昨日回顾 2

面向对象思想介绍 3

传统的面向过程 3

现代的面向对象 4

面向对象基本概念 4

类与对象 4

对象创建的基本形式 5

对象的传值 6

类中成员 8

一般属性 8

一般方法 8

方法中$this关键字 9

静态属性 10

静态方法 11

ｓｅｌｆ： 11

构造方法 12

析构方法 13

昨日回顾

表A上的一个触发器，触发了表B的某项操作，则表B上的触发器，也能够继续触发表C的某项操作。

数据定义语言（DDL）：创建（设计）表，库，视图，等等

数据操作语言（DML）：增删改

数据查询语言（DQL）：select

[数据控制语言（DCL）](#Toc15804)

[mysql中的权限](#Toc23452)：就是一个固定的名词（单词），每个单词代表某种事情（功能），比如：insert， udpate， delete, select create, create view

[ｍｙｓｑｌ中的用户](#Toc32179):

[创建用户](#Toc3768)：create user ‘用户名’@’允许登录的地址’ identified by ‘密码’

[删除用户](#Toc30336)：drop user ‘用户名’@’允许登录的地址’

[修改用户密码](#Toc11412)： set password = password(‘密码’);

set password for ‘用户名’@’允许登录的地址’ = password(‘密码’);

[权限分配](#Toc18500)

[增加权限：](#Toc26844)grant 权限列表 on 数据库名.对象名 to ‘用户名’@’允许登录的地址’ identified by ‘密码’

[删除权限](#Toc25542)：revoke 权限列表 on 数据库名.对象名 to ‘用户名’@’允许登录的地址’

其中：all 表示所有权限

[事务控制语言（DTL）](#Toc30182)

[什么是事务](#Toc11298)：就是一个内部机制，是我们的多条常规sql语句可以像“一条语句”一样实现“整体执行效果”。

ＩｎｎｏＤＢ存储引擎可以使用事务，MyIsam却不可以。

[为什么需要事务？](#Toc9222)

[事务的特点](#Toc2883)：原子性，一致性，隔离性，持久性。

[事务模式：](#Toc19651)autocommit是一个“系统变量”，其决定通常的一条sql语句是否当作一个事务来“执行”。

默认情况下autocommit=1，就是一条语句一个事务，无需额外声明，也就无需commit（提交）

[事务的基本实现流程：](#Toc14028)

1，start transaction（或begin）

2，执行多条常规sql语句

3，判断是否有错：

如果没有错误发生： commit（提交）

如果发生错误： rollback（回滚）

[mysql编程](#Toc28804)

[基本语法形式](#Toc4503)：

程序块的形式：

[标识符ABC] begin

多条语句；

end [标识符ABC] ;

创建存储过程和存储函数和触发器（编程环境）时，通常需要修改语句结束符：delimiter $

[基本流程控制语句：](#Toc25245)if， case， loop， while， repeat

[ｍｙｓｑｌ中的变量](#Toc13822)：

普通变量：需先声明，后赋值使用。并且此变量只能在编程环境中使用。

declare 变量名 类型 [default 默认值]

set 变量名 = 值；

会话变了：变量名前带“＠”，可以直接赋值使用。

set @变量名 = 值；

[存储函数](#Toc22400)：

create function 函数名(形参1 类型，形参2 类型....)

returns 类型

begin

//函数体语句。。。

//不可以有select这种语句。

return XX值；

end；

[存储过程](#Toc32137)：

create procedure 函数名(形参1 类型，形参2 类型....)

begin

//语句块。。。

//可以有select语句，并能够因此“得到”结果集（数据集）

end；

[触发器](#Toc24197)

[什么叫触发器：](#Toc23483)当一个表发生（增删改）某个“事件”的时候，可以在其之前（before）或之后（after）去完成的某些任务。

[定义形式：](#Toc30541)

create trigger 触发器名 before/after insert/update/delete on 表名 for each row

begin

触发器语句块，不能return数据，也不能有select语句。

end；

面向对象思想介绍

ＯＯＰ：Ｏｂｊｅｃｔ　Ｏｒｉｅｎｔｅｄ　Ｐｒｏｇｒａｍ面向对象编程。

传统的面向过程

将要完成的工作，分作若干个步骤，或再细分为子步骤，然后后步骤从前往后一步一步完成，最初达致目标。

现代的面向对象

将要完成的工作拆分为“一个一个对象”的任务（功能），每个对象独自完成自己的任务，任务之间的链接通过“调用”来实现，最终也完成了整体的工作。

万事万物都是对象。

面向对象是设计思想的“升华”——面向过程是解决简单问题的自然的思维过程，面向对象是解决大规模的复杂问题的良好思想。

面向对象基本概念

类与对象

张三是一个对象，李四也是一个对象。王五同样是一个对象。。。。。

他们都隶属于“人”这个“类”（人类），同时我们也可以说隶属于“脊椎动物类”，还可以说属于动物类，甚至生物。

类：是用于描述“某一些具有共同特征”的物体的概念，是某一类物体的总称。

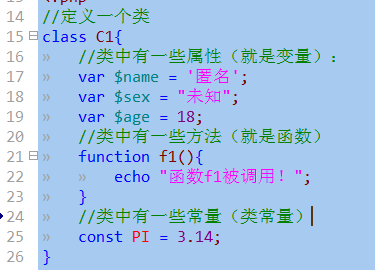
通常，一个类所具有的共同特征包括2大方面的信息：

外观，形状，描述，数据。。。。。属性

行为，动作，功能。。。。。。。。方法。

对象：是指一个具体的“物体”，该物体隶属于某个“类别”（类）。

通常，对象离不开类，没有类，就不能有对象。



可见，类无非就是将一个变量和一些函数以及一些常量用一个“名字”包起来了，此名字就是“类名”。

另一方面，我们就说，这些属性，函数，和常量就“隶属于”该类——只能由该类（及其创建出来的对象）来使用。

原来（在面向过程）的语法中，我们的代码有如下几种代码：

定义变量；

定义函数；

使用变量（输出，赋值，等）

调用函数；

流程控制（if，switch，for，while等）

在面向对象的语法中，则情况就发生变化了：

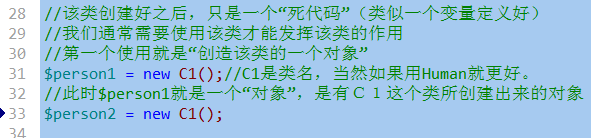
1，定义类；定义类的语法中，只有这3种代码：

1.1定义属性（变量）

1.2定义方法（函数）

1.3定义常量（类常量）

2，创建类的对象；



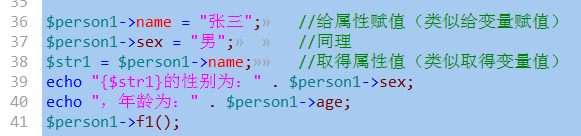
3，使用对象；

使用对象也几乎只有3个方式：

3.1使用其属性：因为属性就是“变量”，则使用其属性，也就是跟以前使用变量一样。

3.2使用其方法：因为方法就是“函数”，则使用其方法，也就是跟以前使用函数一样。

3.3使用其常量：类似使用以前的普通常量。



对象创建的基本形式

方式1：

$对象名1 = new 类名（）； //对象名就是变量名；类是应该定义过了

方式2：



方式3：

$对象名3 = new self; //self指代类本身，这行代码只能在类内部方法中使用。

方式4：通过对象创建对象

* new 对象名;//创建该对象所属类的一个新对象



对象的传值

回顾：

ｐｈｐ中，变量传值方式有２个：

值传递： 传递的时候，拷贝的是数据本身。默认都是值传递。

结果：传递完之后，有了2份同样的数据，但两个变量“相互独立”没有关系。

引用传递： 传递的时候，拷贝的是引用关系。需要使用“＆”才能实现引用传递。

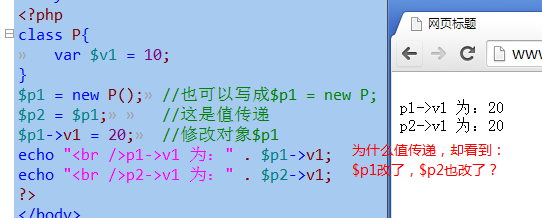
结果：传递完之后，数据仍然只有一份，但是两个变量共同指向该数据。

则在面向对象这个层面，基本来说，仍然如此：

默认是值传递：

可以使用引用传递：

先观察一个现象：



这是什么原因？

原来，对象的存储跟普通变量的存储有所区别：

普通变量：$s1 = 10;

普通数据：10

变量：$s1

引用关系

对于对象（变量）：

$obj1 = new P();

对象数据：

{$v1 = 10}

对象标识符#1

变量：$obj1

引用关系

引用关系

则对象变量复制的时候，值传递就变成这样：

$obj2 = $obj1; //值传递

对象数据：

{$v1 = 10}

变量：$obj1

对象标识符#1

引用关系

引用关系

复制

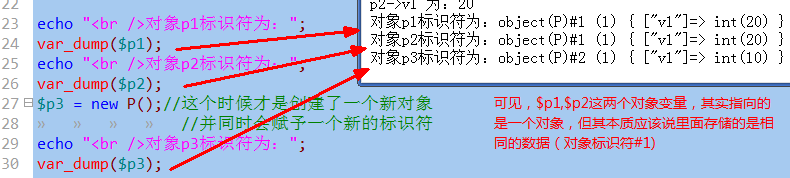
引用关系

对象标识符#1

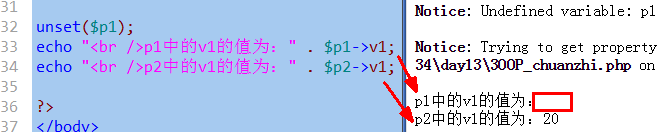
变量：$obj2

引用关系

可见，对象变量进行普通传值的时候，复制的是对象变量中所存储的“对象标识符”

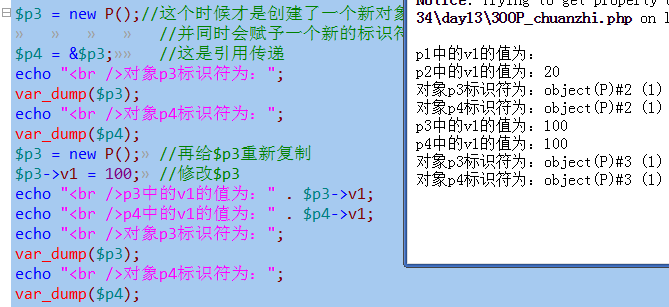


那么，我们断开$p1跟其存储的数据（标识符）之间的关系，结果是：$p2还有效：



可见$p1和$p2之间是值传递。

再来对比看看引用传递：



类中成员

一个类的内部可以有3种代码：

属性

方法

类常量

它们统称为“类中成员”。

一般属性

属性就是放在一个类中的变量。

定义形式：

形式1： var $v1 ; //定义不赋值

形式2： var $v2 = 2; //定义的同时可以赋值，该值只能是“直接值”，常量（值）

//不能是变量值，也不能是“计算表达式”

形式3： public $v1;

形式4： public $2 = 2; //其实var是“public”一个别名，正式用public更好。

错误形式：

$v1 = 1;

var $v2 = 1+3; //右边不能是计算表达式

public $v3 = $v2; //右边不能是变量

使用形式：

$对象->属性名； //注意：属性名前面没有$符号。

说明：

通常，可以出现变量（数据）的位置，就可以出现对象的属性。

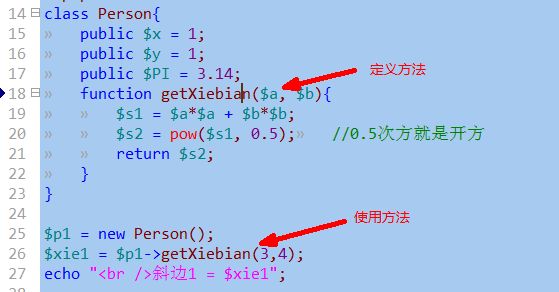
课堂课间练习：

写一个人类对象，具有３个属性，并能够完成求直角三角形的斜边的工作。

一般方法

方法定义跟原来函数定义几乎一样。

使用是通过类或类的对象来进行的。

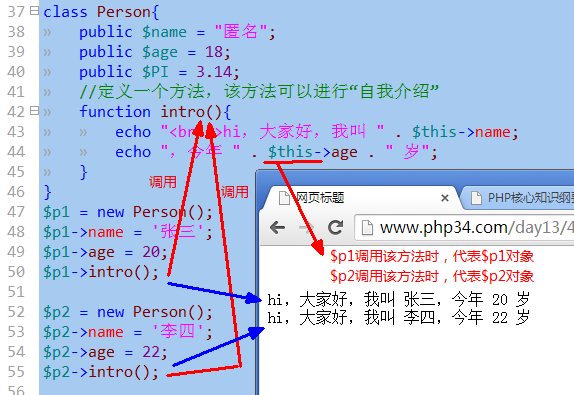


方法前可以加修饰词：ｐｕｂｌｉｃ，　ｐｒｏｔｅｃｔｅｄ，ｐｒｉｖａｔｅ，省略就算“public”

方法中$this关键字

$this是一个“伪对象”，代表当前所属类的当前对象。

举例：



一些简单说法：

类是定义好的某些对象的“模板／模型”

对象是根据该模板／模型所“创建”出来的具体物体

一个对象可以有哪些属性（数据），是由该类来决定的。

一个对象可以做那些事情（方法），也是由该类来决定。

通常认为，一个类（对象），就是拥有一些数据，并通过它自己的方法可以去处理这些数据的“独立体”

一个系统函数：ｇｅｔ＿ｃｌａｓｓ（对象）

作用：获得某个对象的“所属类名”——结果只是一个类名字符串。

$class\_name = get\_class($p1); //结果可能是：”Person”

静态属性

在定义属性的时候，前面加上关键字：static,此时就是静态属性。



静态属性有什么用？

对比来说：一般属性，他的值是隶属于该类的具体某个对象（虽然定义在类中），或者说，每个对象的同样的属性的值，有可能不一样。

静态属性就是：只隶属于类本身——也可以看做是所有对象的“共有数据”。

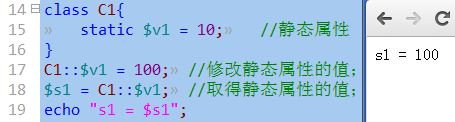
静态属性的使用：

通过特殊的语法：::（双冒号语法，也叫范围解释符）

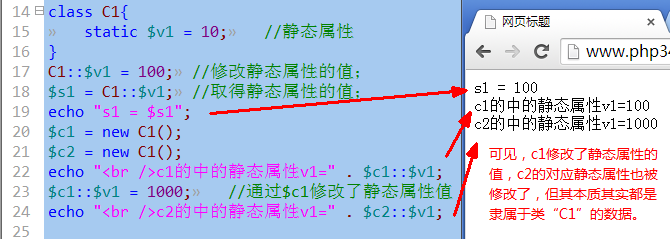
类::$静态属性名；

对比普通属性的使用：

对象->普通属性名； //普通属性也可以叫做“实例属性”，相应的，对象也叫“实例”



实际上，通过对象也可以使用静态属性（但属于最新的语法，也不推荐）：



静态方法

如果在一个方法的前面加上static关键字修饰，则就变成了静态方法。

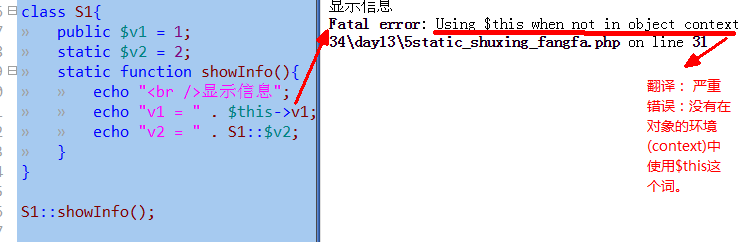
静态方法同样隶属于类，而不是隶属于具体对象。使用方法跟静态属性类似：

类名::静态方法名();

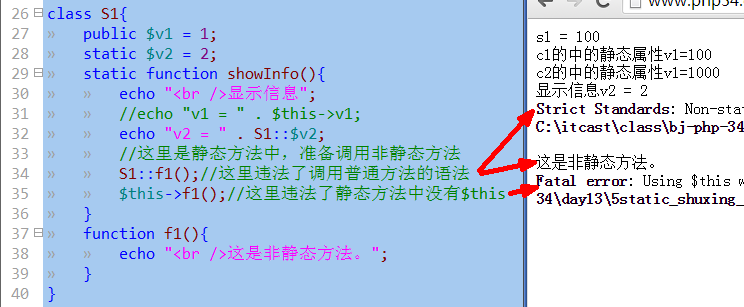
从理念上，可以认为，静态方法是只隶属于类，而为所有对象所“共有”。

如果通过类名来调用静态方法，则该方法中不可以出现$this关键字。

* 注意1：$this在静态方法中不能使用。



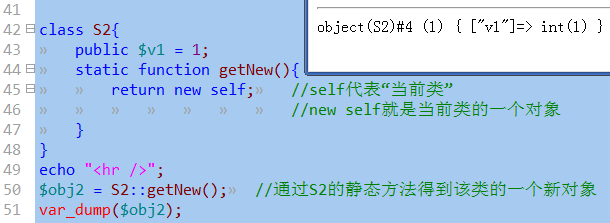
静态方法中不应调用非静态方法。



ｓｅｌｆ：

含义：代表当前类

使用：通常只能在某个类的某个方法内部代表该类的名称。



构造方法

构造方式是类中的一个“特殊”方法，其作用是在实例化一个对象的同时，给该对象的属性赋值，使之一创建完成，就具有了其本身的特有数据（属性值）。

1，该方法名字是固定的，为：\_\_construct();

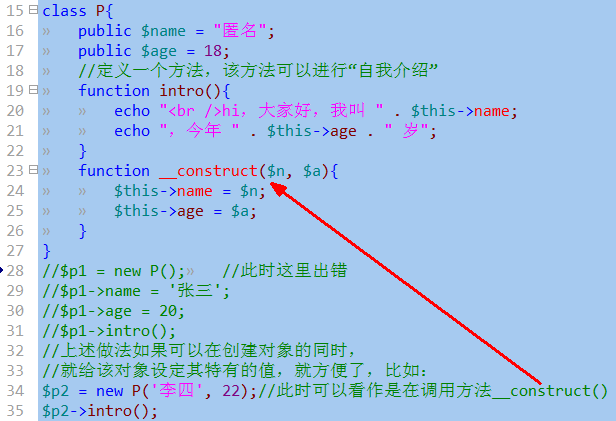
2，该方法必须是普通方法（不能是静态方法）

3，通常该方法应该是public

4，通常该方法中使用$this这个关键字来对属性进行赋值

5，当new 类名()的时候，其实是在调用该构造方法

6，如果一个类中定义了构造方法，则实例化该类时就会调用该方法，且实例化时的参数需要跟构造方法的参数匹配



结果为：



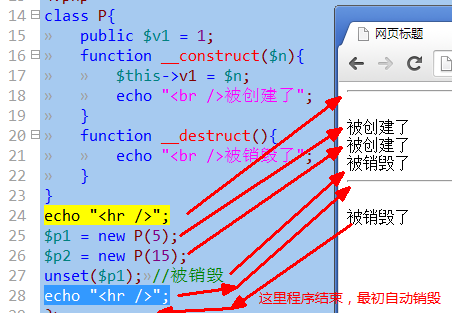
析构方法

构造方法是“创建”对象的时候会自动调用。

析构方法是“销毁”对象的时候会自动调用。

系统方法通常用于在销毁对象的时候来“清理数据”（打扫战场）——如果需要，就可以利用这个机会去处理。

通常，ｐｈｐ程序结束后，所有对象都会自动销毁（其实属于ｐｈｐ内部的垃圾回收机制）



说明：

１，析构方法通常不太需要去定义。

２，析构方法不能调用。

３，析构方法不能有形参。

４，析构方法中可以用于清理一些在ｐｈｐ代码结束后不能清理的数据，如生成的文件。

对象销毁的几个情形：

* 脚本程序运行结束，自动销毁；
* 明确地unset()一个对象变量，则被销毁；
* 改变对象变量的值，被销毁；

实际上，更本质来说，当一个对象（new 出来的对象）没有任何一个变量指向它的时候，该对象就会被自动销毁——自然，如果整个程序结束，也会销毁。