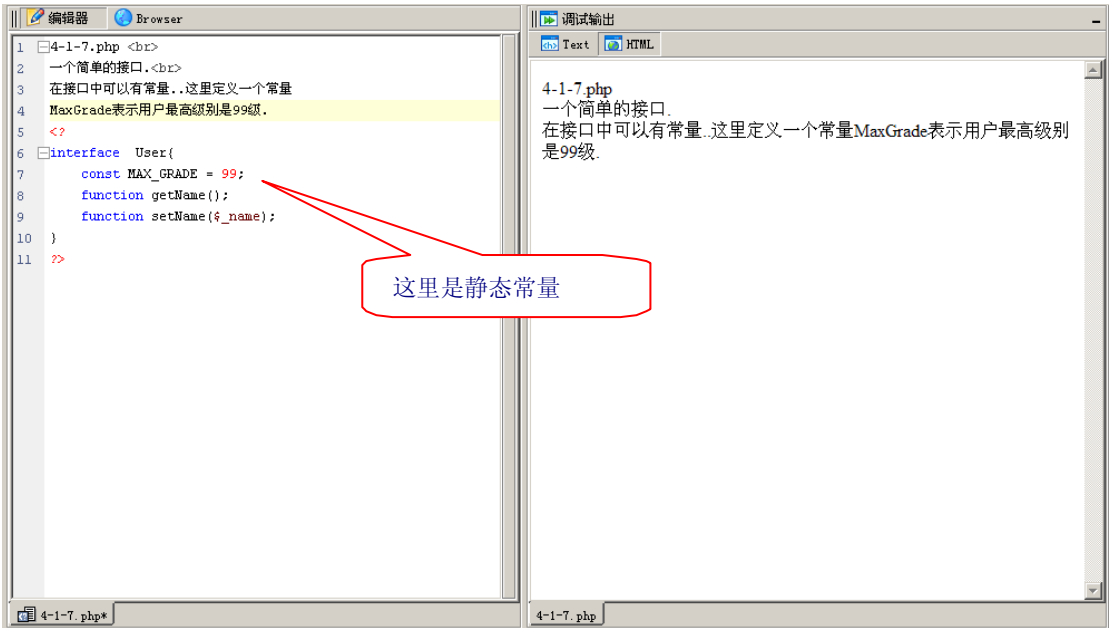
4.1.4 接口中的静态抽象方法

例：4-1-6.php
在接口中可以使用静态抽象方法。在 PHP5.2 中，不建议在抽象类中使用静态抽象方法。而接口中依然保留了静态抽象方法。



4.1.5 接口中的静态常量

在接口中可以定义静态常量。而且不用 static 修饰就是静态的常量。
例 4-1-7.php


编辑器
Browser
1 4-1-7.php
2 一个简单的接口.
3 在接口中可以有常量..这里定义一个常量
4 MaxGrade表示用户最高级别是99级.
5 <?
6 interface User{
7 const MAX_GRADE = 99;
8 function getName();
9 function setName($_name);
10 }
11 ?>
这里是静态常量
4-1-7.php*
调试输出
Text
HTML
4-1-7.php
一个简单的接口.
在接口中可以有常量..这里定义一个常量MaxGrade表示用户最高级别是99级.
4-1-7.php

4.2 实现接口

- 类实现接口要使用 implements 。
- 类实现接口要实现其中的抽象方法。
- 一个类可以实现多个接口。
-

一个类可以使用 implements 实现接口，甚至可以实现多个接口。

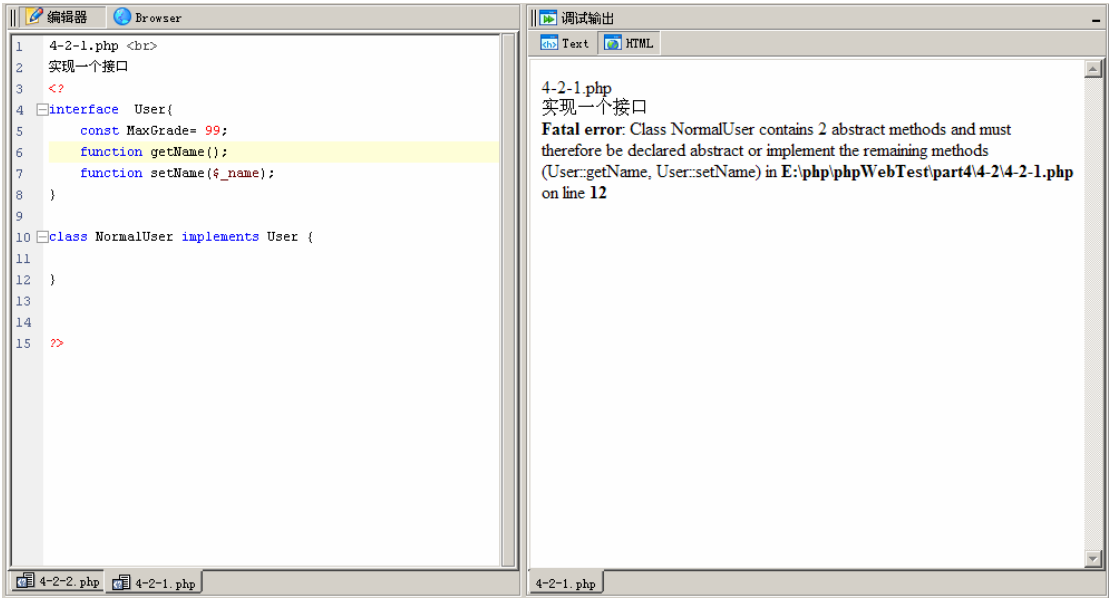
大部分的书说，这样是为了实现 PHP 的多继承。为什么呢？ PHP5 是单继承的，一个类只可以继承自一个父类。接口可以实现多个，这样就是多继承了。

这样说有些道理。但，既然接口里面的方法没有方法体，所谓的多继承又有什么意义？接口的意义在于后面一节继续说的多态。

4.2.1 使用 implements 实现接口

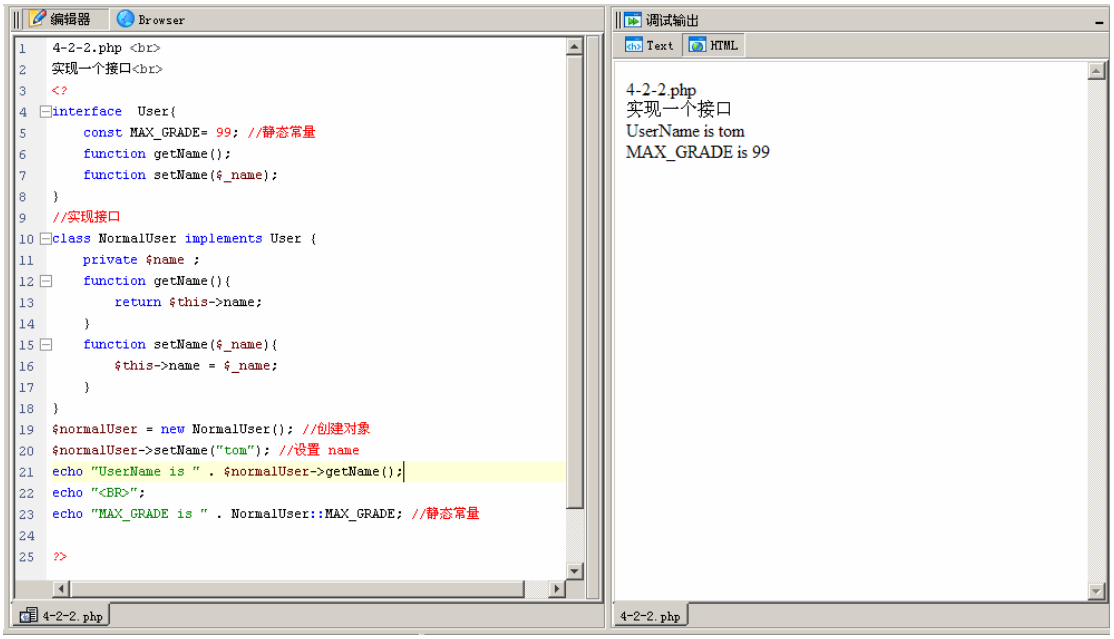
使用 implements 来实现一个接口。如果实现接口而没有实现其中的抽象方法，会报错如下。

例：4-2-1.php



例 4-2-2.php

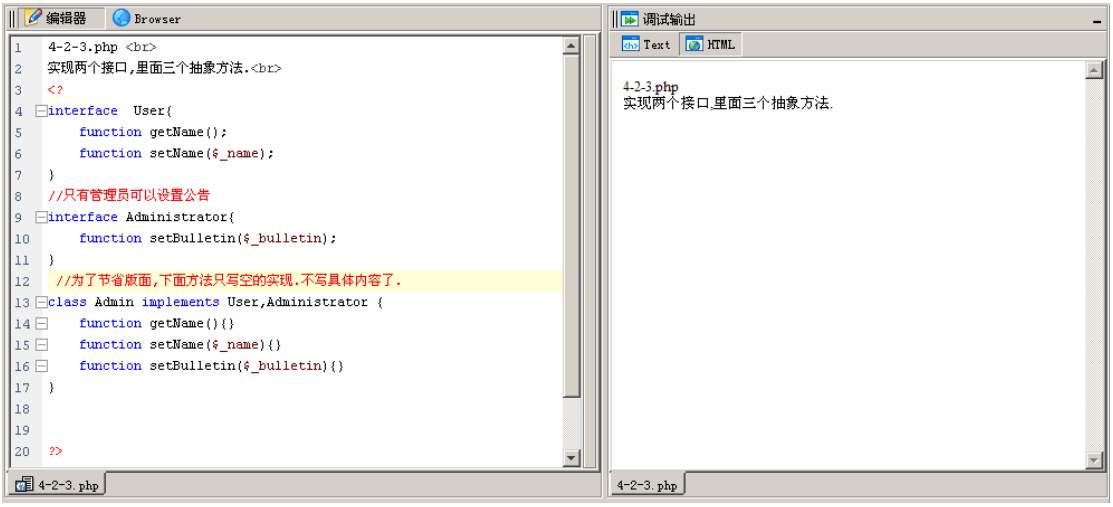
实现接口要实现方法。注意静态变量的使用。



4.2.2 实现多个接口

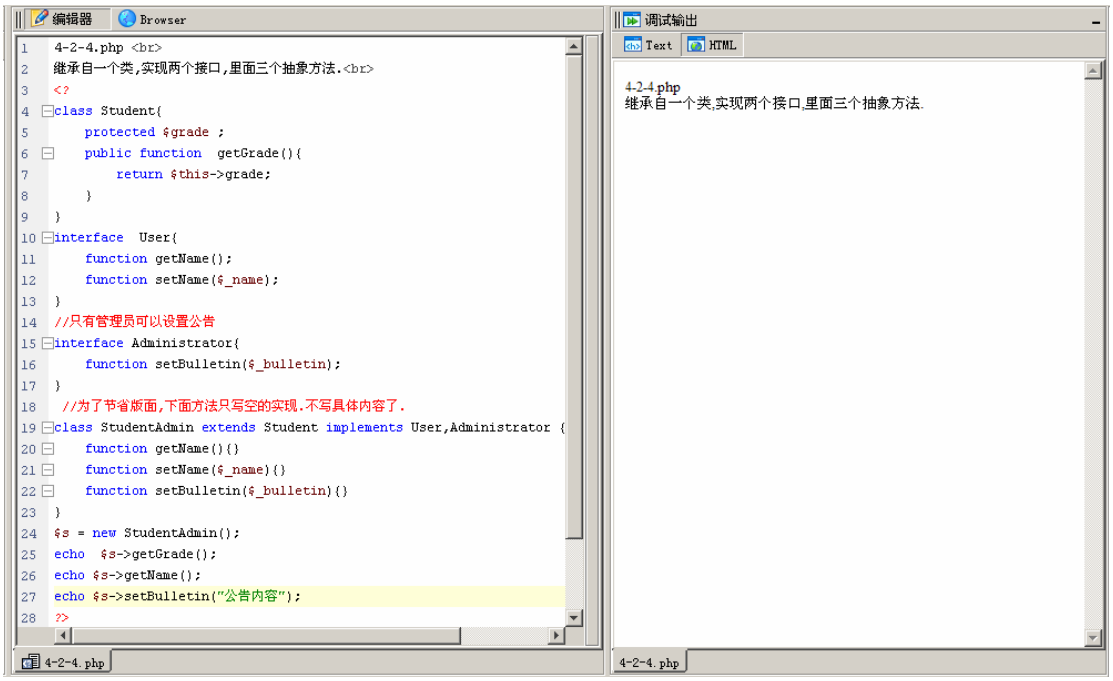
一个类可以实现多个接口。只要使用 ，号将多个接口链接起来就可以。

例： 4-2-3.php



4.2.3 继承并实现接口

这个例子，让 StudentAdmin 继承自 Student 类，并实现了 User 和 Administrator 接口。
例： 4-2-4.php



4.3 接口的继承

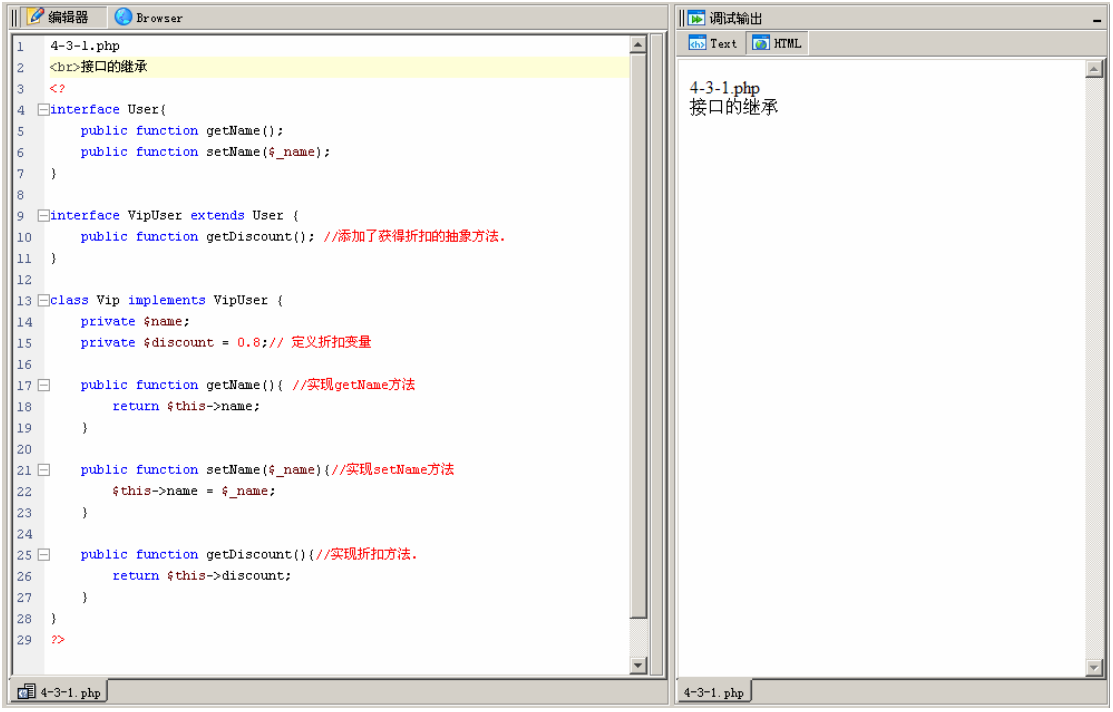
- 一个接口可以继承自另外的接口。
- PHP5 中的类是单继承，但是接口很特殊。一个接口可以继承自多个接口。
- 一个接口继承其它接口时候，直接继承父接口的静态常量属性和抽象方法。

在 PHP5 中，接口是可以继承自另外一个接口的。这样代码的重用更有效了。
要注意只有接口和接口之间使用 继承关键字 extends。
类实现接口必须实现其抽象方法，使用实现关键字 implements。

4.3.1 接口实现继承

要注意只有接口和接口之间使用 继承关键字 extends。
类实现接口必须实现其抽象方法，使用实现关键字 implements。

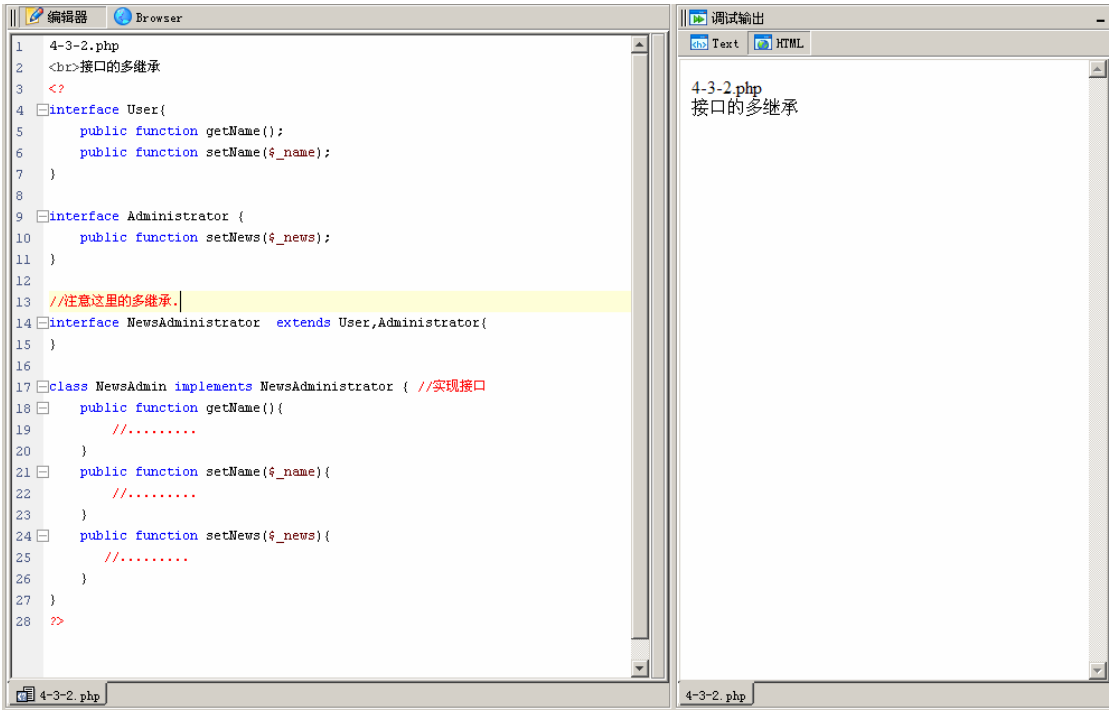
例 4-3-1.php
这个例子定义接口 User，User 有两个抽象方法 getName 和 setName。又定义了接口 VipUser，继承自 User 接口，并增加了和折扣相关的方法 getDiscount。
最后定义了类 Vip ，实现了 VipUser 接口。并实现了其中的三个方法。



4.3.2 接口可以实现多继承

接口可以实现多继承，这是接口很特殊的地方。注意下面的代码和用法。

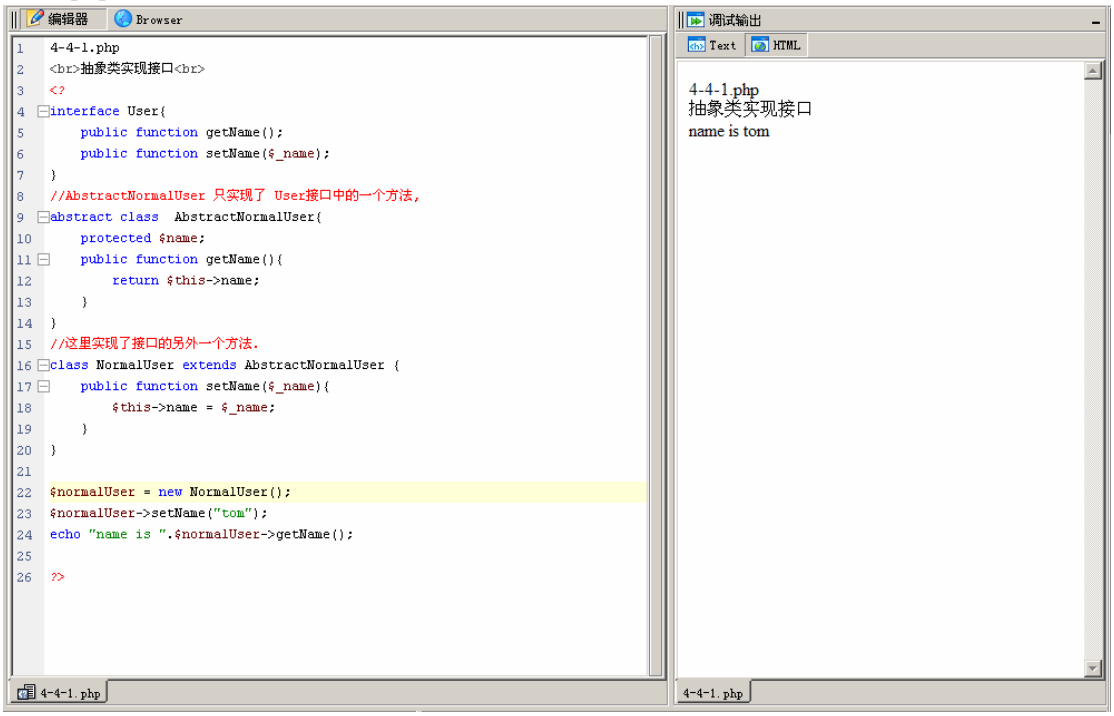
例：4-3-2.php



4.4 抽象类实现接口

- 抽象类实现接口，可以不实现其中的抽象方法，而将抽象方法的实现交付给具体能被实例化的类去处理。

4-4-1.php



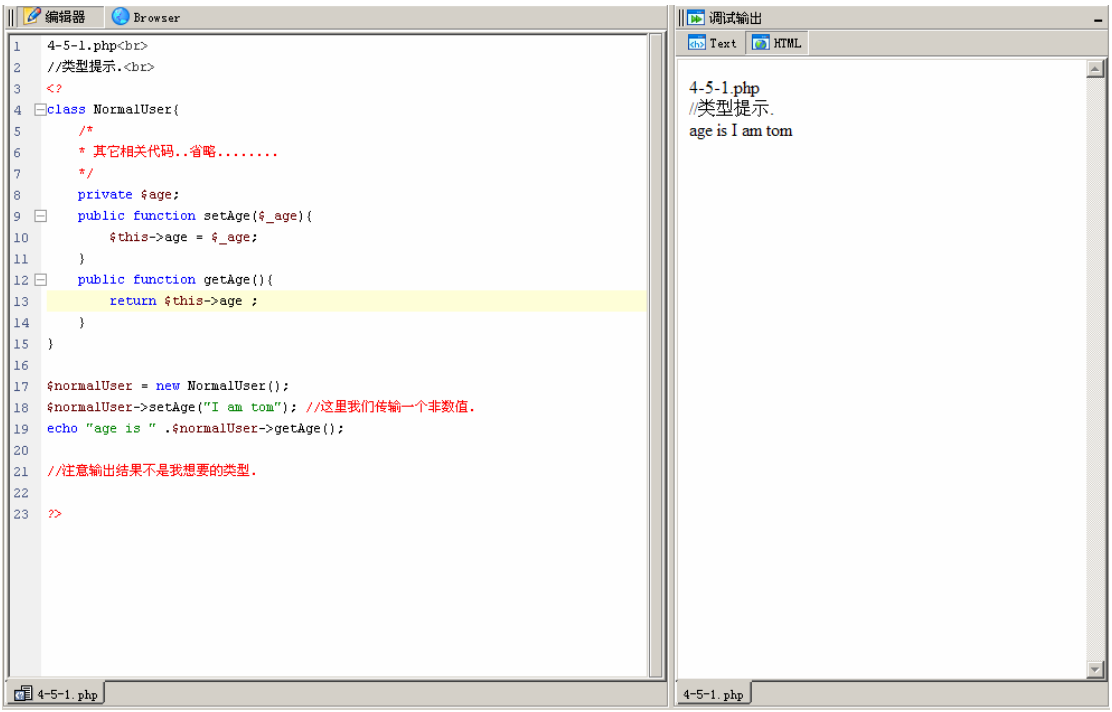
4.5 类型提示

PHP 是弱类型语言，向方法传递参数时候也不太区分类型。这样的使用会引起很多的问题，PHP 开发者认为，这些问题应该是由代码书写者在书写代码时进行检验以避免。

4.5.1 没有类型提示很危险

下面的代码可能会出现问题。

例： 4-5-1.php



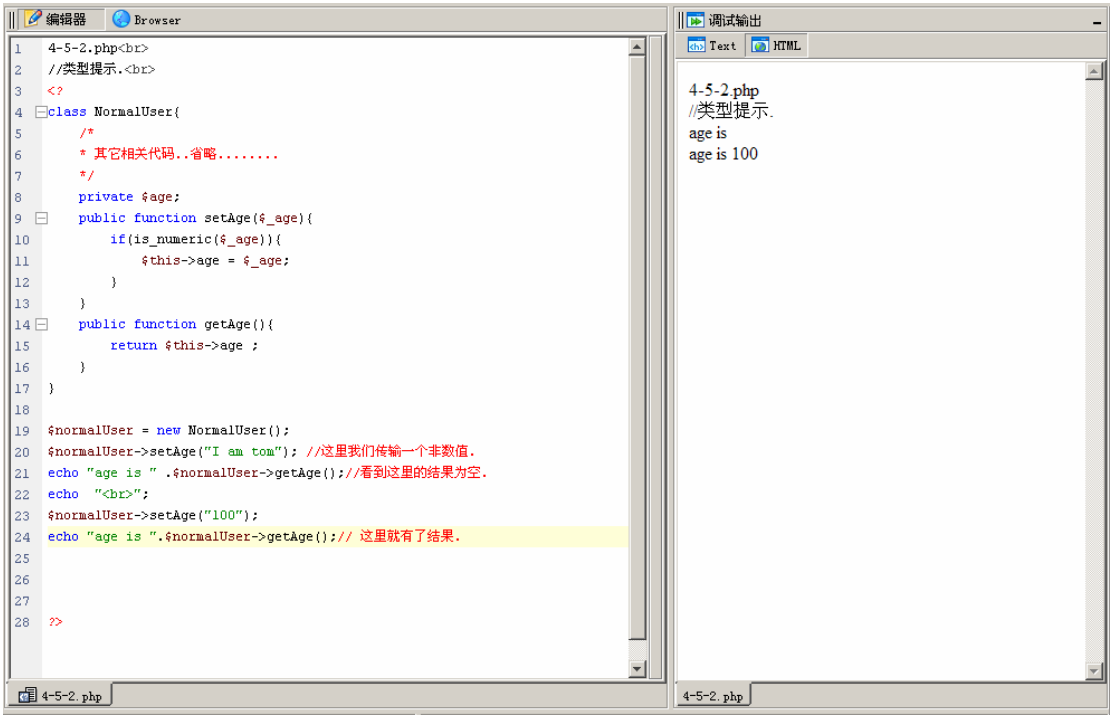
4.5.2 原始类型的类型判断

PHP 中提供了一些函数，来判断数值的类型。我们可使用 **is_numeric()**。判断是否是一个数值或者可转换为数值的字符串。

其它相关的还有 **is_bool()**、**is_int()**、**is_float()**、**is_integer()**、**is_numeric()**、**is_string()**、**is_array()** 和 **is_object()**。

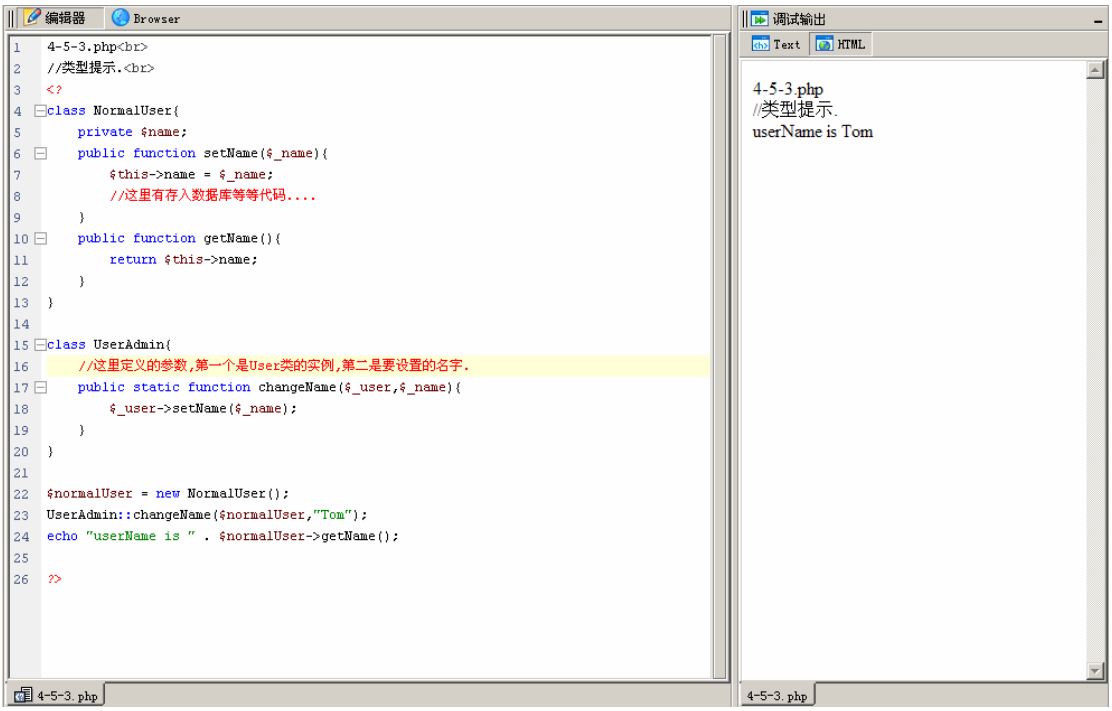
于是代码有了修改

例：4-5-2.php



4.5.3 向方法内传递对象

如果传递的参数是一个对象呢？
下面的代码用起来很正常。
例：4-5-3.php



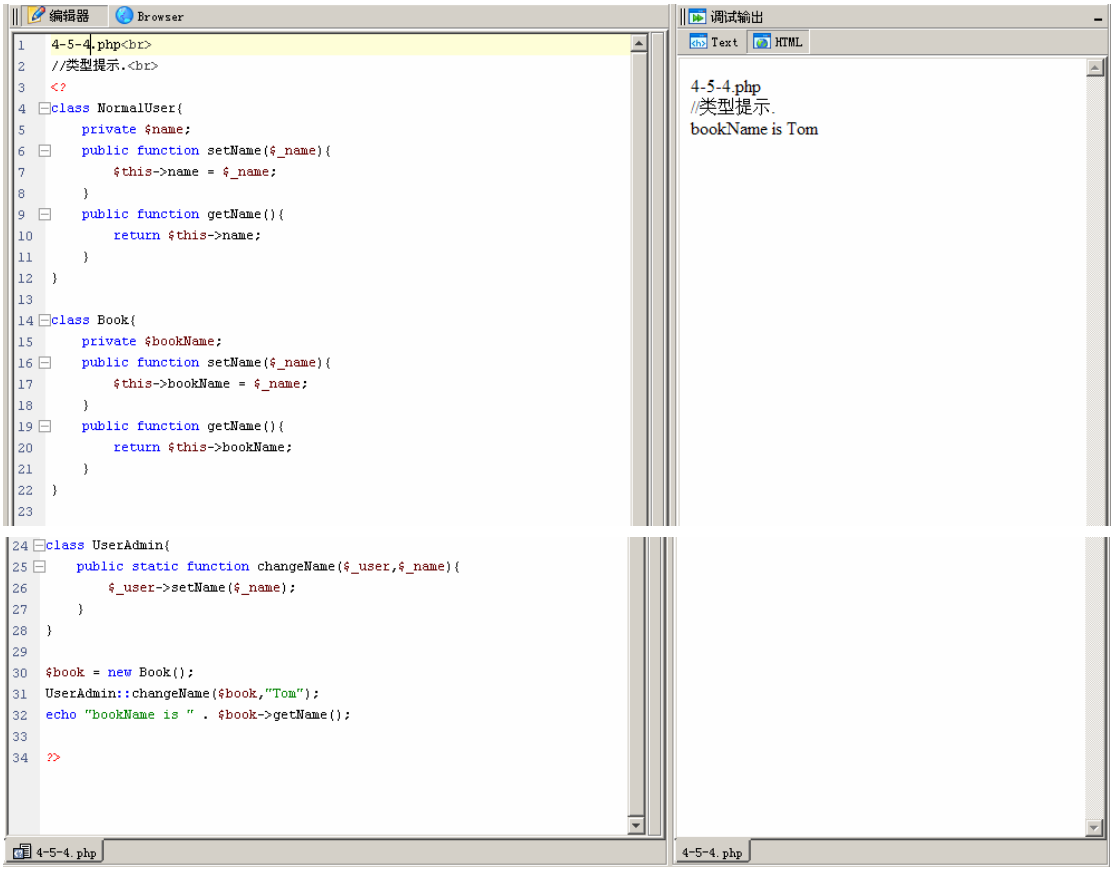
我们还有一个类，和图书相关的类，在图书类中也可以设置图书的书名 setName($name)。

如果我向，刚才代码中的方法 changeName() 中传入一个 Book 的实例，原定于改变人名的方法现在改变了书的书名。

这有什么风险？能把脏衣服扔到洗衣机里面去洗，同样的清洗，把盘子和碗都扔进洗衣机里面洗洗试试。

下面的代码演示我们不详看到的一幕。

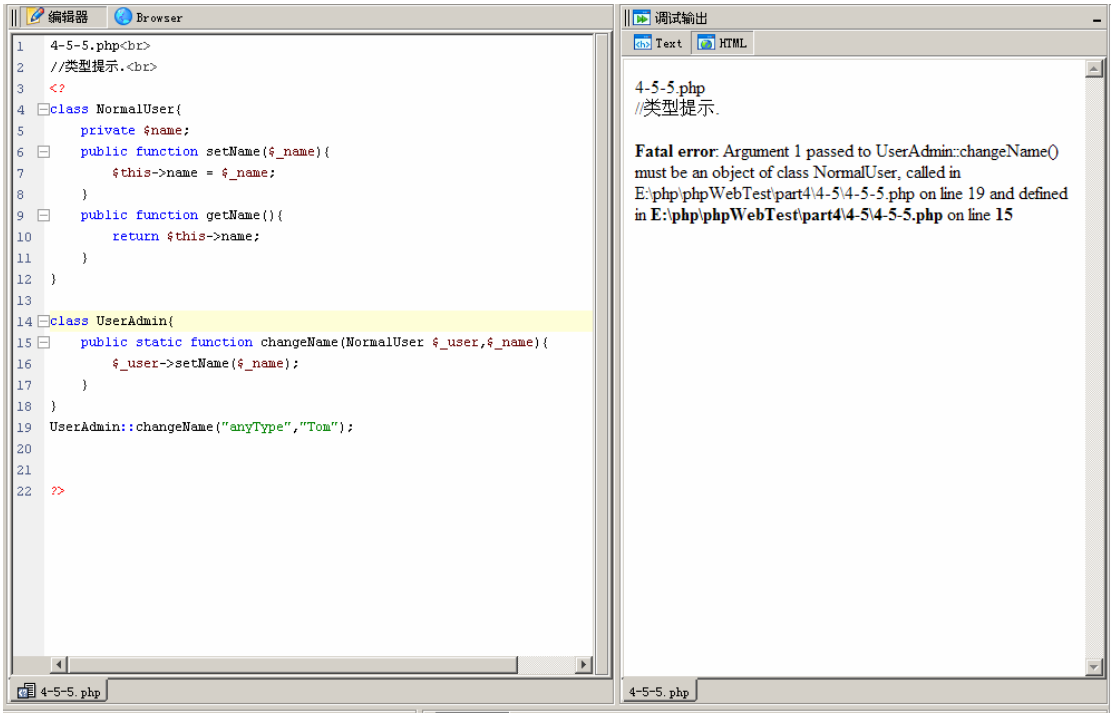
例：4-5-4.php




```
4-5-4.php<br>
//类型提示.<br>
<?
class NormalUser{
    private $name;
    public function setName($_name){
        $this->name = $_name;
    }
    public function getName(){
        return $this->name;
    }
}

class Book{
    private $bookName;
    public function setName($_name){
        $this->bookName = $_name;
    }
    public function getName(){
        return $this->bookName;
    }
}

class UserAdmin{
    public static function changeName($_user,$_name){
        $_user->setName($_name);
    }
}

$book = new Book();
UserAdmin::changeName($book,"Tom");
echo "bookName is " . $book->getName();

?>
```

4.5.4 类型提示保障数据安全

为了避免对象类型不规范引起的问题，PHP5 中引入了类型提示这个概念。
在定义方法参数时，同时定义参数的对象类型。
如果在调用的时候，传入参数的类型不对会报错。这样保证了数据的安全性。
例 4-5-5.php



- 建议在定义方法参数时，要使用类型提示。
- 如果类型不是对象，要采用代码进行类型建议，以增强安全性。

4.6 PHP5 中的多态

多态这个概念，在 Java 中指的是变量可以指向的对象的类型，可是变量声明类型的子类。对象一旦创建，它的类型是不变的，多态的是变量。

在 PHP5 中，变量的类型是不确定的，一个变量可以指向任何类型的数值、字符串、对象、资源等。我们无法说 PHP5 中多态的是变量。

我们只能说在 PHP5 中，多态应用在方法参数的类型提示位置。

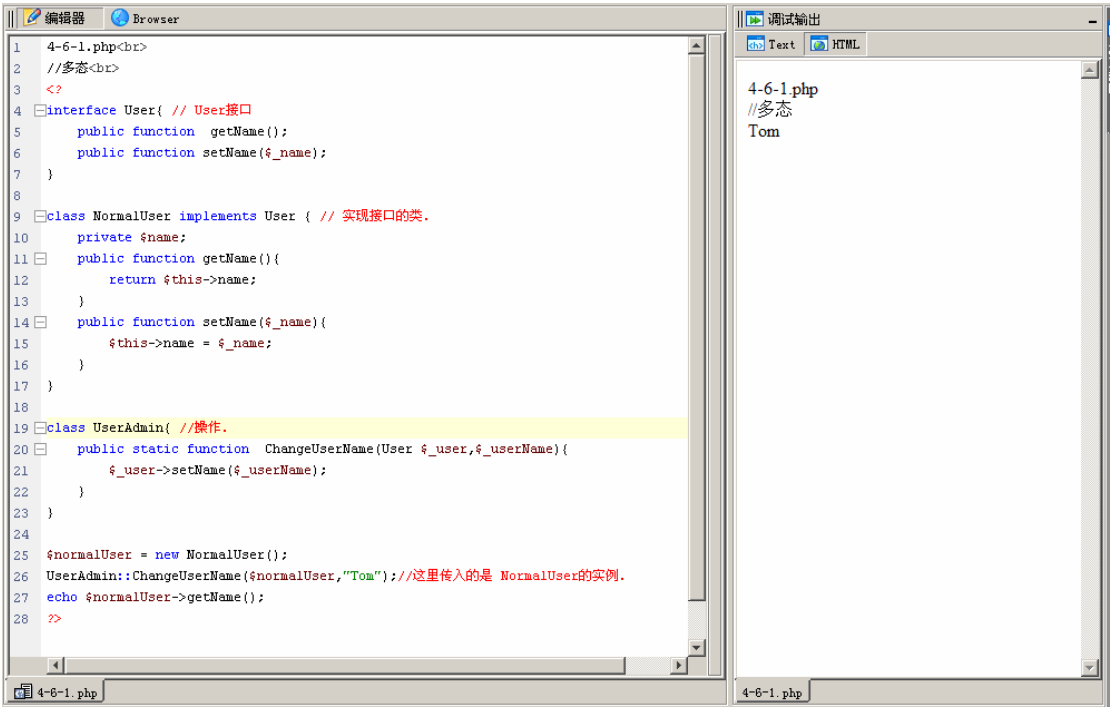
- 一个类的任何子类对象都可以满足以当前类型作为类型提示的类型要求。
- 所有实现这个接口的类，都可以满足以接口类型作为类型提示的方法参数要求。
- 简单的说，一个类拥有其父类、和已实现接口的身份。

4.6.1 通过实现接口实现多态

下面的例子中，UserAdmin 类的静态方法，要求一个 User 类型的参数。

在后面的使用中，传递了一个实现了 User 接口的类 NormalUser 的实例。代码成功运行。

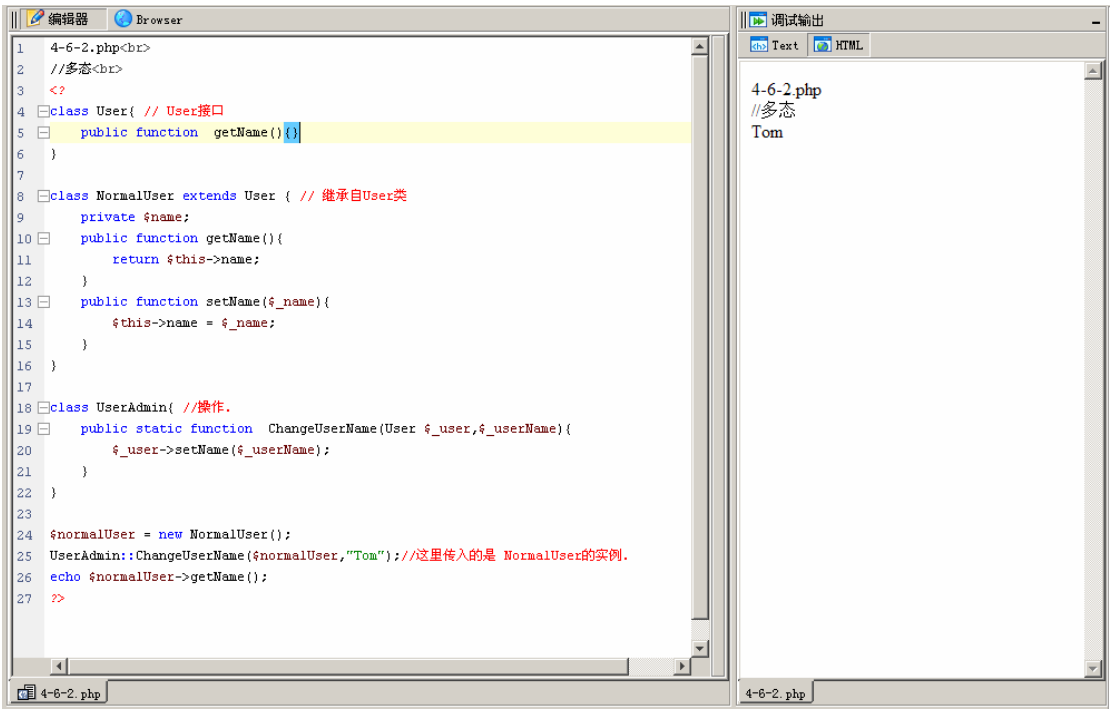
例 4-6-1.php



4.6.2 通过继承关系实现多态

下面是类和子类的关系。

例：4-6-2.php



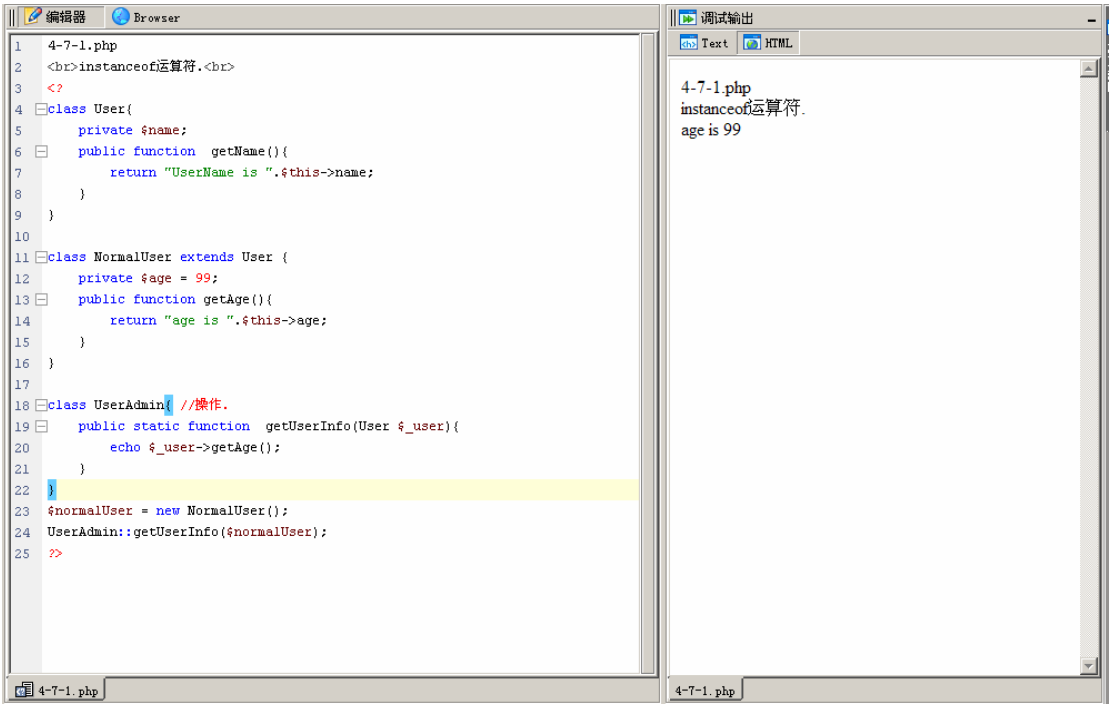
4.7 instanceof 运算符

在 PHP5 中，通过方法传递变量的类型有不确定性。
于是我们很难判断，一些操作是否可以运行。

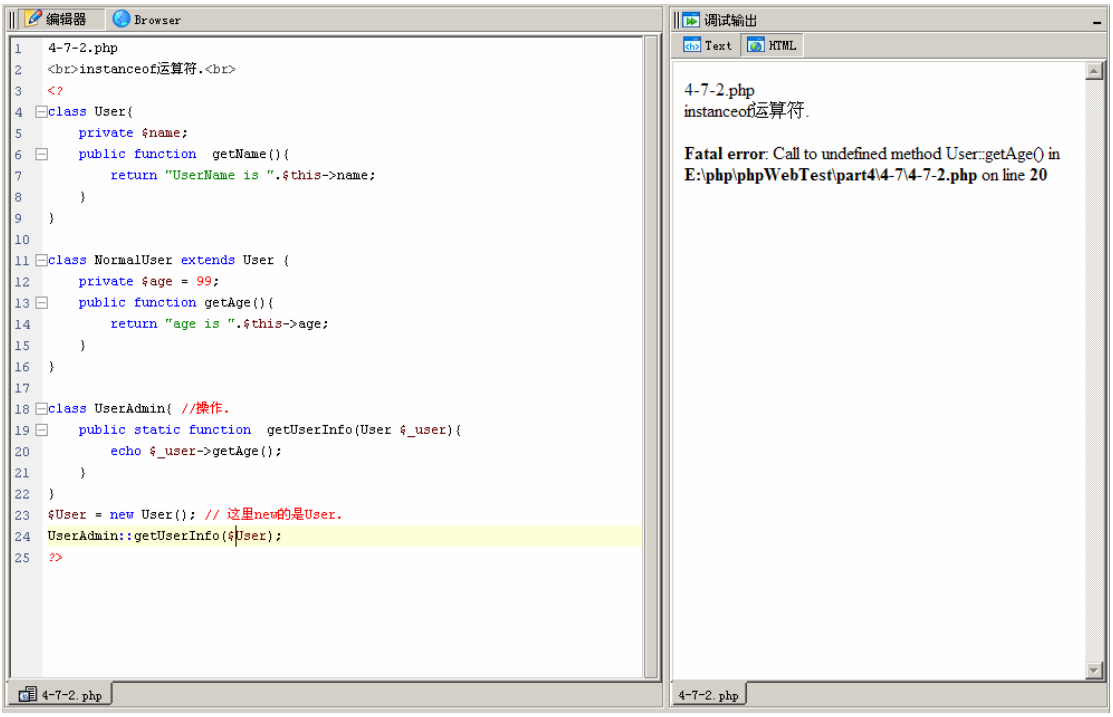
- 使用 instanceof 运算符，可以判断当前实例是否可以有这样的一个形态。
- 当前实例使用 instanceof 与 当前类，父类（向上无限追溯），已经实现的接口比较时，返回真。
- 代码格式　实例名 instanceof 类名

4.7.1 instanceof 运算符的运用

如下例子可以运行。
例 4-7-1.php



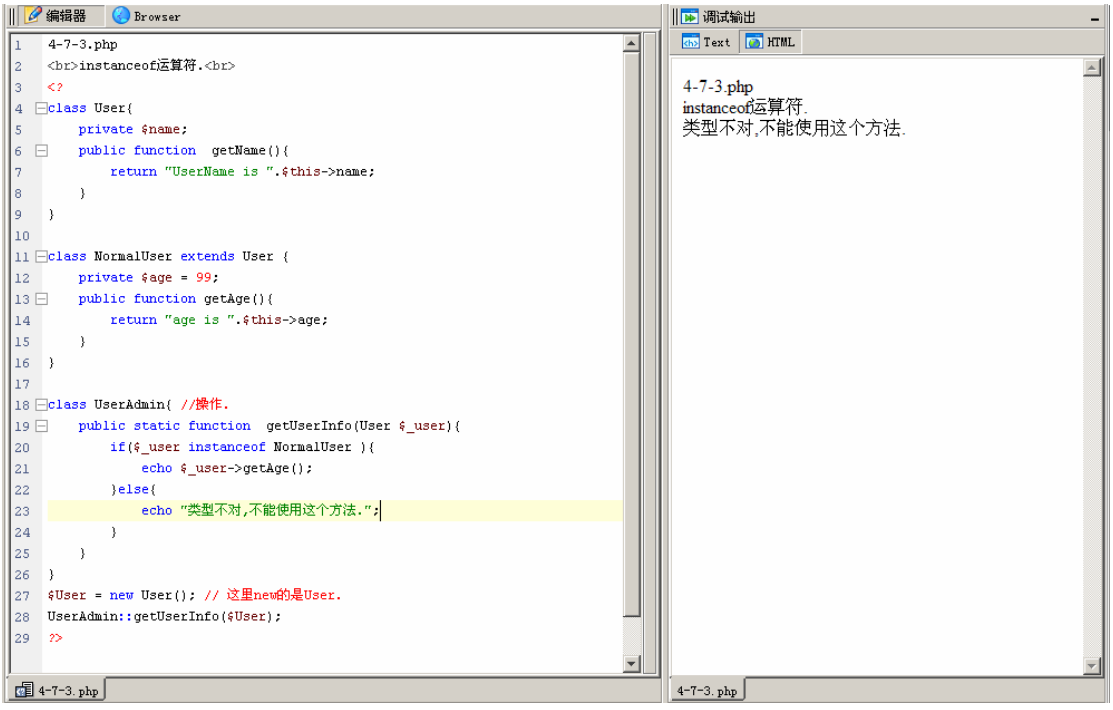
在 User 类中因为没有这个方法而报错，
例 4-7-2.php 就会出错。



4.7.2 使用 instatnceof 运算符保障代码安全

使用 instatnceof 运算符，在操作前先进行类型判断。以保障代码的安全性。

例：4-7-3.php

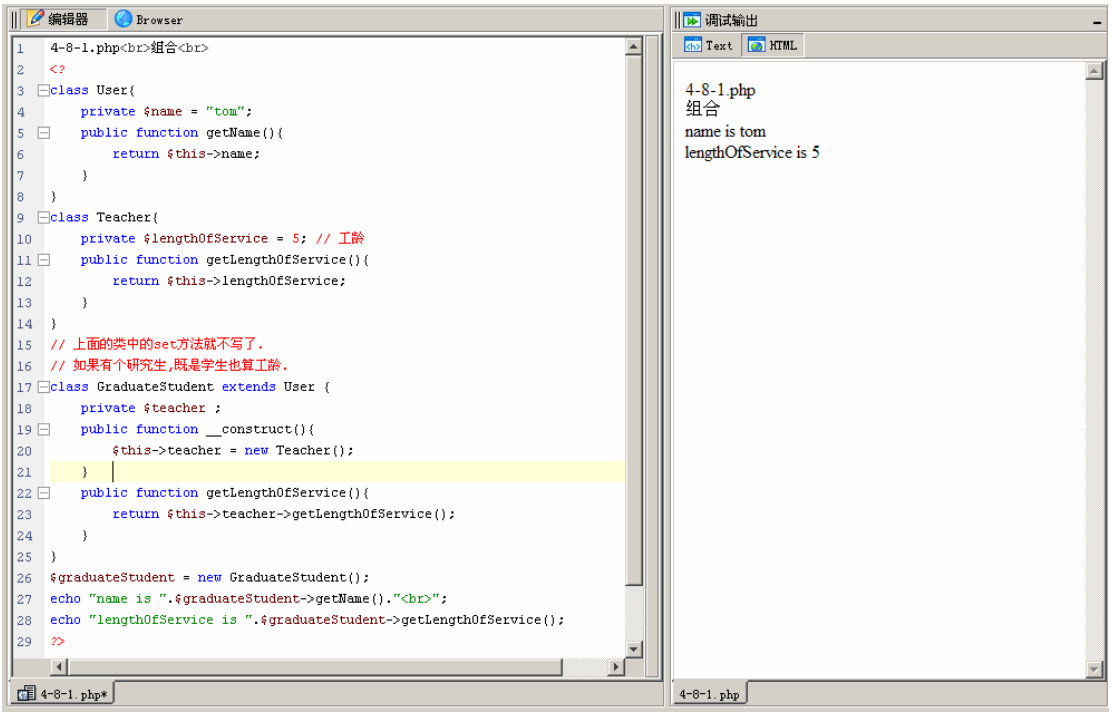


4.8 使用接口与组合模拟多继承

4.8.1 通过组合模拟多重继承

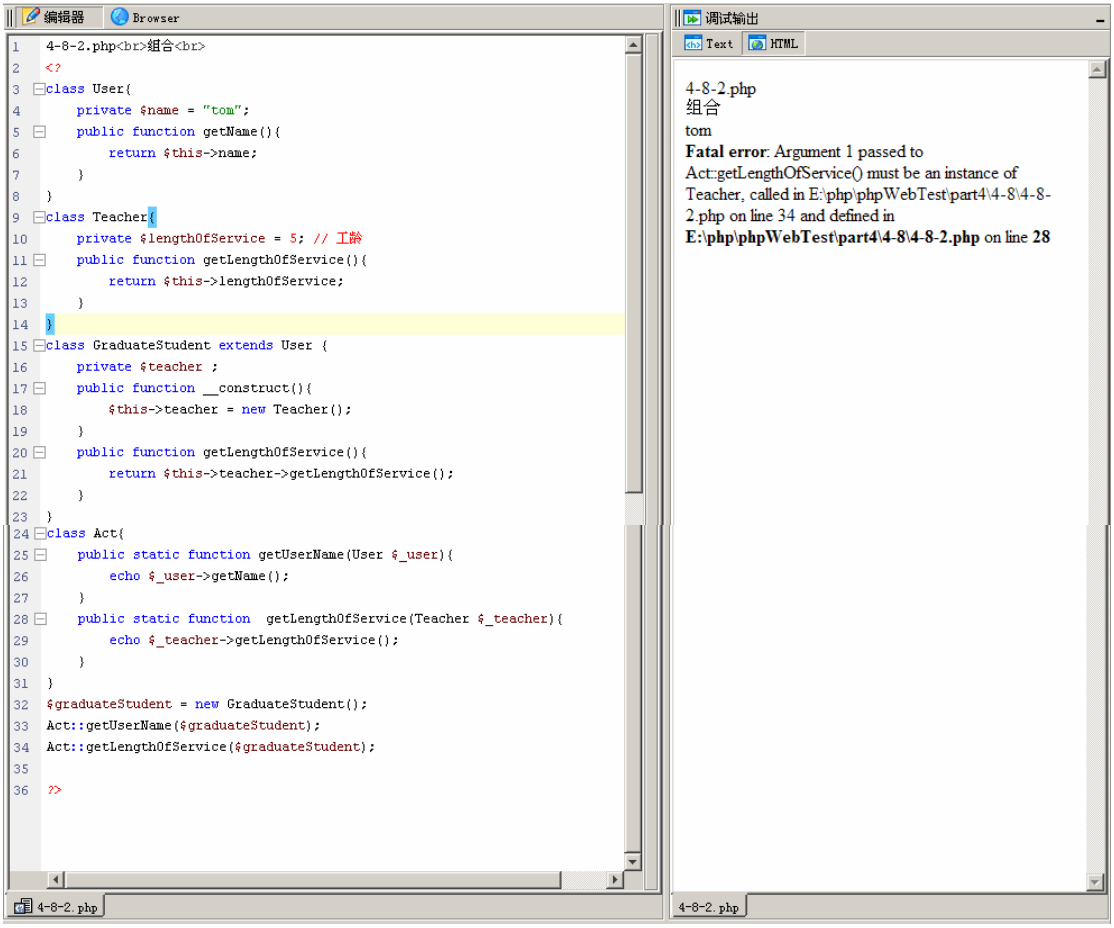
在 PHP 中不支持多重继承，如果我们向使用多个类的方法而实现代码重用有什么办法么？那就是组合。在一个类中去将另外一个类设置成属性。

下面的例子，模拟了多重继承。
例 4-8-1.php：



4.8.2 不完全的多重继承

上面的例子是多重继承么？
我们继续运行下面的代码。

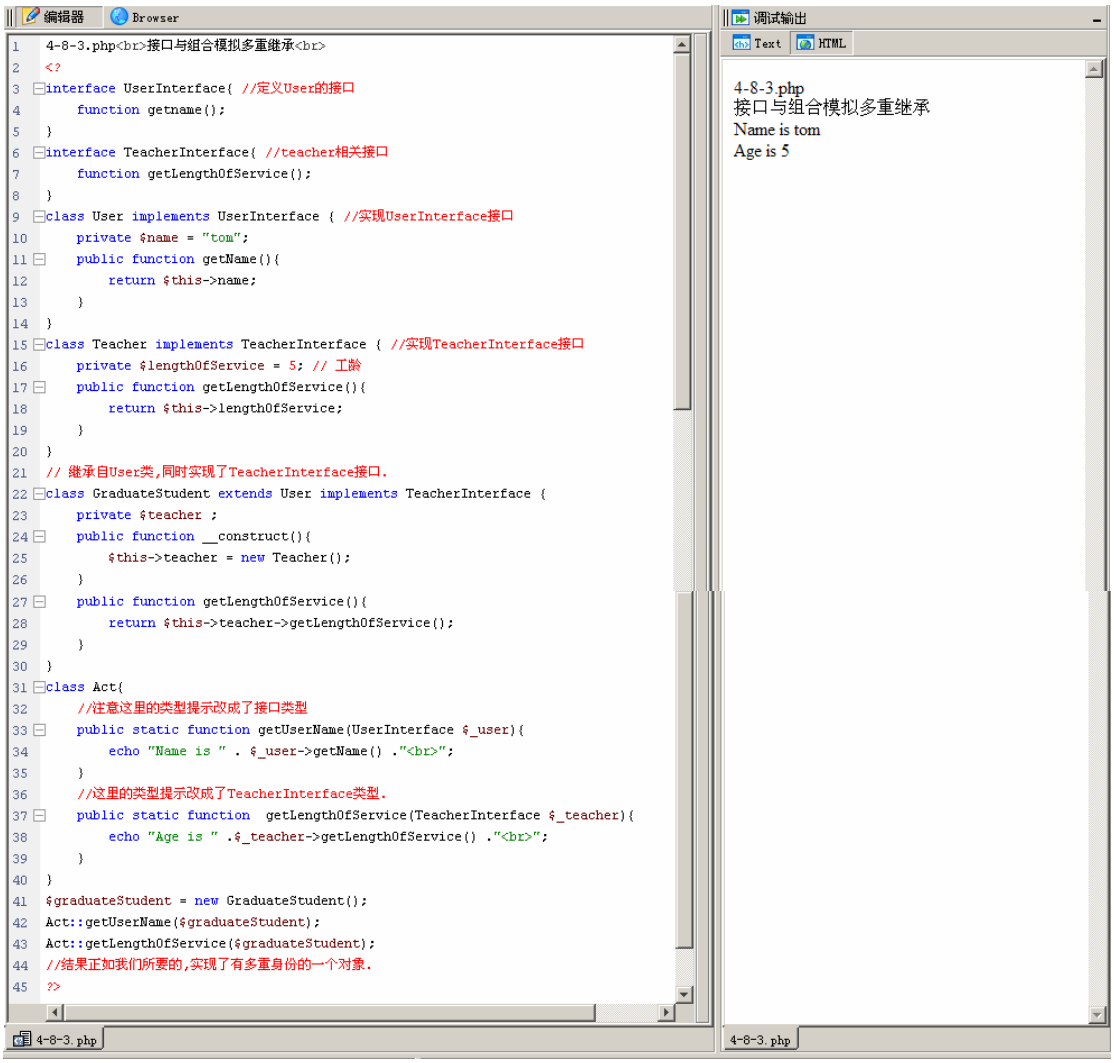


我们并不能将$graduateStudent 代表的实例当作 Teacher 类型传入。
这样的多重继承是失败的，没有多重身份，怎么能叫多重继承呢？

4.8.3 使用接口实现多重继承

通过把类的特征抽象为接口，并通过实现接口的方式让对象有多重身份，自然就可以完全模拟多重继承了。

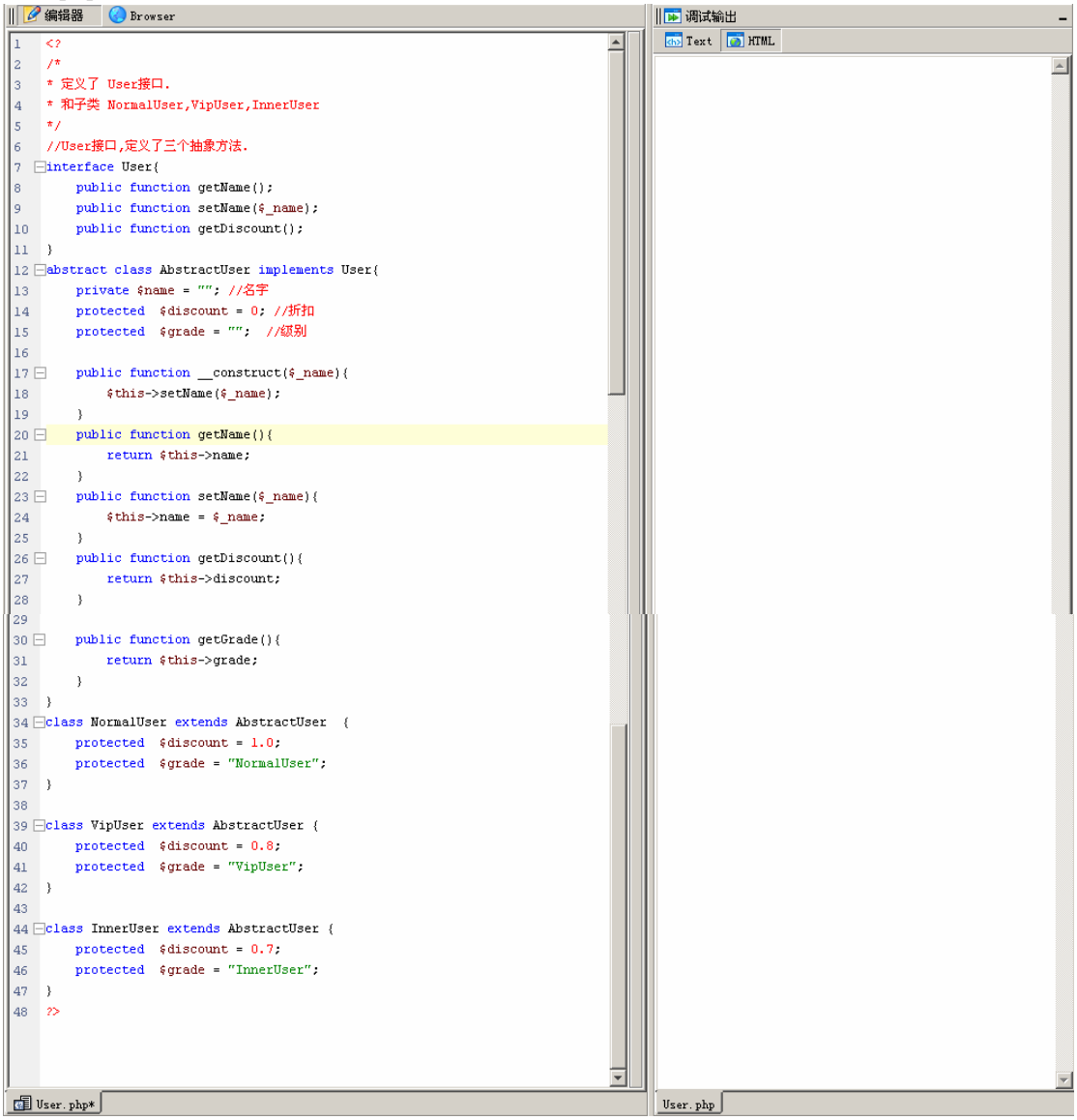
例：4-8-3.php



4.9 接口实例

写一个概念性的例子。
我们设计一个在线销售系统，用户部分设计如下：
将用户分为，NormalUser ,VipUser,InnerUser 三种。
要求根据用户的不同折扣计算用户购买产品的价格。
并要求为以后扩展和维护预留空间。

用户部分先声明了一个接口 User，用户都是 User 的实现。
User.php




```php
<?
/*
 * 定义了 User接口.
 * 和子类 NormalUser,VipUser,InnerUser
 */
//User接口,定义了三个抽象方法.
interface User{
    public function getName();
    public function setName($_name);
    public function getDiscount();
}
abstract class AbstractUser implements User{
    private $name = ""; //名字
    protected  $discount = 0; //折扣
    protected  $grade = "";  //级别

    public function __construct($_name){
        $this->setName($_name);
    }
    public function getName(){
        return $this->name;
    }
    public function setName($_name){
        $this->name = $_name;
    }
    public function getDiscount(){
        return $this->discount;
    }

    public function getGrade(){
        return $this->grade;
    }
}
class NormalUser extends AbstractUser  {
    protected  $discount = 1.0;
    protected  $grade = "NormalUser";
}

class VipUser extends AbstractUser {
    protected  $discount = 0.8;
    protected  $grade = "VipUser";
}

class InnerUser extends AbstractUser {
    protected  $discount = 0.7;
    protected  $grade = "InnerUser";
}
?>
```

关于产品，我们进行了如下设计。
声明一个接口 Product，然后从 Product 继承下 Book 接口。
在线销售的图书最后是实现了 Book 接口的 BookOnline 类。
Product.php

编辑器 Browser

```php
<?
/*与产品相关的类放.*/
Interface Product{ //定义产品接口
    public function getProductName();
    public function getProductPrice();
}

interface Book extends Product { // book是产品的一个分类
    public function getAuthor();
}

class BookOnline implements Book{ // 定义book类.
    private $productName;  // 产品名
    private $productPrice; // 产品价格
    private $author;  //作者

    public function __construct($_bookName){
        $this->productName = $_bookName;
        //这里放置相关初始化的代码.
        //与数据库关联的代码.
    }

    public function getProductName(){
        return $this->productName;
    }

    public function getProductPrice(){
        //这里从数据库读取价格.
        //假设价格是 100元.
        $this->productPrice = 100;
        return $this->productPrice;
    }

    public function getAuthor(){
        //从数据库里面取值.
        return $this->author;
    }
}


?>
```

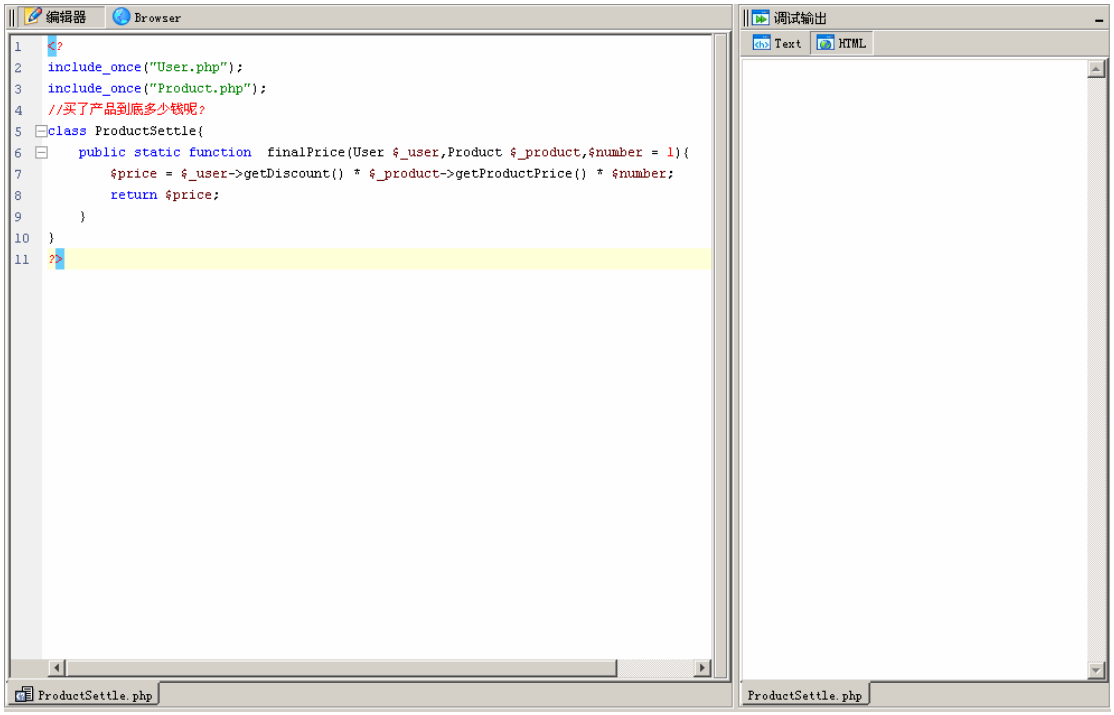
Product.php

调试输出

Text HTML

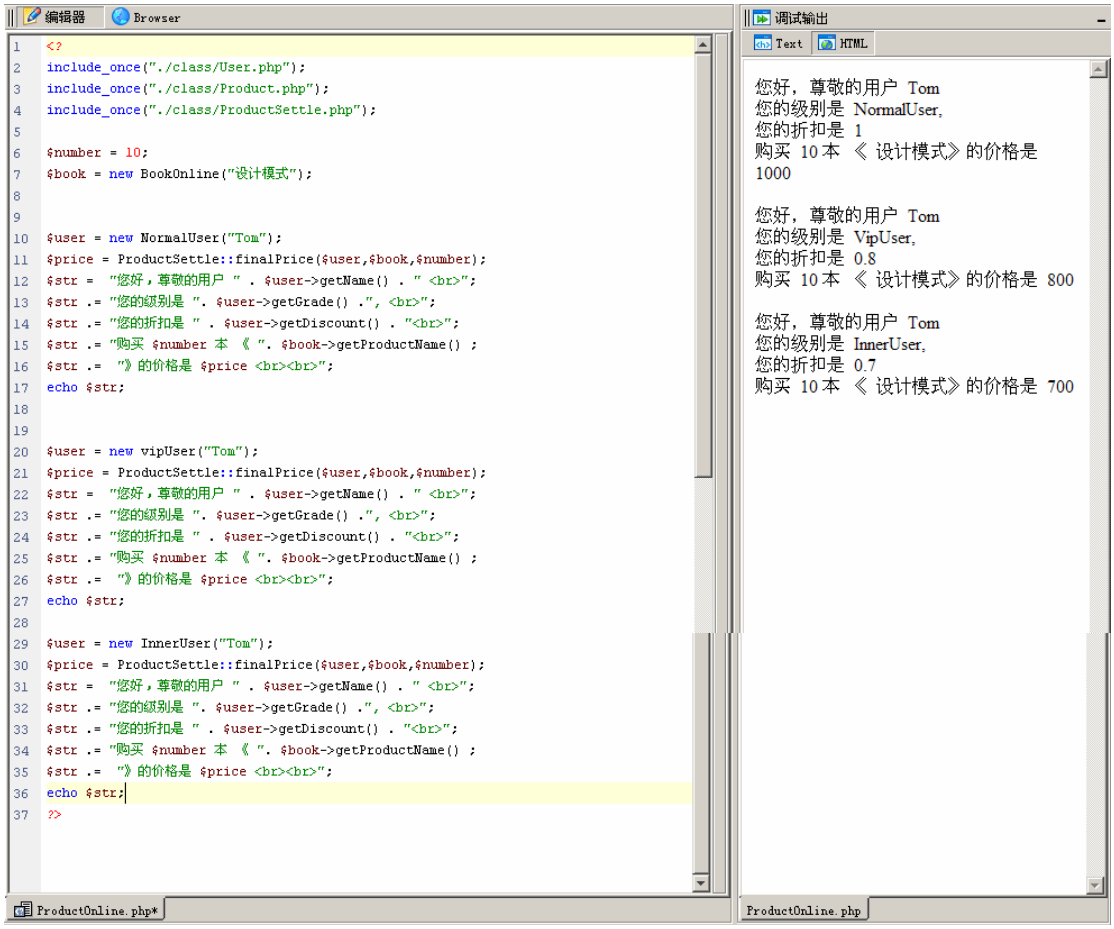
Product.php

关于结算，我们使用了独立的结算类，使用静态方法做计算。
产品结算。注意参数类型。




```php
<?
include_once("User.php");
include_once("Product.php");
//买了产品到底多少钱呢？
class ProductSettle{
    public static function  finalPrice(User $_user,Product $_product,$number = 1){
        $price = $_user->getDiscount() * $_product->getProductPrice() * $number;
        return $price;
    }
}
?>
```

下面的例子是实现。大家可以自己分析下。



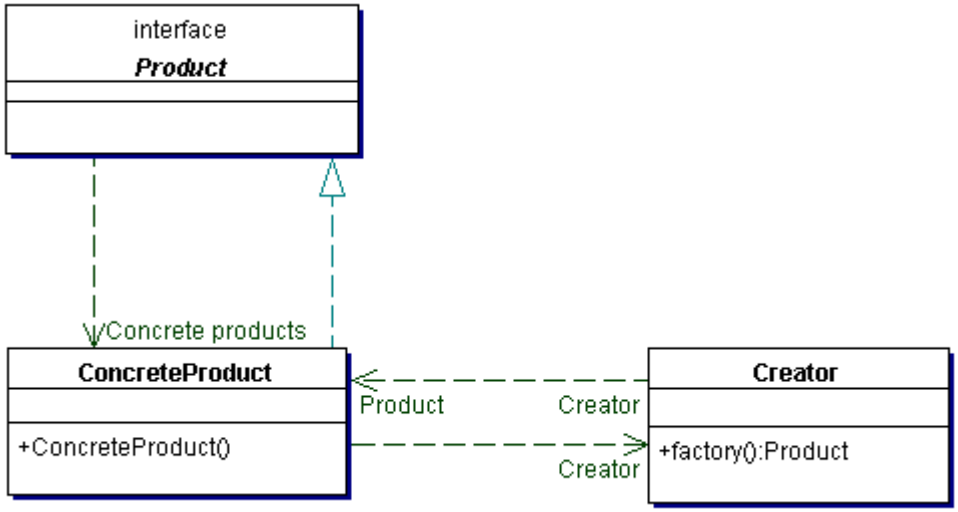

```php
<?
include_once("./class/User.php");
include_once("./class/Product.php");
include_once("./class/ProductSettle.php");

$number = 10;
$book = new BookOnline("设计模式");


$user = new NormalUser("Tom");
$price = ProductSettle::finalPrice($user,$book,$number);
$str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
$str .= "您的级别是 ". $user->getGrade() .", <br>";
$str .= "您的折扣是 " . $user->getDiscount() . "<br>";
$str .= "购买 $number 本 《 ". $book->getProductName() ;
$str .=  "》的价格是 $price <br><br>";
echo $str;


$user = new vipUser("Tom");
$price = ProductSettle::finalPrice($user,$book,$number);
$str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
$str .= "您的级别是 ". $user->getGrade() .", <br>";
$str .= "您的折扣是 " . $user->getDiscount() . "<br>";
$str .= "购买 $number 本 《 ". $book->getProductName() ;
$str .=  "》的价格是 $price <br><br>";
echo $str;

$user = new InnerUser("Tom");
$price = ProductSettle::finalPrice($user,$book,$number);
$str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
$str .= "您的级别是 ". $user->getGrade() .", <br>";
$str .= "您的折扣是 " . $user->getDiscount() . "<br>";
$str .= "购买 $number 本 《 ". $book->getProductName() ;
$str .=  "》的价格是 $price <br><br>";
echo $str;
?>
```

4.10 简单工厂模式

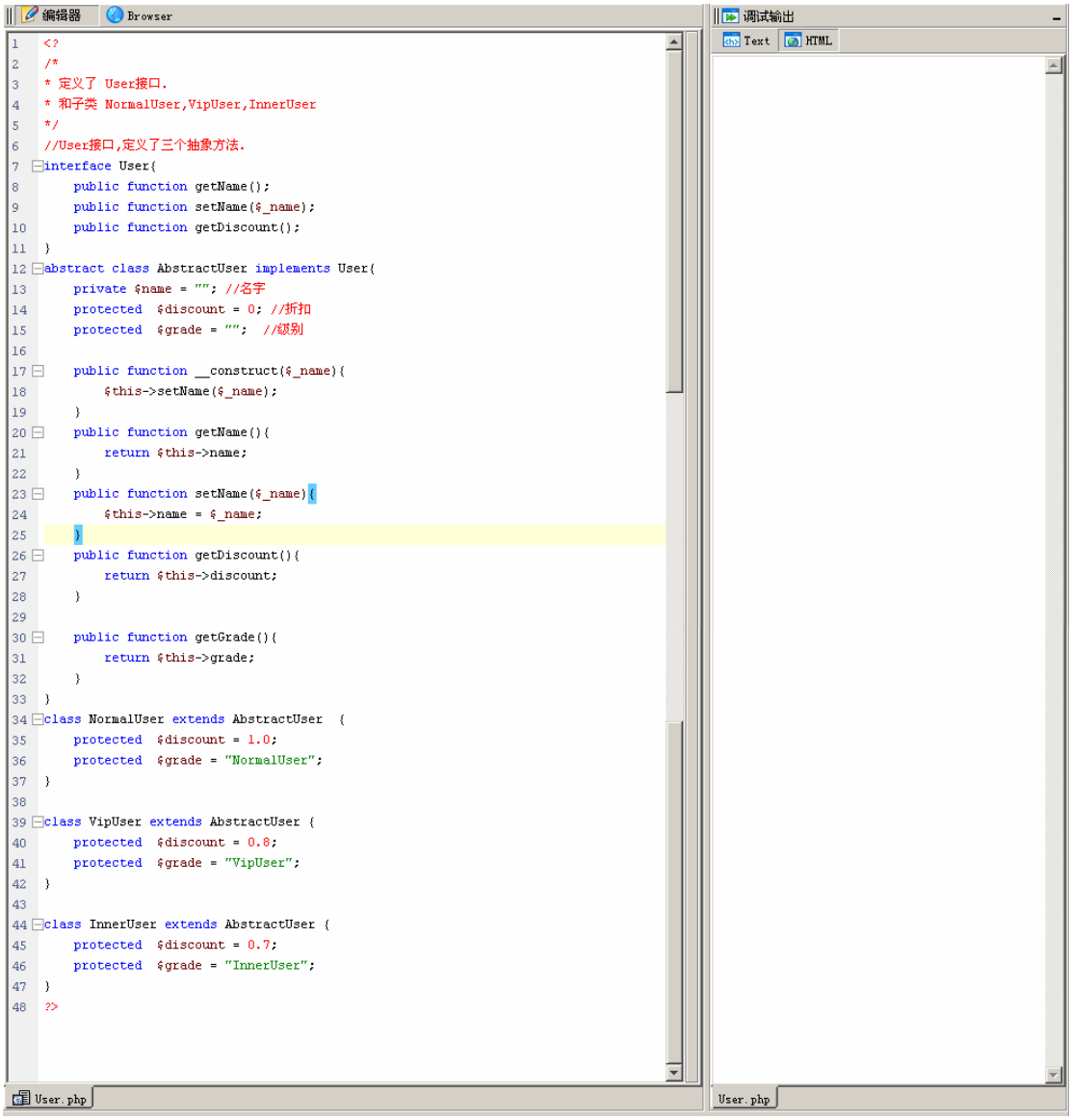
简单工厂模式是一种比较简单的设计模式，也是我们常用的设计模式。
使用简单工厂模式，能够根据不同的参数，使用不同的工厂，创建不同的对象。

简单工厂模式是类的创建模式,又叫做静态工厂方法(Static Factory Method)模式。

简单工厂模式是由一个工厂类根据传入的参量决定创建出哪一种产品类的实例。



说起来比较复杂，看一下就很简单了。
上一节的例子，在这一节我们做一些修改，我们用简单工厂模式建立。
大家注意比较改变前后代码的变化和使用。
User.php

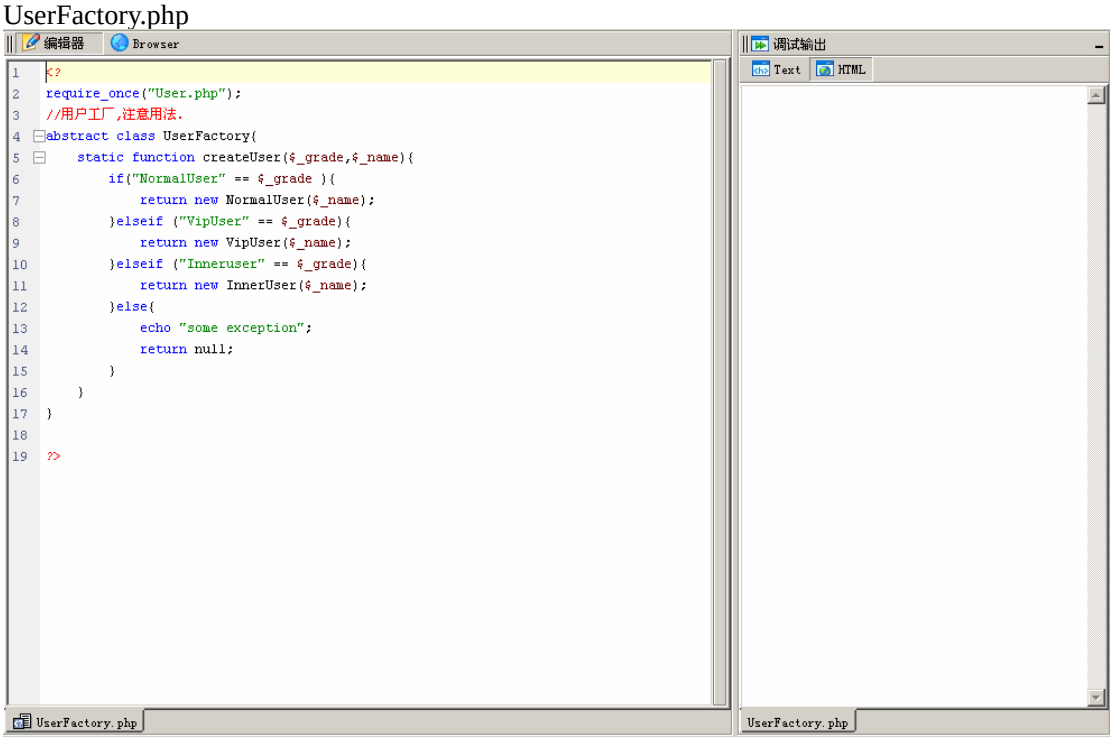



```php
<?
/*
* 定义了 User接口.
* 和子类 NormalUser,VipUser,InnerUser
*/
//User接口,定义了三个抽象方法.
interface User{
    public function getName();
    public function setName($_name);
    public function getDiscount();
}
abstract class AbstractUser implements User{
    private $name = ""; //名字
    protected  $discount = 0; //折扣
    protected  $grade = "";  //级别

    public function __construct($_name){
        $this->setName($_name);
    }
    public function getName(){
        return $this->name;
    }
    public function setName($_name){
        $this->name = $_name;
    }
    public function getDiscount(){
        return $this->discount;
    }

    public function getGrade(){
        return $this->grade;
    }
}
class NormalUser extends AbstractUser  {
    protected  $discount = 1.0;
    protected  $grade = "NormalUser";
}

class VipUser extends AbstractUser {
    protected  $discount = 0.8;
    protected  $grade = "VipUser";
}

class InnerUser extends AbstractUser {
    protected  $discount = 0.7;
    protected  $grade = "InnerUser";
}
?>
```

用户工厂

用户工厂根据传递来的参数，创建不同的对象。但他们拥有共同的形态 User

UserFactory.php




```php
<?
require_once("User.php");
//用户工厂,注意用法.
abstract class UserFactory{
    static function createUser($_grade,$_name){
        if("NormalUser" == $_grade ){
            return new NormalUser($_name);
        }elseif ("VipUser" == $_grade){
            return new VipUser($_name);
        }elseif ("Inneruser" == $_grade){
            return new InnerUser($_name);
        }else{
            echo "some exception";
            return null;
        }
    }
}

?>
```

产品的定义在这个类中实现

Product.php

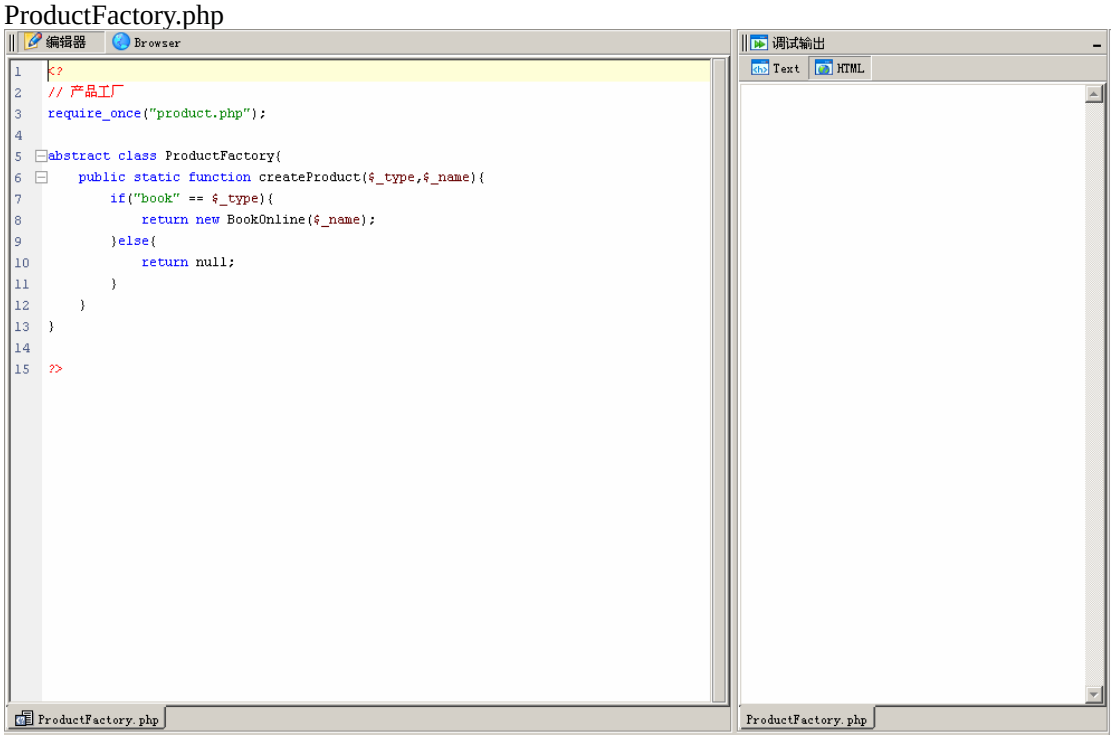
编辑器 Browser

```php
<?
/*与产品相关的类放.*/
Interface Product{ //定义产品接口
    public function getProductName();
    public function getProductPrice();
}

interface Book extends Product { // book是产品的一个分类
    public function getAuthor();
}

class BookOnline implements Book{ // 定义book类.
    private $productName;  // 产品名
    private $productPrice; // 产品价格
    private $author;  //作者

    public function __construct($_bookName){
        $this->productName = $_bookName;
        //这里放置相关初始化的代码.
        //与数据库关联的代码.
    }

    public function getProductName(){
        return $this->productName;
    }

    public function getProductPrice(){
        //这里从数据库读取价格.
        //假设价格是 100元.
        $this->productPrice = 100;
        return $this->productPrice;
    }

    public function getAuthor(){
        //从数据库里面取值.
        return $this->author;
    }
}


?>
```

调试输出 Text HTML

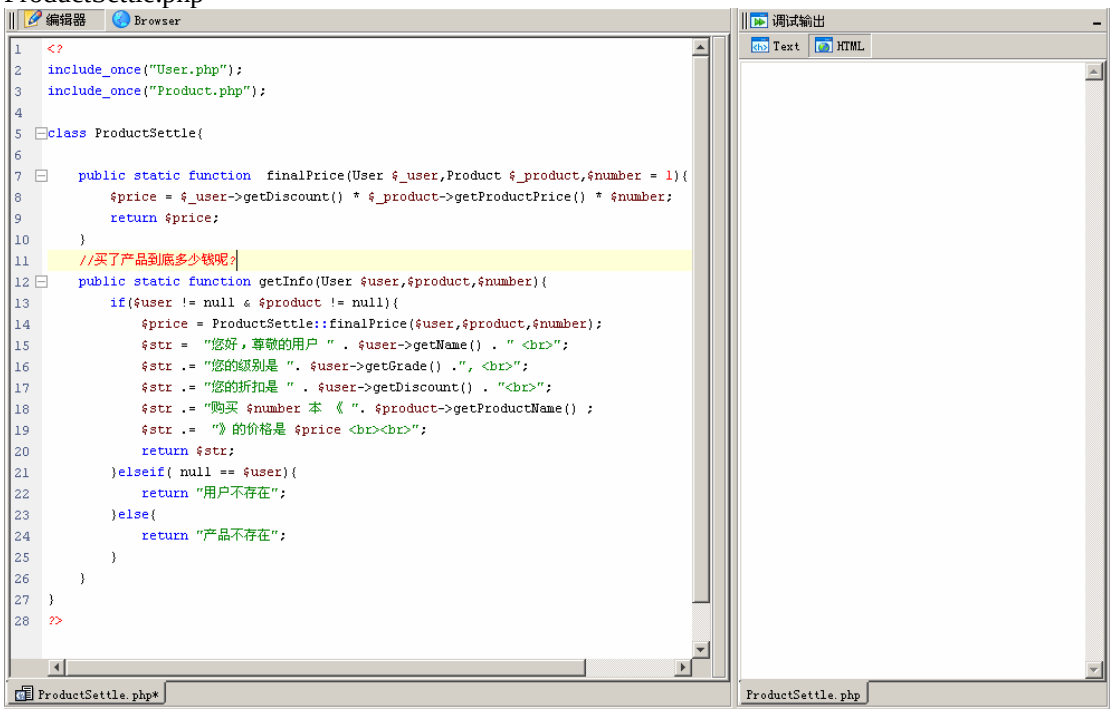
Product.php

产品工厂，根据参数创建不同的对象。
现在的工厂只能创建 book 的对象，以后有了新产品，来这里加上就好了。
ProductFactory.php



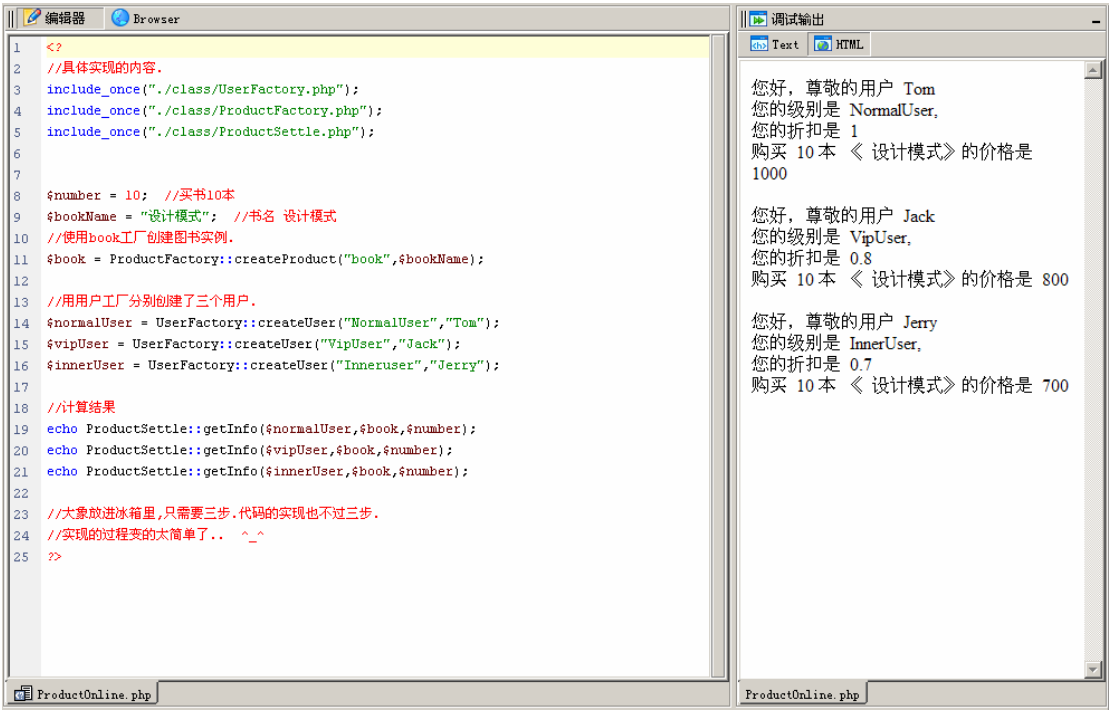
```php
<?
// 产品工厂
require_once("product.php");

abstract class ProductFactory{
    public static function createProduct($_type,$_name){
        if("book" == $_type){
            return new BookOnline($_name);
        }else{
            return null;
        }
    }
}

?>
```

定义一个操作类，里面的操作关联性不强，就用静态方法方便使用。
ProductSettle.php



```php
<?
include_once("User.php");
include_once("Product.php");

class ProductSettle{

    public static function  finalPrice(User $_user,Product $_product,$number = 1){
        $price = $_user->getDiscount() * $_product->getProductPrice() * $number;
        return $price;
    }
    //买了产品到底多少钱呢?
    public static function getInfo(User $user,$product,$number){
        if($user != null & $product != null){
            $price = ProductSettle::finalPrice($user,$product,$number);
            $str =  "您好，尊敬的用户 " . $user->getName() . " <br>";
            $str .= "您的级别是 ". $user->getGrade() .", <br>";
            $str .= "您的折扣是 " . $user->getDiscount() . "<br>";
            $str .= "购买 $number 本 《 ". $product->getProductName() ;
            $str .=  "》的价格是 $price <br><br>";
            return $str;
        }elseif( null == $user){
            return "用户不存在";
        }else{
            return "产品不存在";
        }
    }
}
?>
```

最后的实现。
上面所有的内容都定义完毕了，使用起来很方便。
只需要三步就可以了。



小结

这一章介绍了接口、多态和简单工厂模式。
希望大家对面向对象有了更多的了解。
有什么建议就到论坛给我留言好了。
http://www.phpchina.com/bbs/thread-10556-1-1.html

最近太忙了，有个孩子很累人啊。
这一章没有怎么太多的组织例子。对例子的描述也简单了些。希望大家包涵。
写到这里，对于面向对象的内容大体上应该有个认识了。
也应该从此开始，在写代码时候考虑如何面向对象了。
在 PHP 程序员中，这些概念少了些。而 Java 程序员，从任何一本书的第一节开始就是面向对象的。由此，很多偏见的观点只有 Java 和 .net 才能相提并论，PHP 总是和 Asp 放在一起。
也许在你了解了面向对象之后，可以看看 Java 的初级教程，会觉得一样简单。

下一章，将介绍 PHP5 中的异常处理。

刀客羽朋
于石家庄 2006-12-24