第三章 PHP5 面向对象高级类特性

这一周来正好白天没课，赶出来这章。

宝宝出生将近一个月，快 10 斤重了。
宝宝快张大，你想编程么？宝宝妈说，做这个行业太累了。

刀客羽朋
于石家庄 2006-11-23

目录

3.1 static 变量、方法

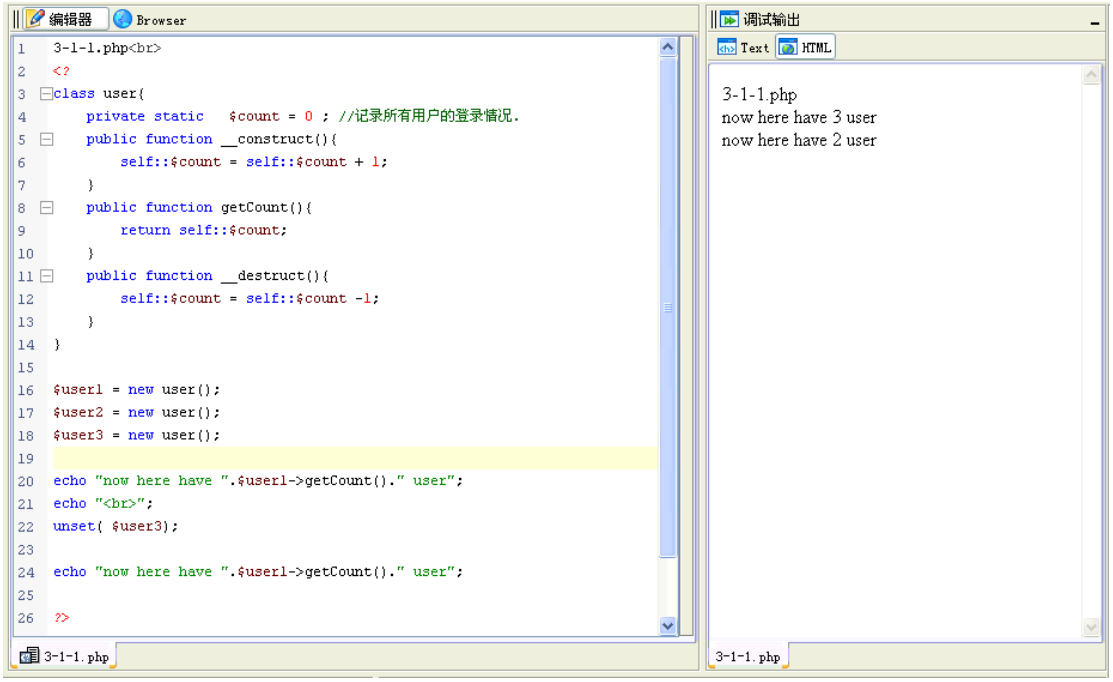
- static 关键字用来修饰属性、方法，称这些属性、方法为静态属性、静态方法。
- static 关键字声明一个属性或方法是和类相关的，而不是和类的某个特定的实例相关，因此，这类属性或方法也称为“类属性”或“类方法”
- 如果访问控制权限允许，可不必创建该类对象而直接使用类名加两个冒号“::”调用。

- **static** 关键字可以用来修饰变量、方法。
- 不经过实例化，就可以直接访问类中 static 的属性和 static 的方法。
- static 的属性和方法，只能访问 static 的属性和方法，不能类访问非静态的属性和方法。因为静态属性和方法被创建时，可能还没有任何这个类的实例可以被调用。
- static 的属性，在内存中只有一份，为所有的实例共用。
- 使用 self:: 关键字访问当前类的静态成员。

3.1.1 静态属性公用特性

- 一个类的所有实例，共用类中的静态属性。
- 也就是说，在内存中即使有多个实例，静态的属性也只有一份。

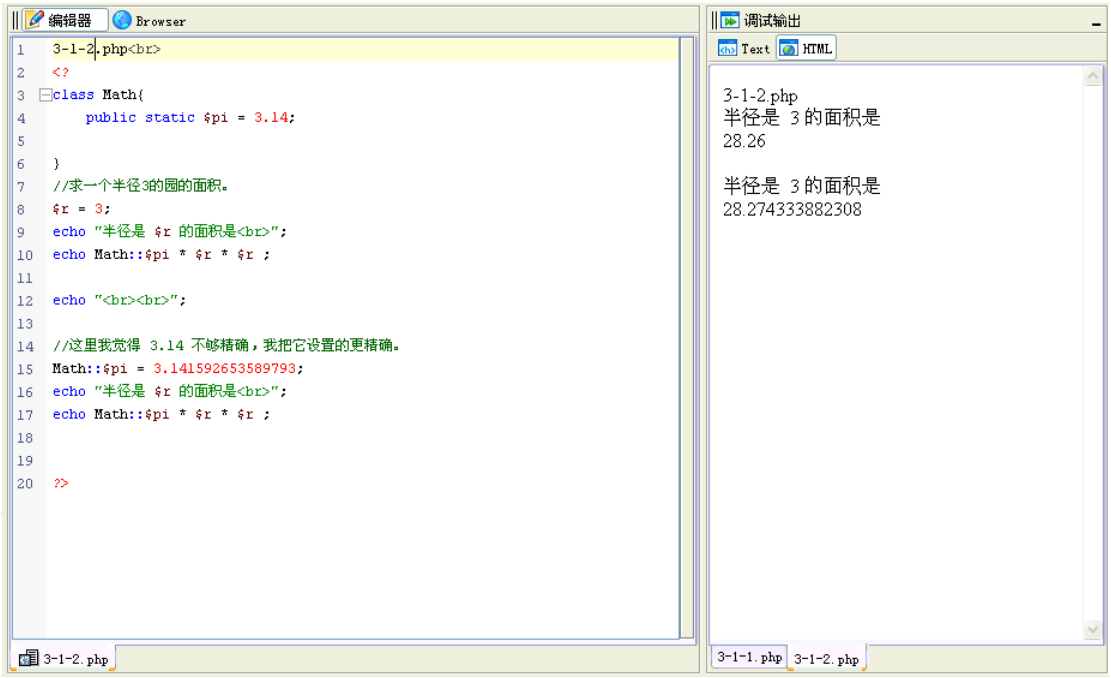
下面例子中的设置了一个计数器$count 属性，设置 private 和 static 修饰。
这样，外界并不能直接访问$count 属性。而程序运行的结果我们也看到多个实例在使用同一个静态的$count 属性。
例 3-1-1.php



3.1.2 静态属性直接调用

- 静态属性不需要实例化就可以直接使用，在类还没有创建时就可以直接使用。
- 使用的方式是 类名::静态属性名

例 3-1-2.php



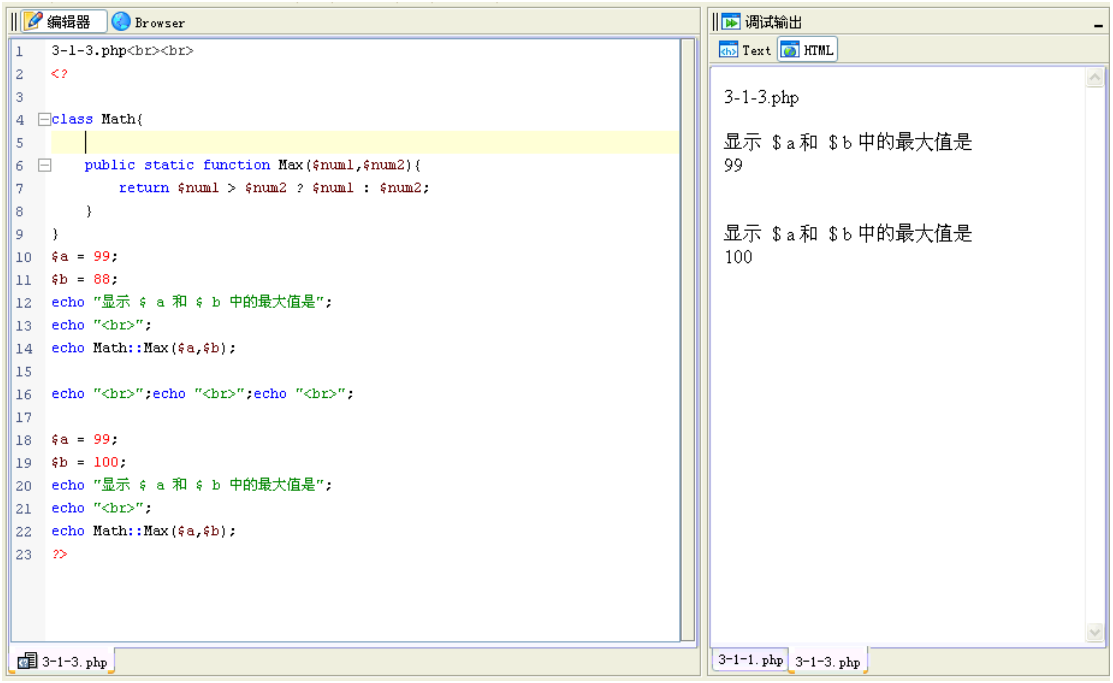
类没有创建，静态属性就可以直接使用。
那静态属性在什么时候在内存中被创建？
在 PHP 中没有看到相关的资料。引用 Java 中的概念，来解释应该也具有通用性。
静态属性和方法，在类被调用时创建。

类没有创建，静态属性就可以直接使用。 那静态属性在什么时候在内存中被创建？
在 PHP 中没有看到相关的资料。我们引用 Java 中的概念，来解释 PHP5 的静态修饰符，应该也具有通用性。
静态属性和方法，在类被调用时创建。
类被调用，是指类被创建或者类中的任何静态成员被调用。

3.1.3 静态方法

- 静态方法不需要所在类被实例化就可以直接使用。
- 使用的方式是 类名:: 静态方法名

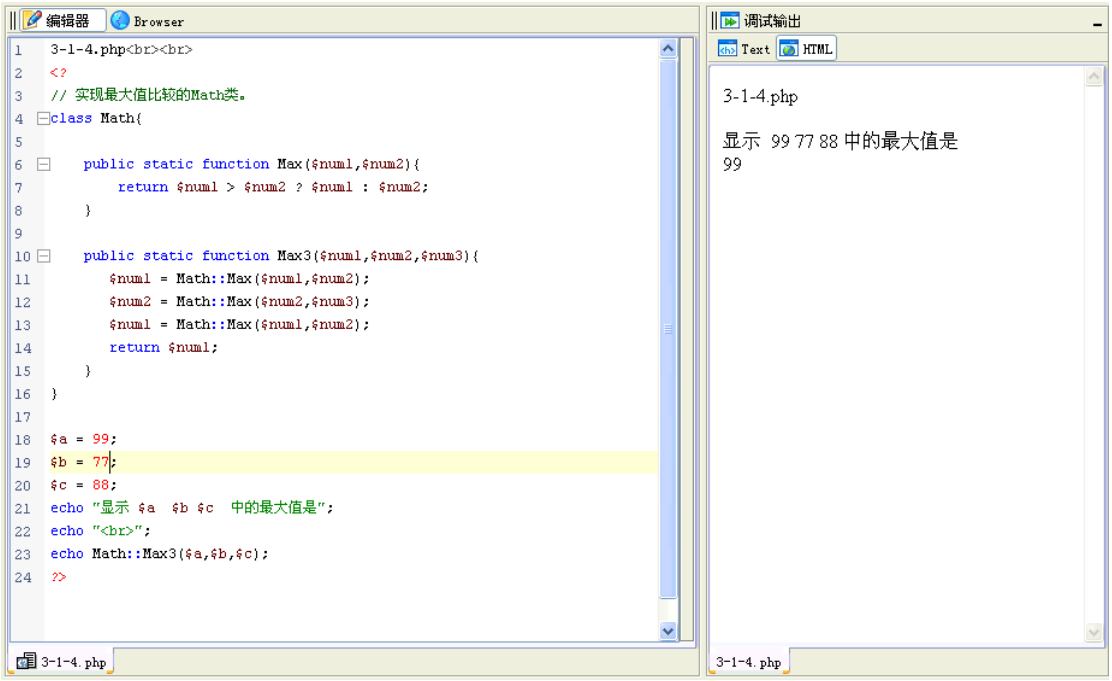
下面我们继续写这个 Math 类，用来进行数学计算。我们设计一个方法用来算出其中的最大值。
既然是数学运算，我们也没有必要去实例化这个类，如果这个方法可以拿过来就用就方便多了。
我们这只是为了演示 static 方法而设计的这个类。在 PHP 提供了 max() 函数比较数值。
例 3-2-3.php



3.1.4 静态方法如何调用静态方法

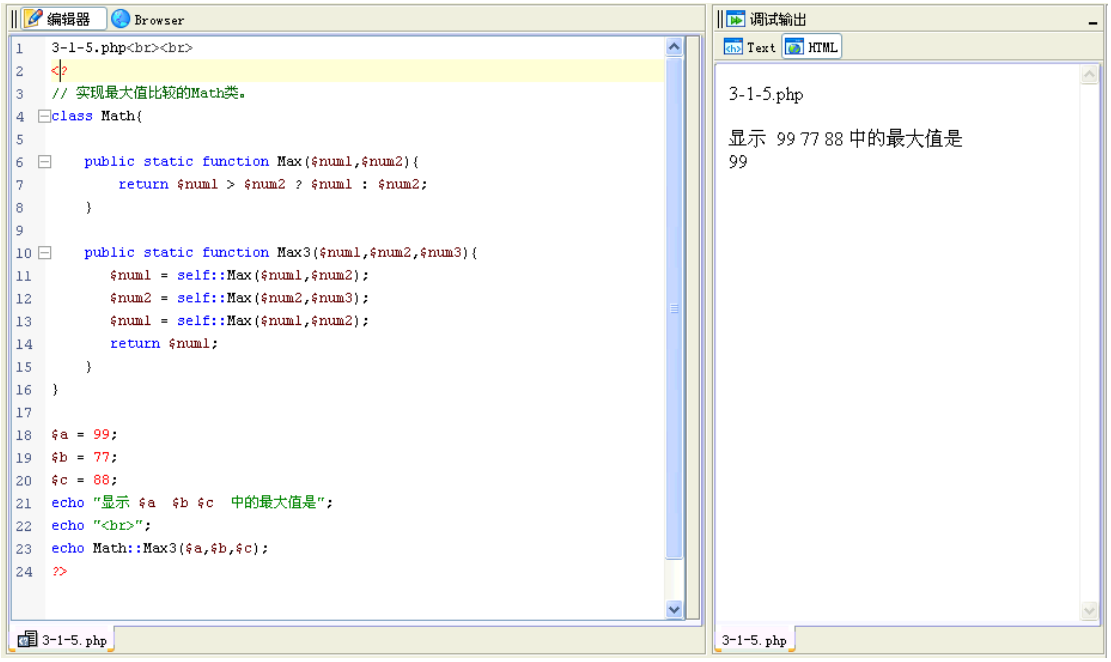
第一个例子，一个静态方法调用其它静态方法时，直接使用 类名

例 3-1-4.php



也可以使用 **self::** 调用当前类中的其它静态方法。（建议）

例 3-1-5.php

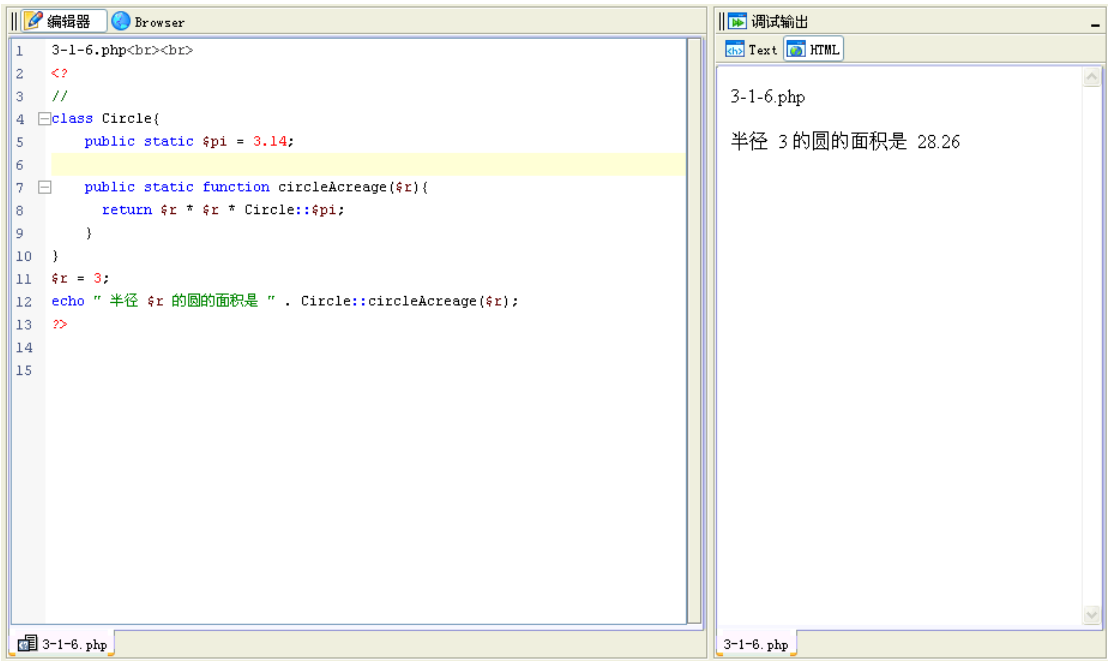


记住这个 self:: 它表示当前类的静态成员，与 $this 不同，$this 指当前对象。

3.1.5 静态方法调用静态属性

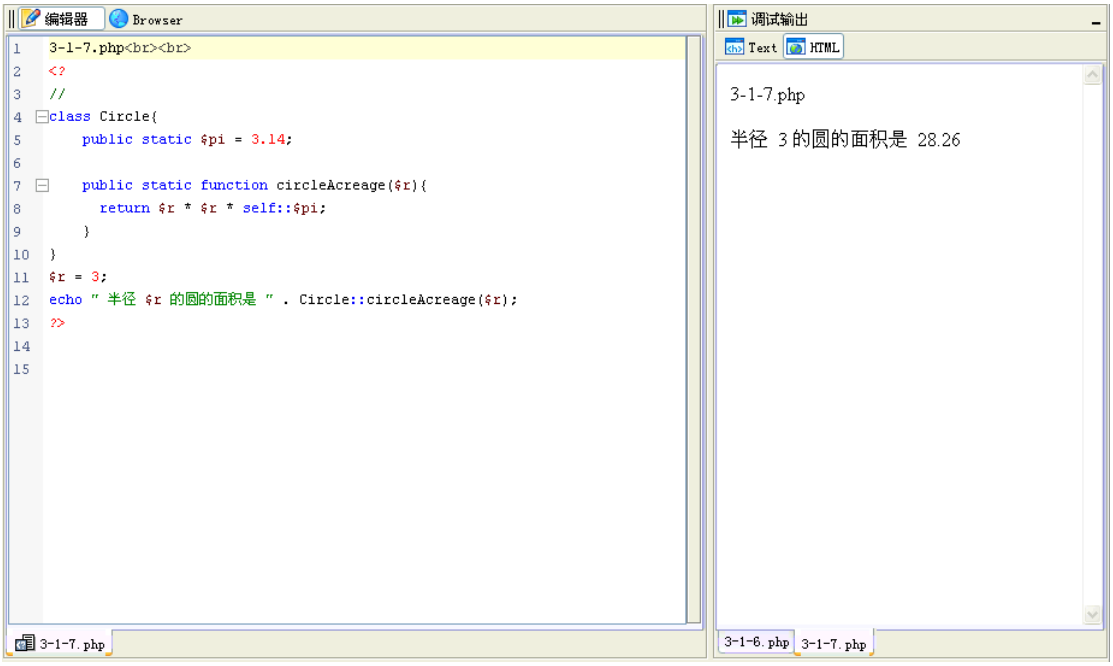
使用 **类名::静态属性名** 调用本类中的静态属性。

例 3-1-6.php



使用 **self::** 调用本类的静态属性。（建议）

例 3-1-7.php

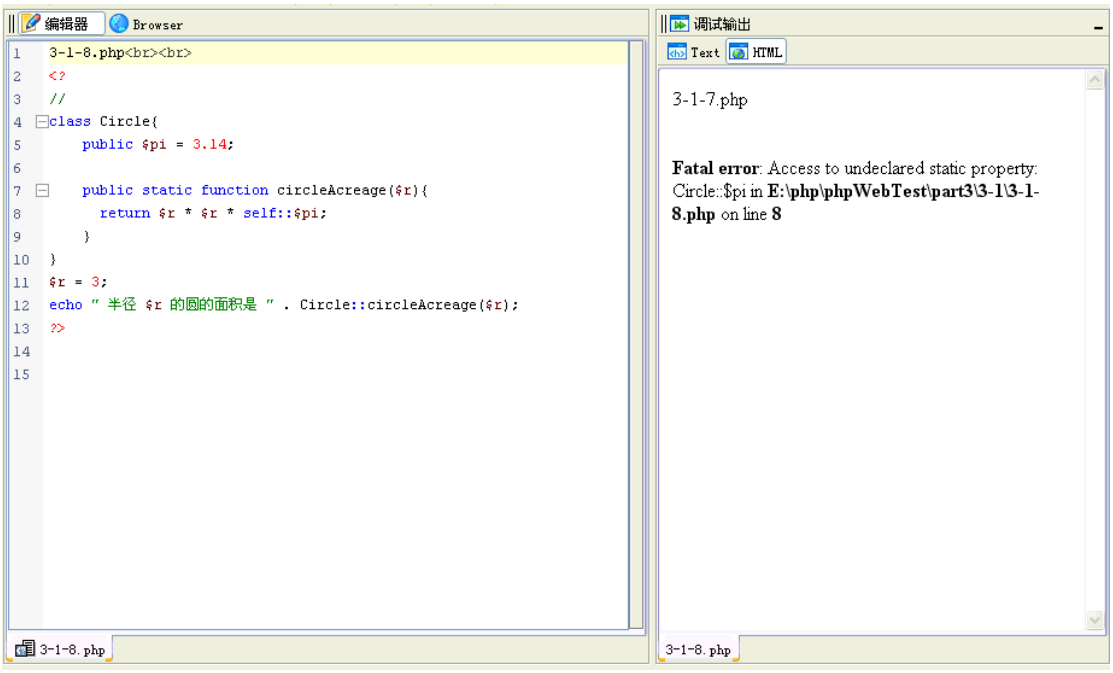


3.1.6 静态方法不能调用非静态属性

- 静态方法不能调用非静态的属性。

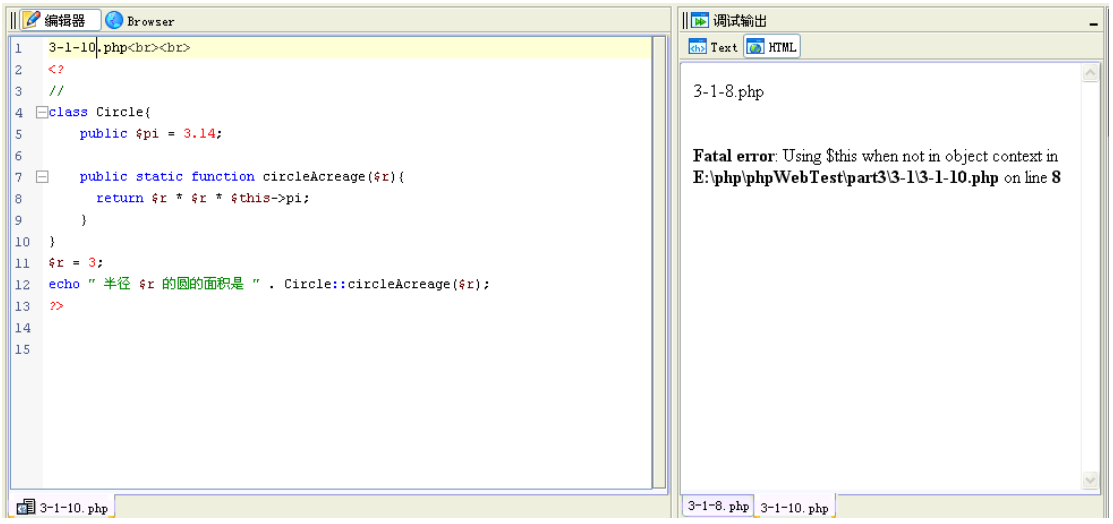
不能使用 self::调用非静态属性。

例 3-1-8.php



也不能使用 $this 获取非静态属性的值。

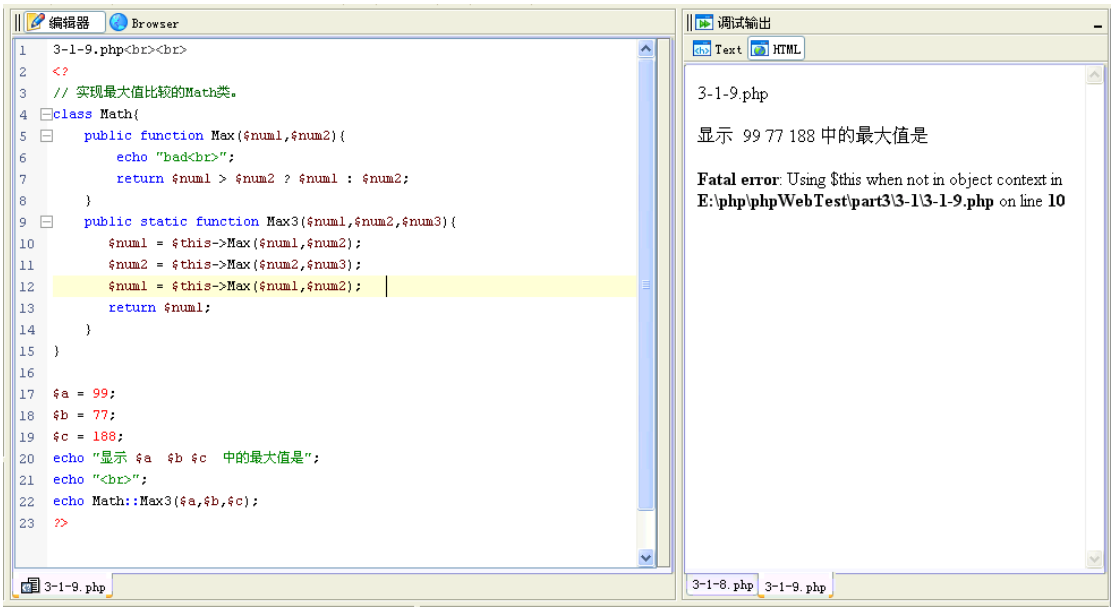
例 3-1-10



3.1.6 静态方法调用非静态方法

PHP5 中，在静态方法中不能使用 $this 标识调用非静态方法。

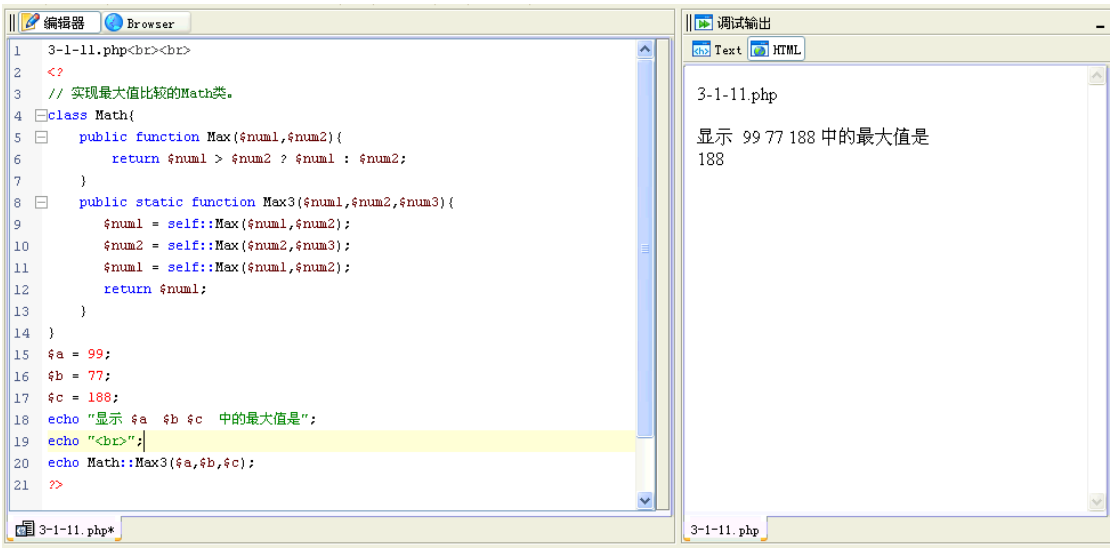
例 3-1-9



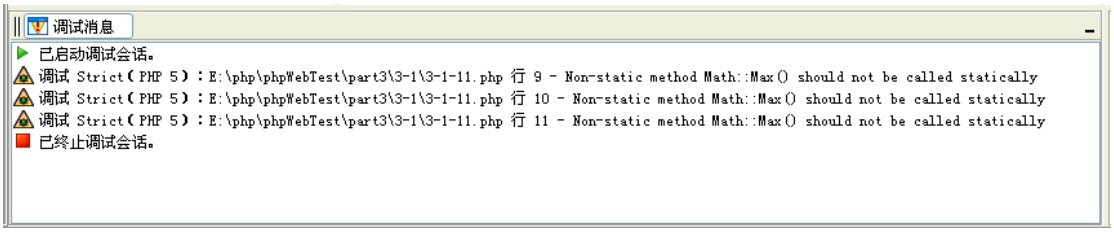
- 当一个类中有非静态方法被 self:: 调用时，系统会自动将这个方法转换为静态方法。

下面的代码被执行了，而且有结果。因为 Max 方法被系统转换为静态方法了。

例：3-1-11.php



在 Zend Studio 的调试部分显示出了这些错误的信息。

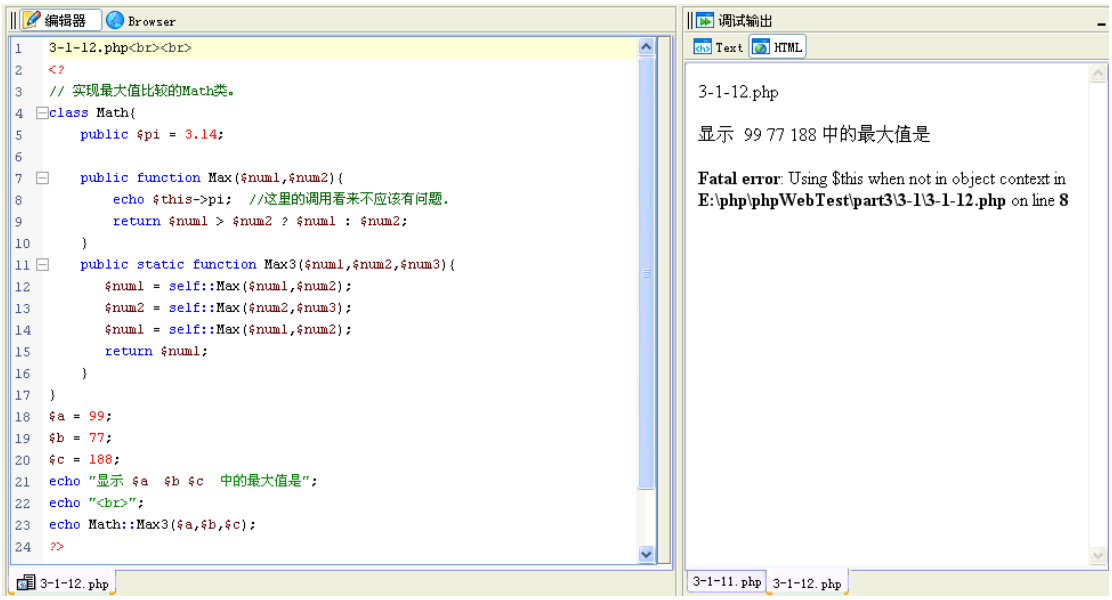


下面的例子中，我们让静态方法 Max3 用过 self::调用了非静态方法 Max，有让非静态方法 Max 通过$this 调用非静态属性 $pi 。

在运行是报出了错误，这个错误和前一个例子 3-1-9.php 一样，这次倒是非静态方法 Max 报出了静态方法调用非静态属性的错误。

这倒是证明了一点，在这里我们定义的非静态方法 Max 被系统自动转换成静态方法了。

例：3-1-12.php



3.1.7 设计模式之单件模式

单件模式要解决的问题就是“如何让这个类只有一个实例”。

我们的 web 应用中，大量使用了数据库连接，如果反复建立与数据库的连接必然消耗更多的系统资源。

我们如何解决这个问题，建立唯一的数据库连接是必要的方式。

我们又如何知道与这个数据库的连接是否已经建立？ 还是需要现在建立？

单件模式可以解决这个问题。

先假设我们需要一个类完成在内存中只有一份的功能，我们该如何做呢？
我们一步一步的使用前面学过的知识来写一个单件的例子。

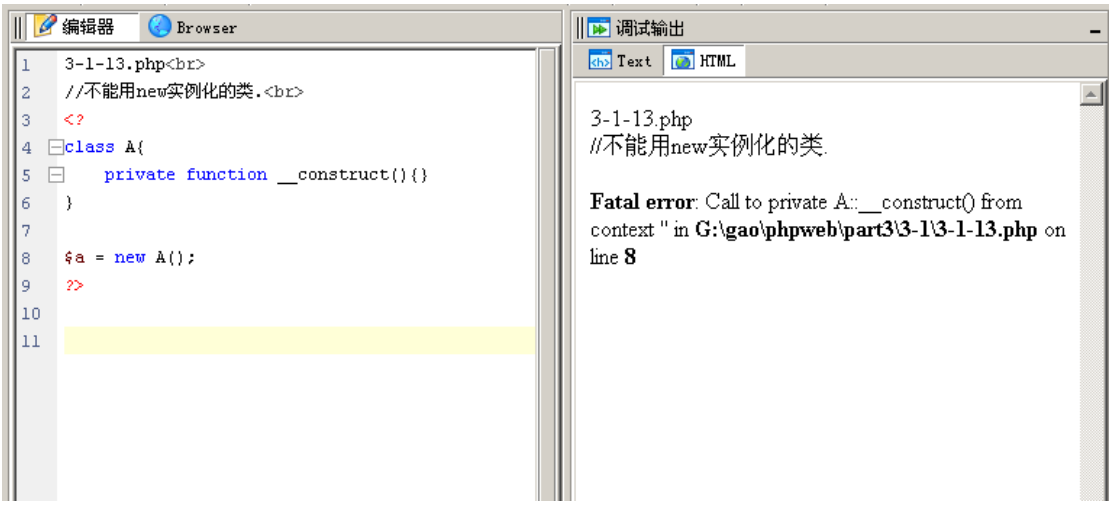
问题 1：

- 前面学过，每次用 new 类名 的方式，就可以创建一个对象。
- 我们必须禁止外部程序用 new 类名的方式来创建多个实例。

解决办法：

- 我们将构造函数设置成 private ，让构造函数只能在内部被调用，而外部不能调用。
- 这样，这个类就不能被外部用 new 的方式建立多个实例了。

例：3-1-13.php 不能被外部用 new 实例化的类。



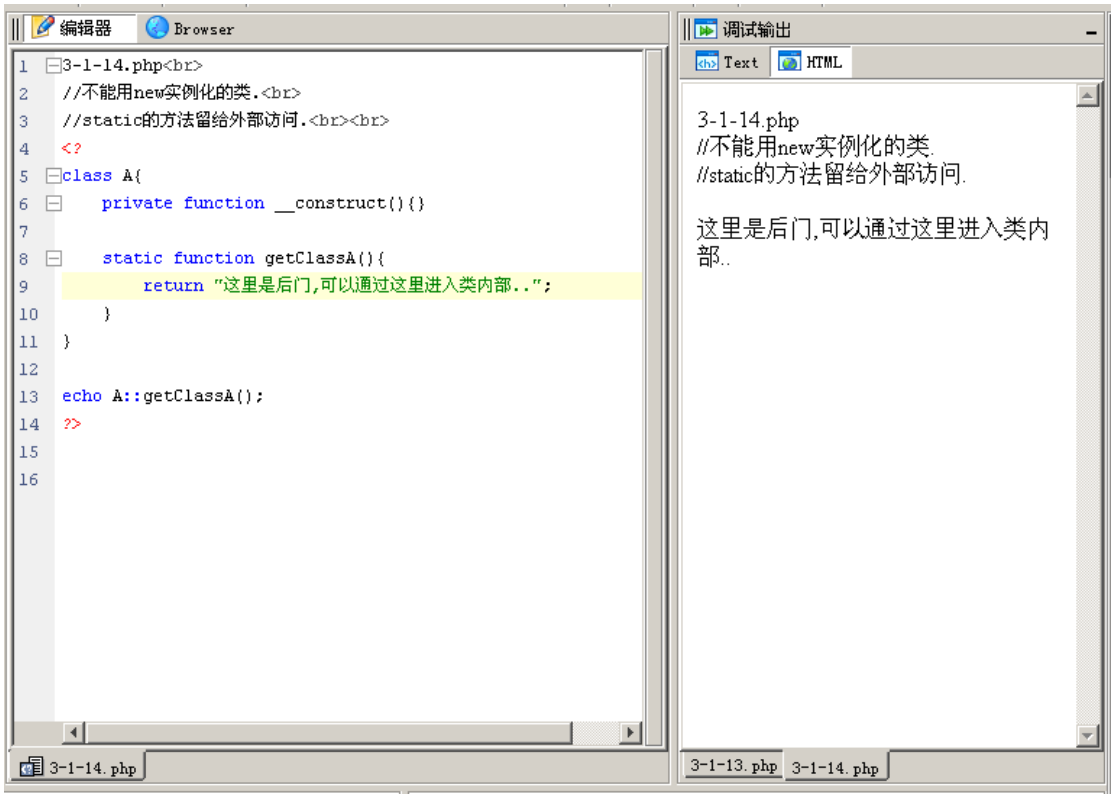
问题 2：

- 我们已经禁止外部用 new 实例化这个类，我们改如何让用户访问这个类呢?
- 前门堵了，我们需要给用户留个后门。

解决办法：

- static 修饰的方法，可以不经实例化一个类就可以直接访问这个方法。
- 后门就在这里。

例：3-1-14.php



问题 3：

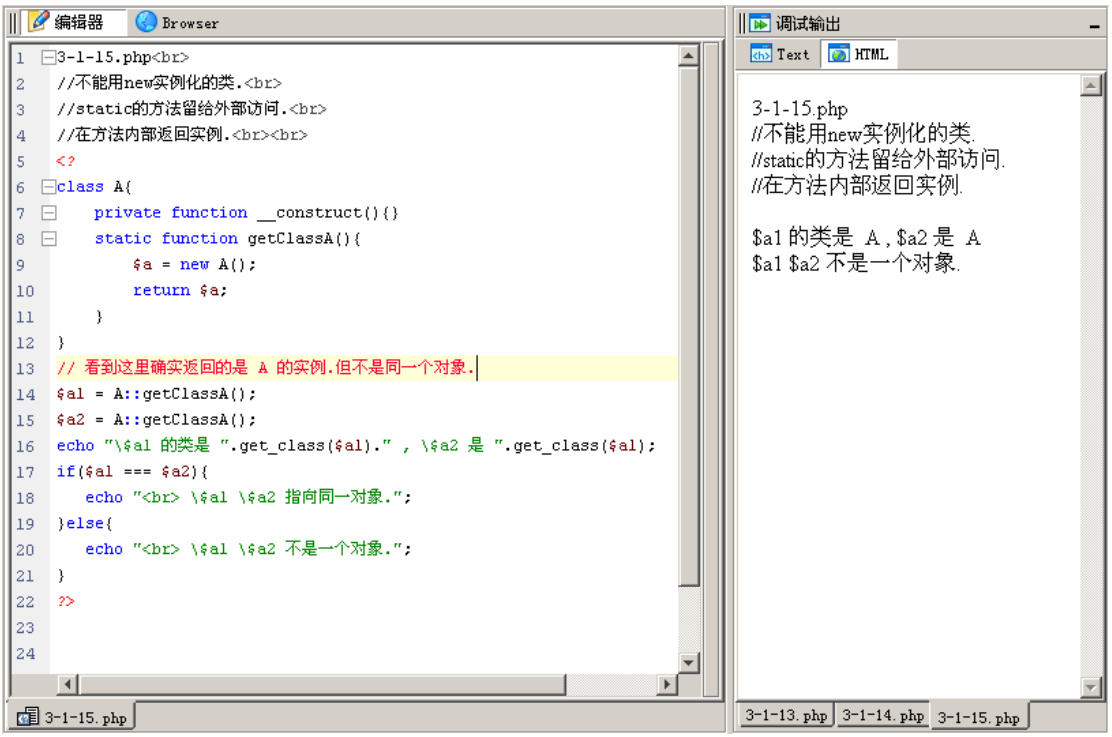
- 虽然我们已经进入类内部，但我们要的是这个类的唯一实例？
- 先不管别的，我们先需要一个实例。
- 通过这个 static 的方法返回这个实例，如何做呢？

解决办法：

- private 的构造函数，不能被外部实例化。
- 但是我们已经成功潜入类的内部了（间谍？007？），我们在内部当然可以调用 private 的方法创建对象。
- 我们这样做看看。

例 3-1-15.php

下面的例子我们确实返回了 A 类的实例，但注意两次执行返回的不是同一个实例。



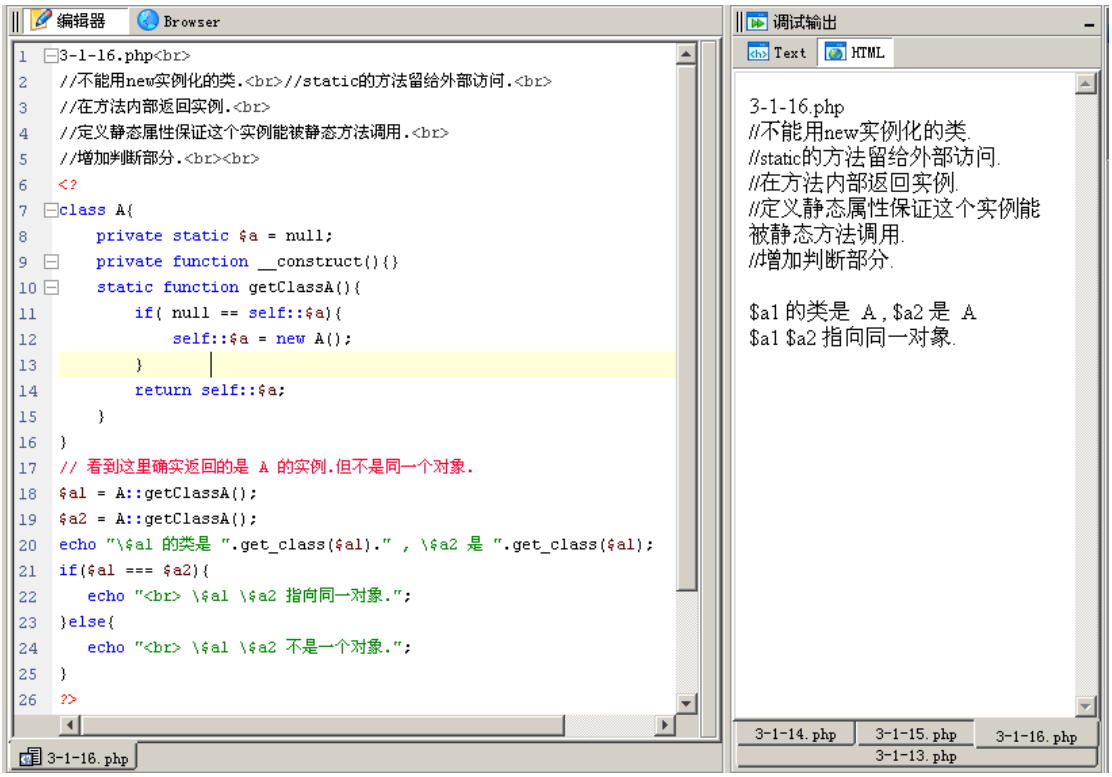
问题 4：

- 我们已经通过 static 方法返回了 A 的实例。但还有问题。
- 我们如何保证我们多次操作获得的是同一个实例的呢？

解决办法：

- static 的属性在内部也只有一个。
- static 属性能有效的被静态方法调用。
- 将这个属性也设置成 private，以防止外部调用。
- 先将这个属性设置成 null。
- 每次返回对象前，先判断这个属性是否为 null 。
- 如果为 null 就创建这个类的新实例，并赋值给这个 static 属性。
- 如果不为空，就返回这个指向实例的 static 属性。

例：3-1-16.php



- 到此，我们写了一个最简单的 单件模式 。
- 现在，你可以尝试写一个应用单件设计模式的数据库连接类。
- 要记住单件模式的使用效果和书写方式。

3.2final 类、final 方法和常量

- final---用于类、方法前。
- final 类---不可被继承。
- final 方法---不可被覆盖。

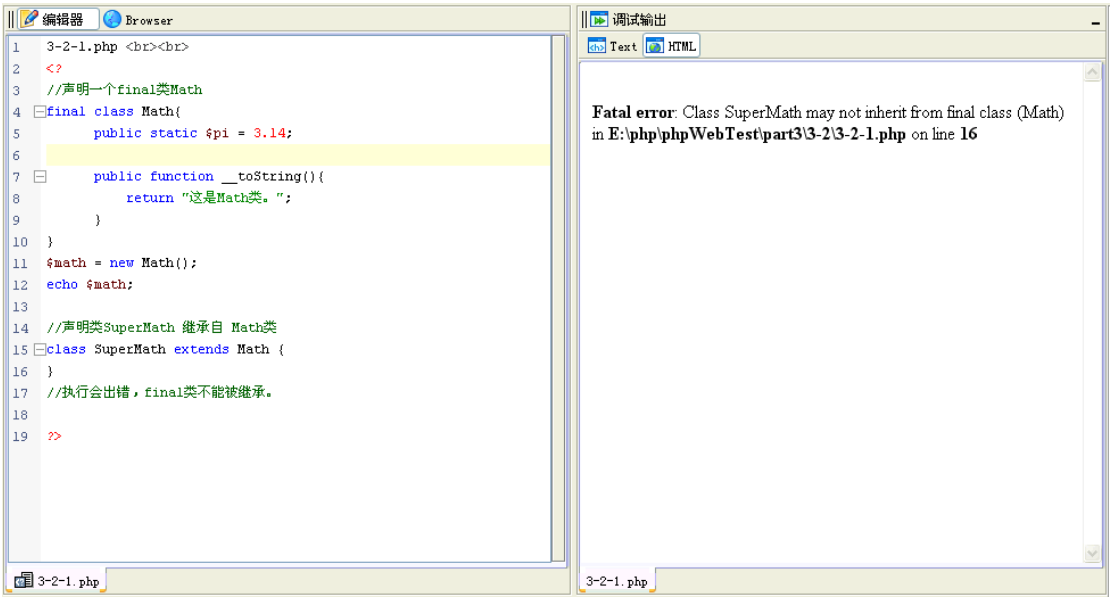
3.2.1 final 类的不能被继承

如果我们不希望一个类被继承，我们使用 final 来修饰这个类。

于是这个将无法被继承。

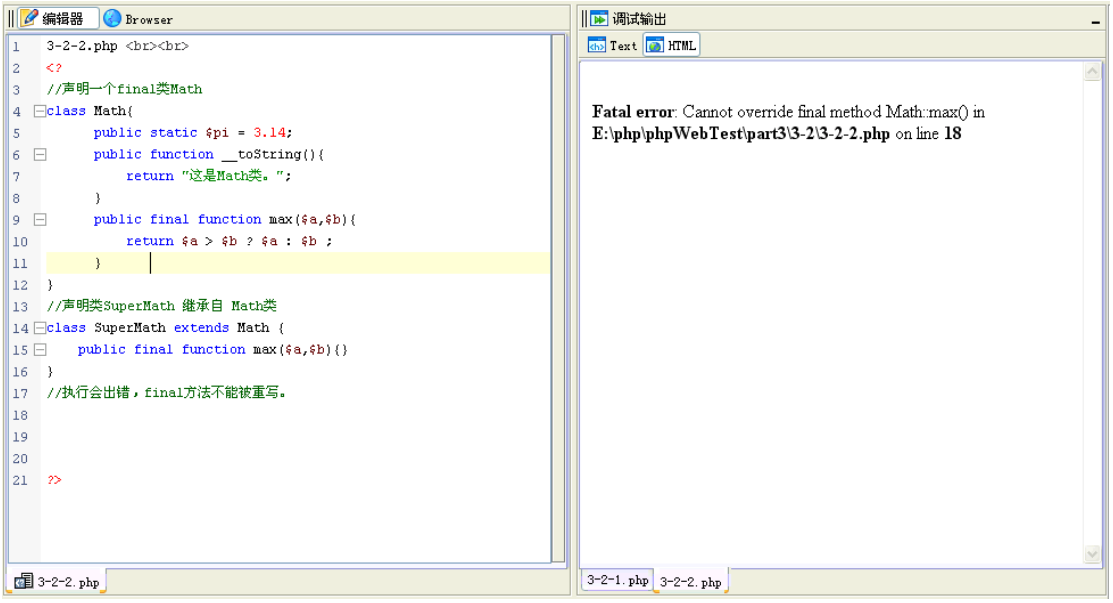
比如我们设定的 Math 类，涉及了我们要做的数学计算方法，这些算法也没有必要修改，，也没有必要被继承，我们把它设置成 final 类型。

例 3-2-1.php



3.2.2 final 方法不能被重写

如果不希望类中的某个方法被子类重写，我们可以设置这个方法为 final 方法，只需要在这个方法前加上 final 修饰符。
如果这个方法被子类重写，将会出现错误。
例：3-2-1.php

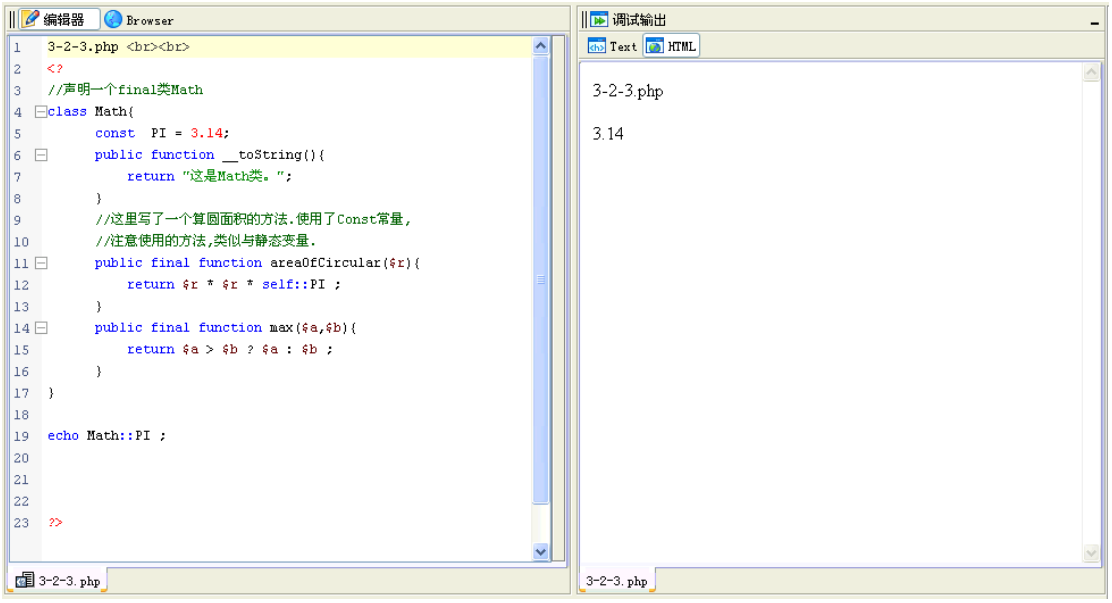


3.2.3 PHP5 中的常量

- 在 PHP5 类中继续使用 cons 修饰常量。
- 我们使用 const 定义一个常量，定义的这个常量不能被改变。
- 在 PHP5 中 const 定义的常量与定义变量的方法不同，不需要加 $ 修饰符。
  const PI = 3.14； 这样就可以。

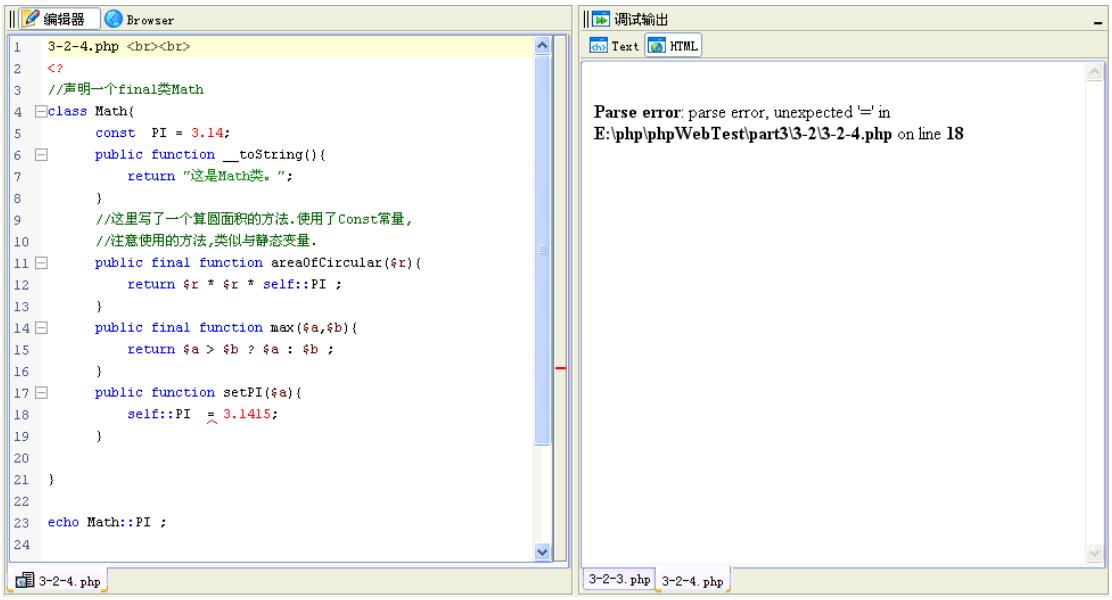
- 而使用 const 定义的常量名称一般都大写，这是一个约定，在任何语言中都是这样。
- 如果定义的常量由多个单词组成，使用 _ 连接，这也是约定。
- 比如， MAX_MUMBER 这样的命名方式。一个良好的命名方式，是程序员必须注意的。
- 类中的常量使用起来类似静态变量，不同点只是它的值不能被改变。我们使用 类名::常量名 来调用这个常量。

例 3-2-3.php



尝试为 const 定义的常量赋值，将会出现错误。

例 3-2-4.php



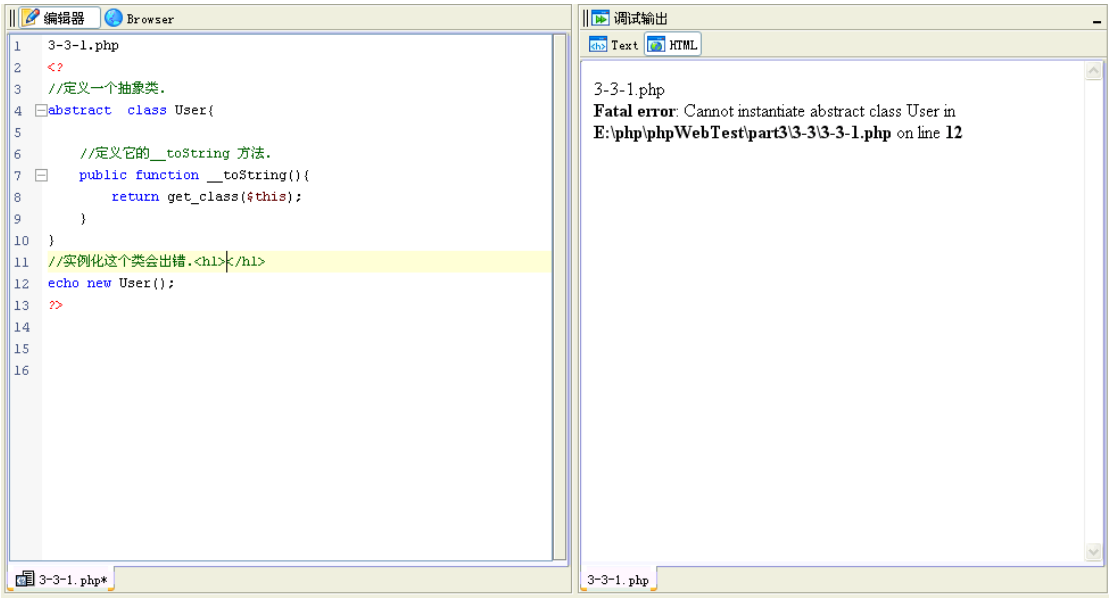
3.3 abstract 类和 abstract 方法

- 可以使用 abstract 来修饰一个类或者方法。
- 用 abstract 修饰的类表示这个类是一个抽象类，用 abstract 修饰的方法表示这个方法是一个抽象方法。
- 抽象类不能被实例化。
- 抽象方法是只有方法声明，而没有方法的实现内容。

3.3.1 abstract 抽象类

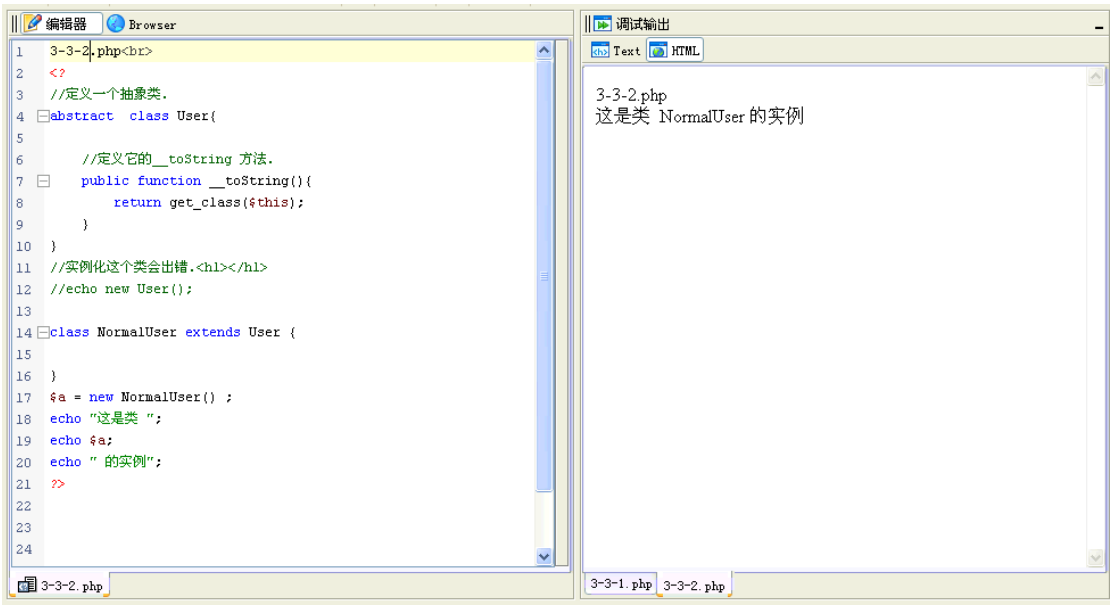
- 可以使用 abstract 来修饰一个类。
- 用 abstract 修饰的类表示这个类是一个抽象类。
- 抽象类不能被实例化。

这是一个简单抽象的方法，如果它被直接实例化，系统会报错。
例：3-3-1.php



例：3-3-2.php

下面例子的 NormalUser 继承自 User 类，就可以被实例化了。



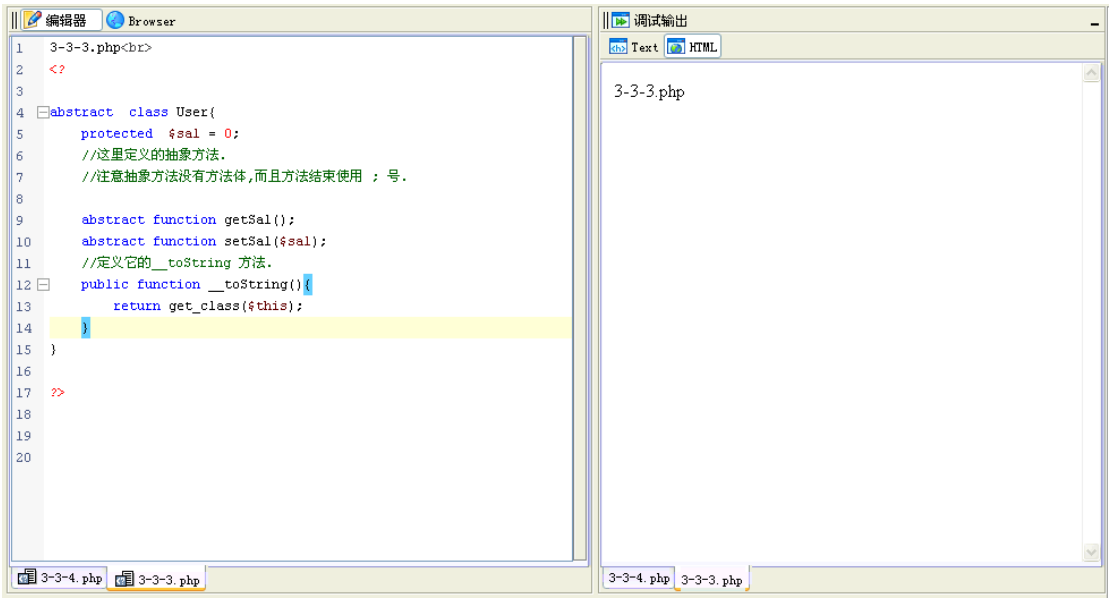
单独设置一个抽象类是没有意义的，只有有了抽象方法，抽象类才有了血肉。
下面介绍抽象方法。

3.3.2 abstract 抽象方法

- 用 abstract 修饰的类表示这个方法是一个抽象方法。
- 抽象方法，只有方法的声明部分，没有方法体。
- 抽象方法没有 {} ，而采用 ； 结束。
- 一个类中，只要有一个抽象方法，这个类必须被声明为抽象类。
- 抽象方法在子类中必须被重写。

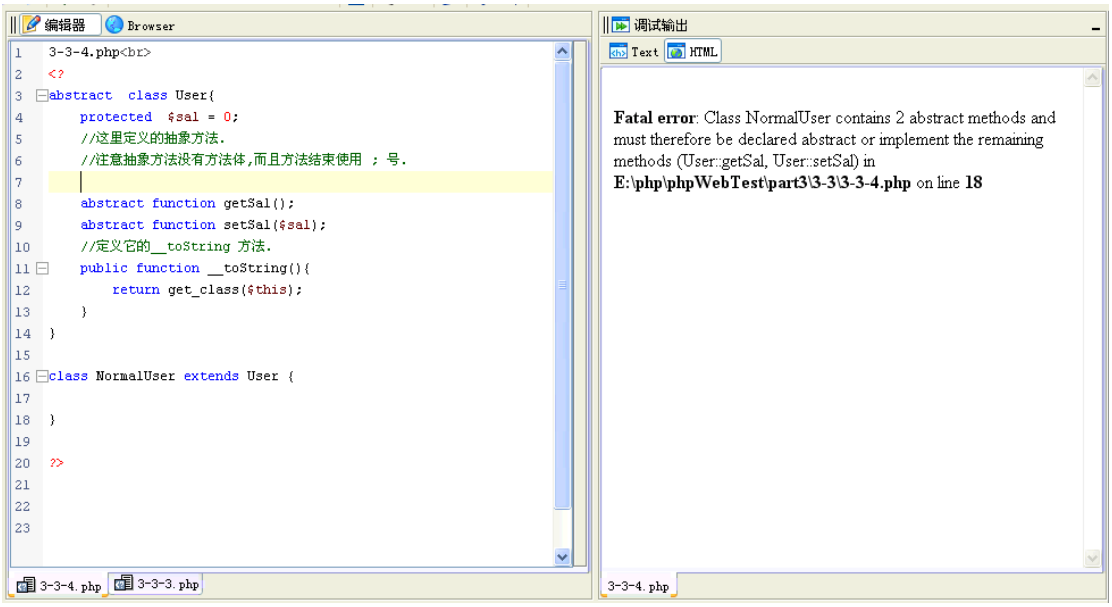
下面是一个抽象类，其中有两个抽象方法，分别是 setSal() 和 getSal()。用来取回$sal 员工的工资。

例：3-3-3.php



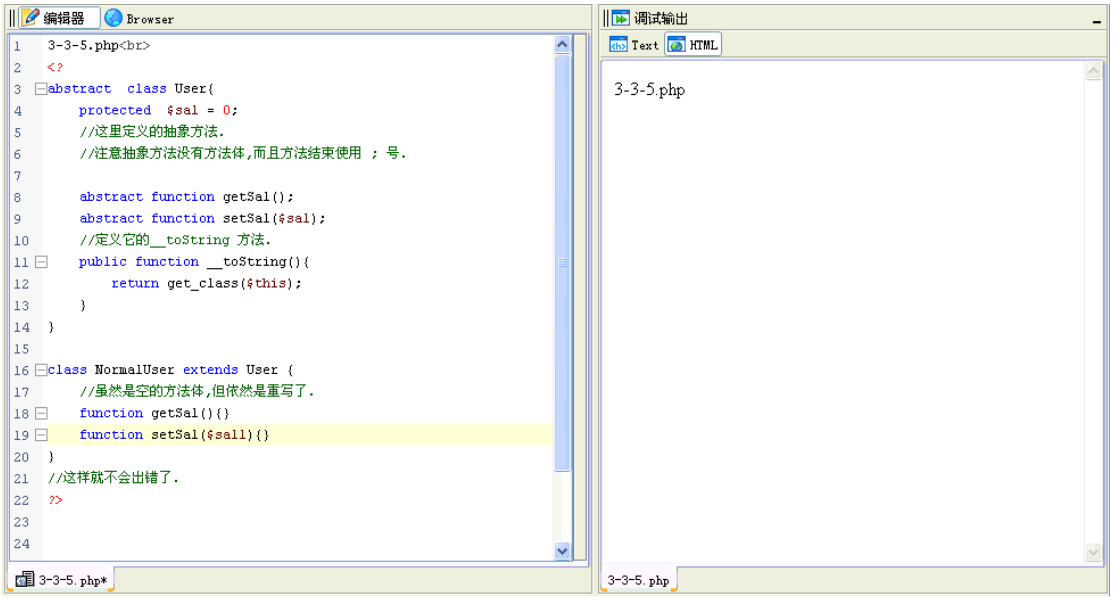
既然 User 类不能被直接继承，我们写一个 NormalUser 类继承自 User 类。
当我们写成如下代码时，系统会报错。
这个错误告诉我们，在 User 类中有两个抽象方法，我们必须在子类中重写这两个方法。

例：3-3-4.php



例：3-3-5.php

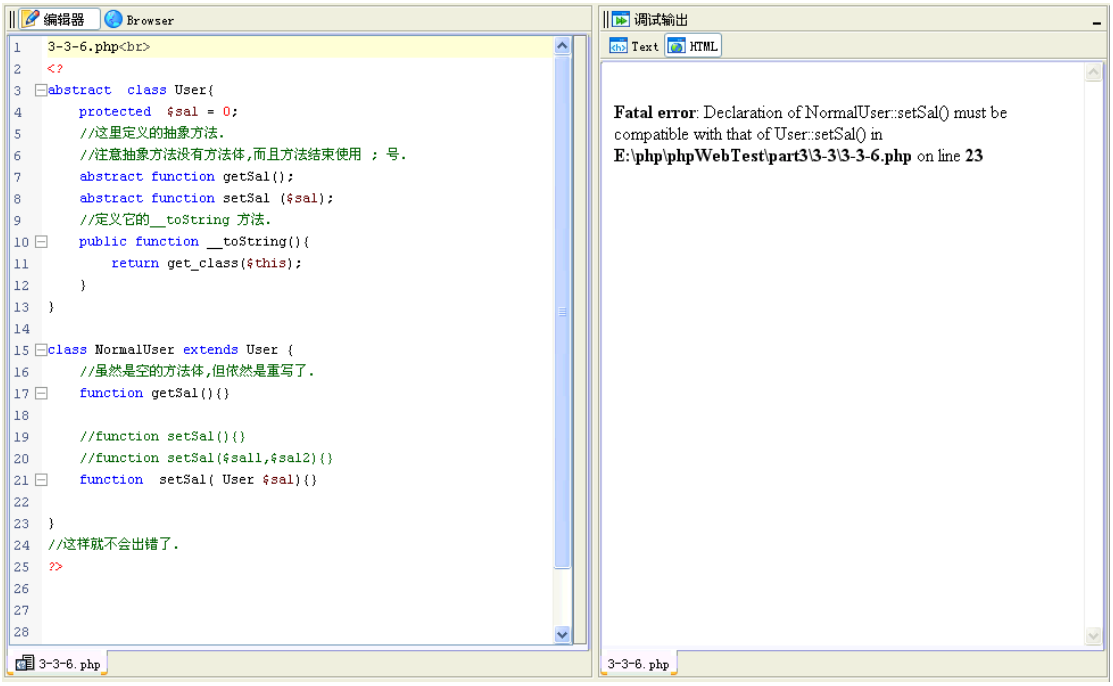
下面例子，重写了这两个方法，虽然方法体里面 {} 的内容是空的，也算重写了这个方法。
注意看重写方法的参数名称，这里只要参数数量一致就可以，不要求参数的名称必须一致。



下面 19-21 行，三种写重写的方式都会报错。

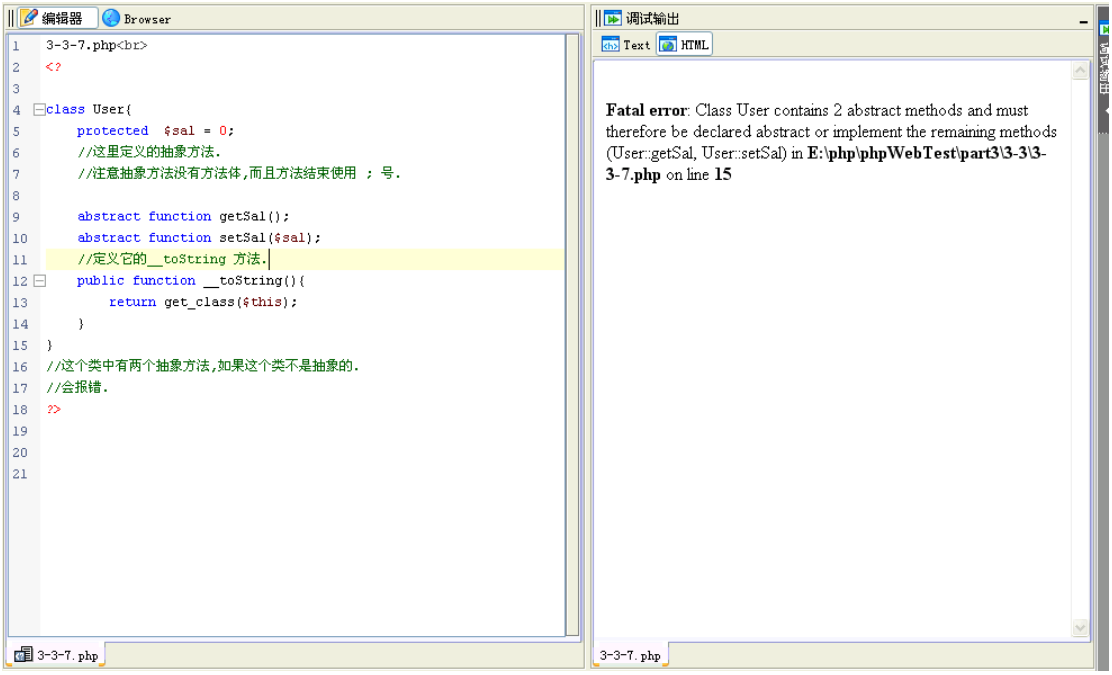
- 19 行，缺少参数。
- 20 行，参数又多了。
- 21 行，参数类型不对。（这种写法在以后章节介绍）

例:3-3-3.php



- 一个类中，如果有一个抽象方法，这个类必须被声明为抽象类。

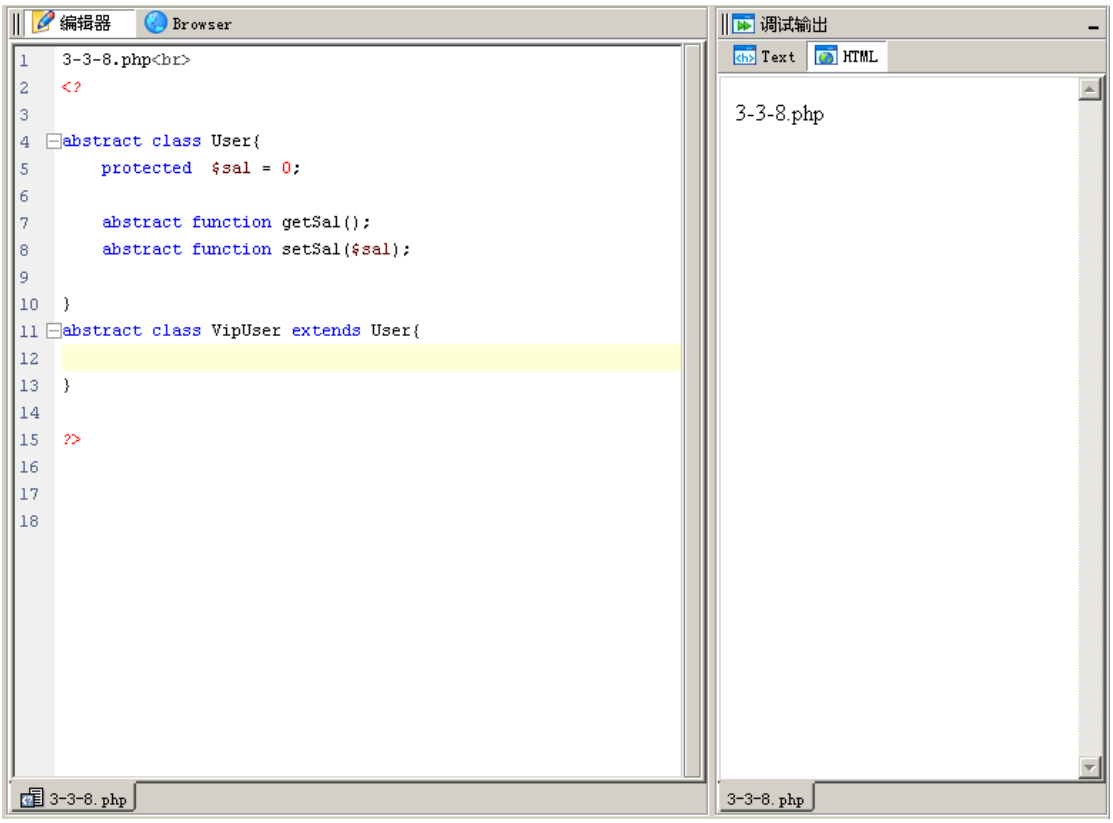
下面这个类不是抽象类，其中定义了一个抽象方法，会报错。
例：3-3-7.php



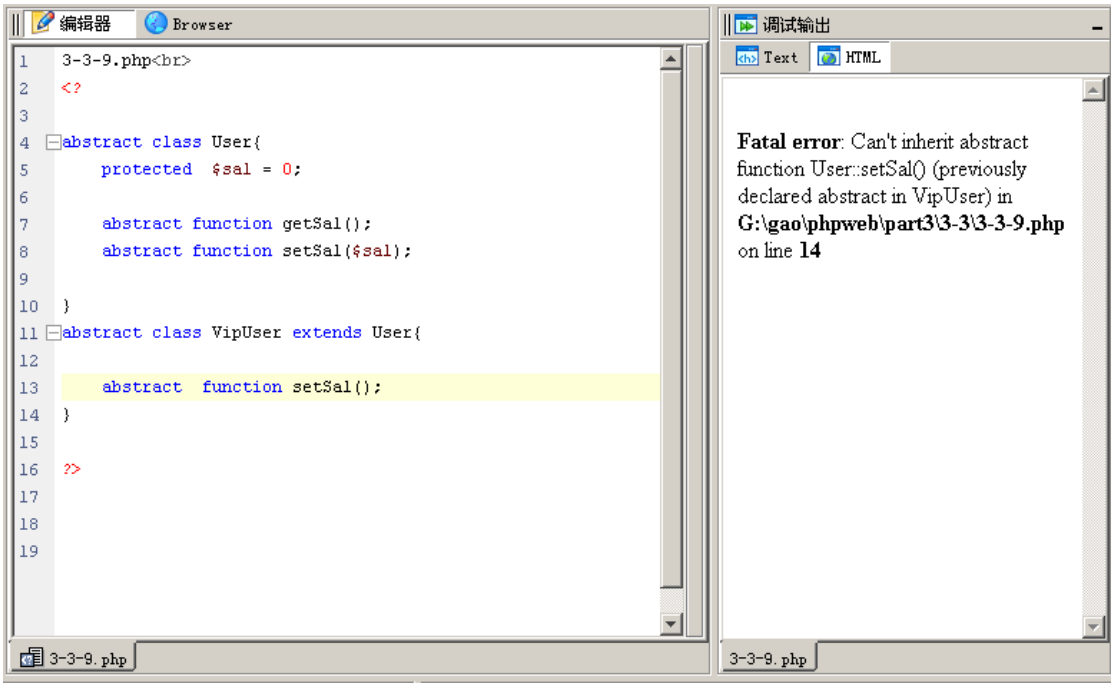
3.3.3 抽象类继承抽象类

- 抽象类继承另外一个抽象类时，不用重写其中的抽象方法。
- 抽象类中，不能重写抽象父类的抽象方法。
- 这样的用法，可以理解为对抽象类的扩展。

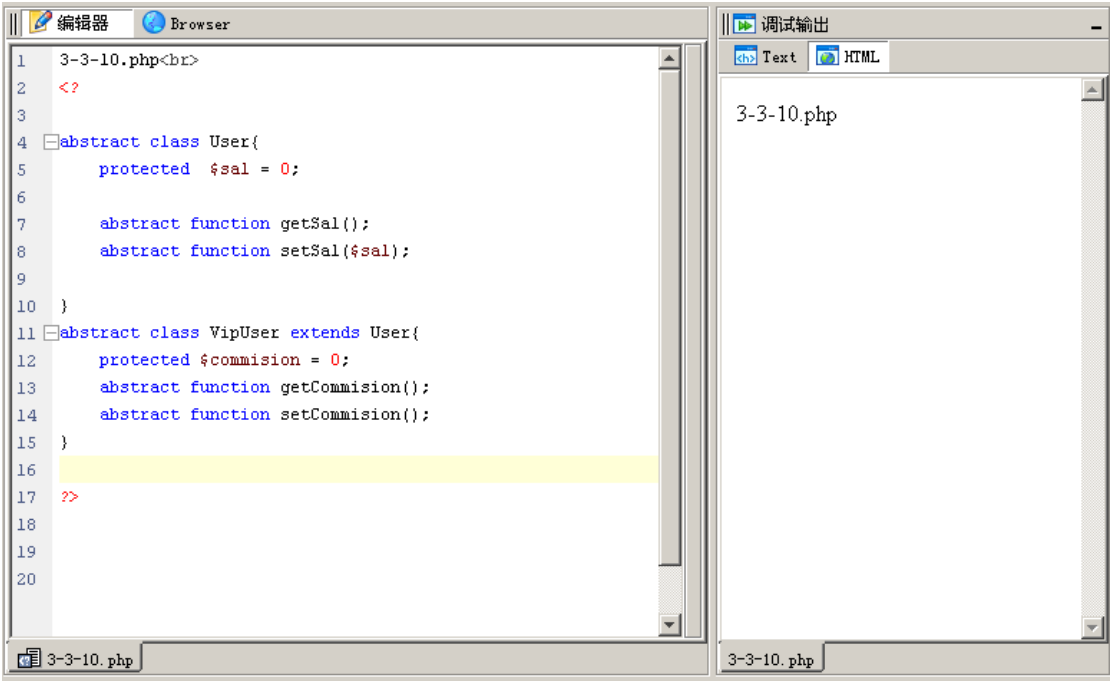
下面的例子，演示了一个抽象类继承自另外一个抽象类时，不需要重写其中的抽象方法。
例：3-3-8.php



抽象类在被继承后，其中的抽象方法不能被重写。
如果发生重写，系统会报错。
例 3-3-9.php



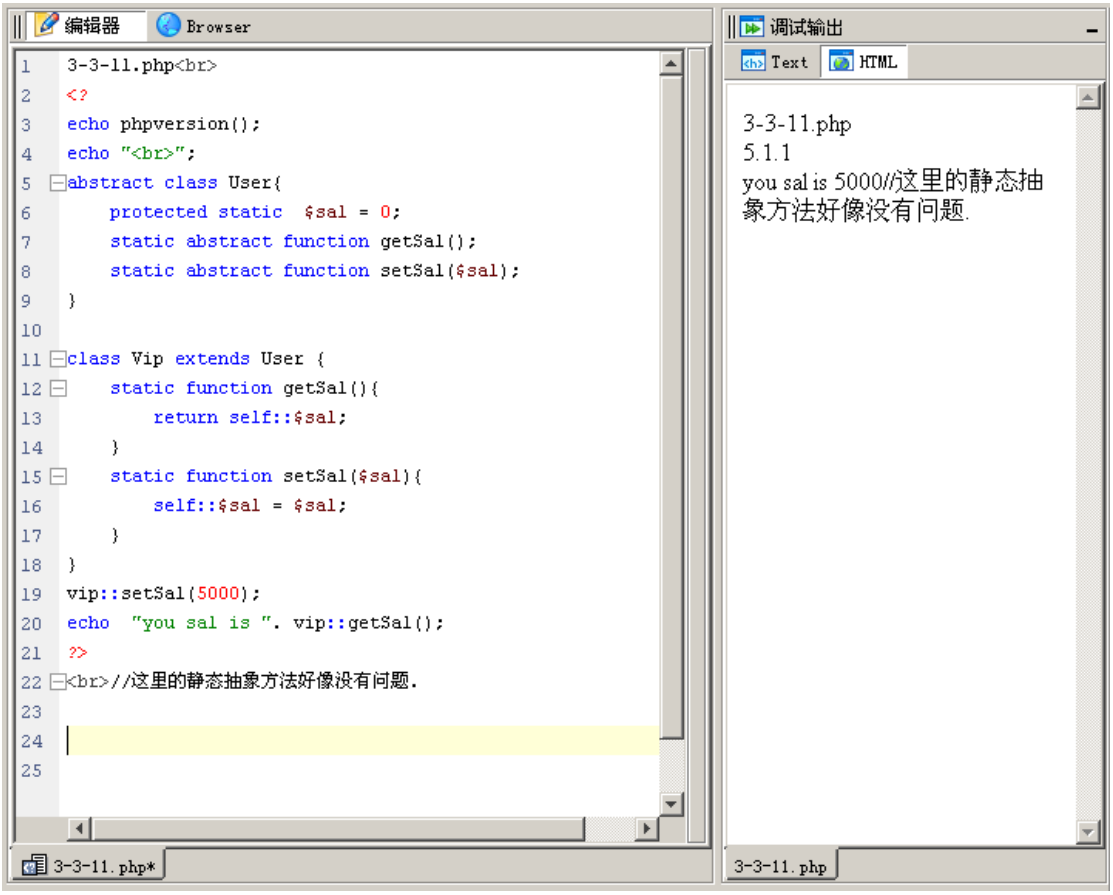
抽象类继承抽象类，目的对抽象类的扩展。
例 3-3-10.php



3.3.4 静态抽象方法

在 PHP5.1 中，抽象类中支持静态抽象方法。
下面这个例子，看到静态抽象方法可以声明。实现这个方法时，必须是静态的方法。

例：3-3-11.php



3.3.5 PHP5.2.0 中的静态抽象方法

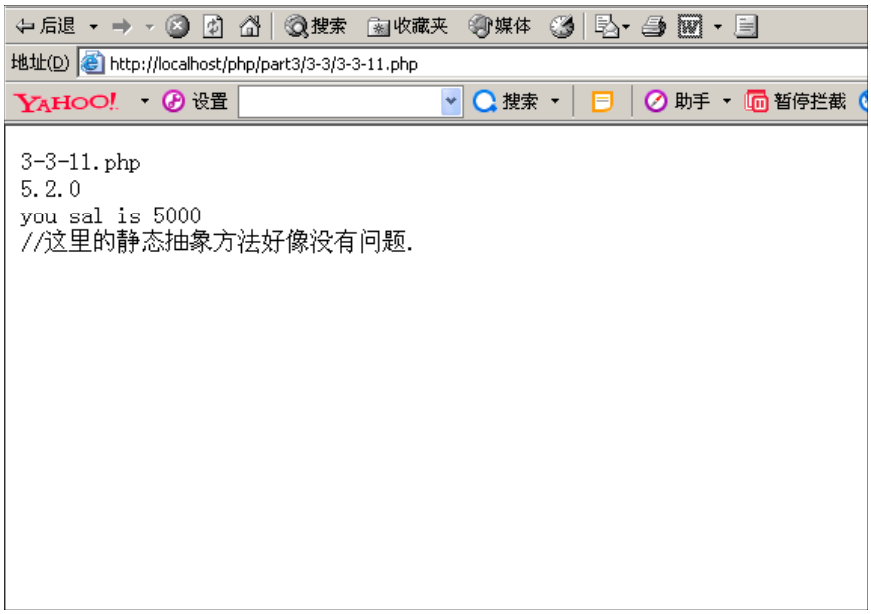
在 PHP5.2.0 的 更新信息中有这样一段话：

```
- Dropped abstract static class functions. (Marcus)

  Due to an oversight, PHP 5.0 and 5.1 allowed abstract static functions in
  classes. In PHP 5.2, only interfaces can have them.
```

译文：(因为疏漏，在 PHP5.0 和 PHP5.1 的类中允许静态抽象方法。
在 PHP5.2 中，只有接口可以拥有静态抽象方法。)

在 PHP5.2.0 版本中测试，刚才的代码执行正常。在 PHP5.2.0 中是依然兼容。



3.4 设计模式之模版模式

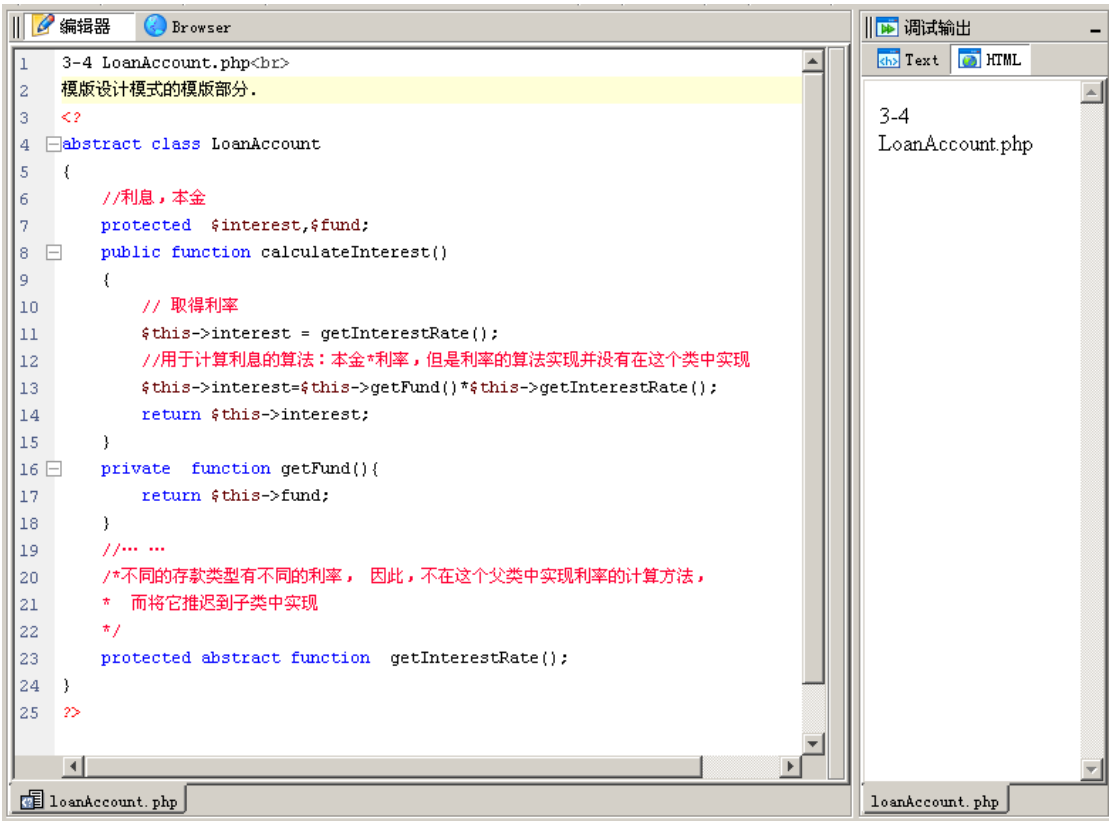
抽象类的应用就是典型的模版模式，先声明一个不能被实例化的模版，在子类中去依照模版实现具体的应用。

3.4.1 模版模式实例

我们写这样一个应用：

> 银行计算利息，都是利率乘以本金和存款时间，但各种存款方式计算利率的方式不同，所以，在账户这个类的相关方法里，只搭出算法的骨架，但不具体实现。具体实现由各个子类来完成。

例：LoanAccount.php




```
3-4 LoanAccount.php<br>
模版设计模式的模版部分.
<?
abstract class LoanAccount
{
    //利息，本金
    protected  $interest,$fund;
    public function calculateInterest()
    {
        // 取得利率
        $this->interest = getInterestRate();
        //用于计算利息的算法：本金*利率，但是利率的算法实现并没有在这个类中实现
        $this->interest=$this->getFund()*$this->getInterestRate();
        return $this->interest;
    }
    private  function getFund(){
        return $this->fund;
    }
    //… …
    /*不同的存款类型有不同的利率， 因此，不在这个父类中实现利率的计算方法，
    *  而将它推迟到子类中实现
    */
    protected abstract function  getInterestRate();
}
?>
```

以后，所有和计算利息的类都继承自这个类，而且必须实现其中的 getInterestRate() 方法，这种用法就是模版模式。

> 下一章将介绍面向对象语言的其它重要组成 接口、多态。