# 面向对象是什么？

编程思想。指导程序如何编写！

思想指导程序，程序体现思想。

思想的基础：

操作的支配者

程序：

方法的对象。

对象：具有功能和属性的，现实业务逻辑中的实体，在计算机程序中的映射！

PHP程序中，对象如何产生？

通过对类的实例化产生的。

类：对象的设计图。（对象的属性和方法的抽象）

实例化：通过类，得到对象！

Tip：只负责设计类，实例化的工作，new （PHP核心程序）自己完成。

类的声明：

Class 类名 {

常量 constant

声明的成员列表

属性 property（member variables）

方法 method（member function）

}

对象标志符 = New 类(构造方法实参列表)

对象标识符通常存储于变量中！

## 模拟声明类，实例化的过程

PHP源代码，存储于PHP文件，而运行于内存中。

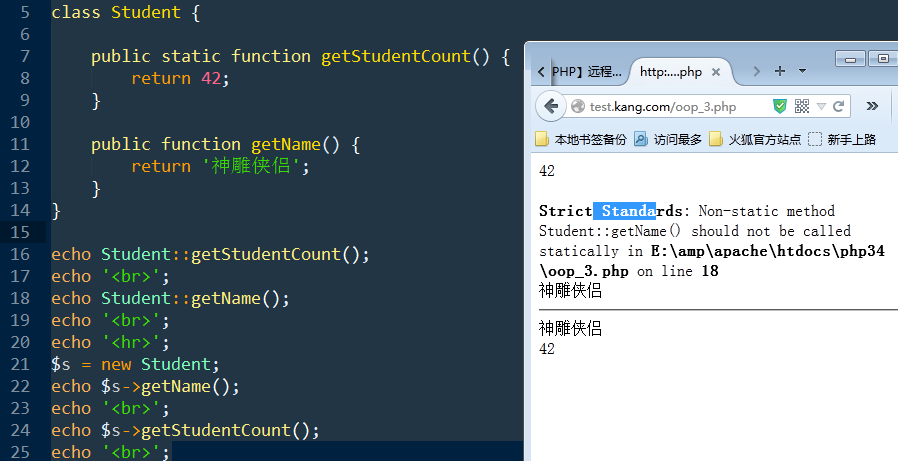
PHP核心程序，找到相应的PHP文件，获取内部PHP代码，载入内存，编译，执行！

将内存分成两个逻辑区域：

代码区，数据区。

## 静态成员

注意静态方法：



都可以在语法在调用：

逻辑上：静态为所有对象共享（属于类），非静态方法为每个对象所独有（属于对象）

语法上：静态方法没有当前对象的概念。非静态方法，在使用对象的调用过程中，可以确定当前对象。

强烈建议：

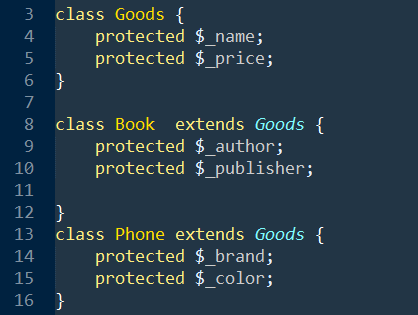
类静态调用静态方法

对象调用非静态方法（对象方法）

# 继承

一个对象如果使用另一个对象的成员，则称之为继承自另一个对象。

子类继承父类。



问：Book类与Phone类在成员结构上是否有变化？

不会！仅仅是在类之间建立关联。无论做什么操作，都是利用关联相互查找的。

注意$this，在子类对象调用继承自父类的方法时，方法中的$this是子类（调用方法类）的对象。

重写：不是重新定义，而是沿着继承链，查找，先找到的是要使用的。

静态成员的继承：

# 访问修饰

Public, protected, private

用来描述：

成员，在源代码中的哪些位置可以访问 。

Public：公共的：类内，类外，继承链类内

Protected: 类内，继承链类内。

Private：本类内。

访问限制：

依赖于源代码结构的。

而不是依赖于对象。

注意：

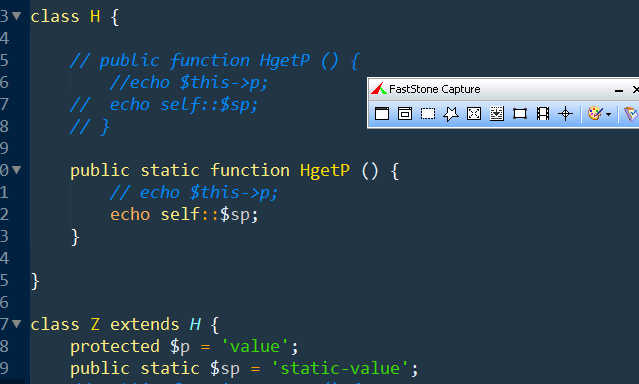
继承和访问的关系。不是一件事。

私有成员可以被继承。



如果是静态成员，则父类不能访问子类定义的静态成员。

注意：继承是单向的，向上继承。





对象是否可以访问成员，的前提是对象是否具有该成员，才能去思考是否可以访问。

# 抽象abstract，终极final

可以用来修饰 类，及 方法。

## Abstract

抽象类：

不能实例化，可以被继承。

抽象方法：

包含于抽象类中，继承抽象类的子类，必须要重写实现父类抽象类中的抽象方法。

可以限制子类子类中一定存在，但是实现可以不同的方法！

通常在，子类的实现过程中，大多数都是不同的实现。（10中有8种实现）

（如果子类中的实现，大多数为相同的，仅仅存在少数不同（10中有2个实现不同），应该父类定义，而不同的重写）

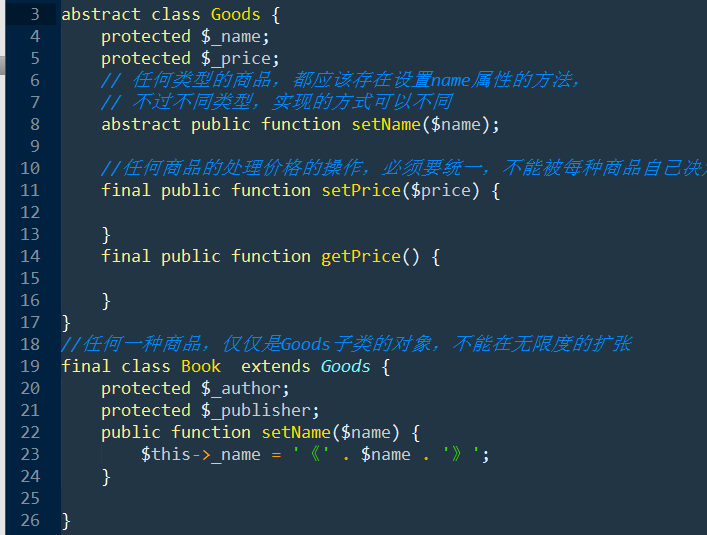
## Final

终极类：

限制类不能被继承。

终极方法：

限制方法所属的类如果被继承，该方法不能被重写。



设计类，当需要某种限定时，使用抽象，或者 终极。

# 接口结构interface

接口：interface：对外的操作，都称之为接口。语法上，只有公共的操作才是接口方法。

对象（实体）：满足封装性：隐藏内部实现，开放外部操作。

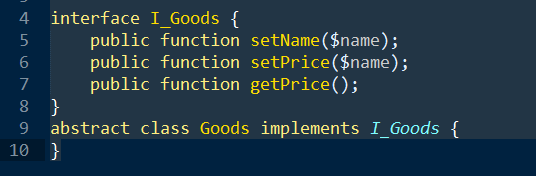
开发者：不想（不需要）告知用户对象（实体）的实现原理（实现步骤）

使用者：不想（不需要）知道你怎么实现，只需要知道怎么使用。

使用 访问修饰，和 接口结构来完成封装性体现。

访问修饰：决定哪些可以被访问到（外部操作），哪些不能被看到（内部）

接口结构： 用来说明一个类（对象）具备哪些公共操作结构。反之，如果接口上说明了，则类或者对象必要存在。（说明书）。



语法上的注意：

接口中仅仅可以定义抽象方法。

与抽象类的差异？

抽象类：在一个普通类的基础上，增加一部分抽象方法。因此具备 为子类提供基础操作的同时对子类的结构进行限制。

接口：仅仅说明类应该具有的操作。

抽象类：中的抽象方法，应该没有限制。

接口：中的抽象方法，都要是public。

抽象类：被子类所单继承。

接口：可以被类多实现。

# 自动加载

管理类文件的方式：

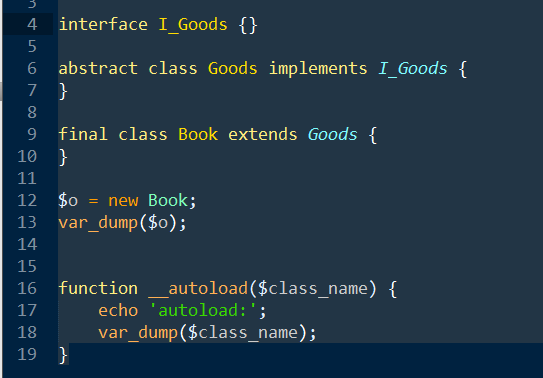
当程序执行到需要一个类时，会自动触发（调用）一段代码，该段代码负责完成通过类名（接口名）找到对应的类文件，再将其载入（require）的机制。

自动：PHP核心程序，可以检测类需要但是没有定义，调用某段代码

用户代码：定义（实现）这段完成载入类文件的代码。

语法：

默认的代码，为\_\_autoload()函数，参数为当前类名。



前提：

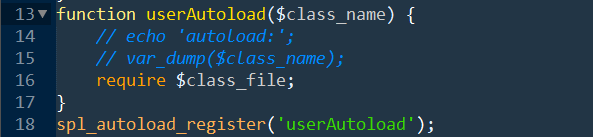
需要通过类名（接口名）能够推导出所在的文件地址！

自定义的自动加载方法（函数）：

\_\_autoload():

1. 如果项目中有多个模块，不能存在多个\_\_autoload().(PHP中的函数名必须唯一的)
2. 一旦代码中注册（spl\_autoload\_register）其他的自动加载方法，则\_\_autoload()自动失效。

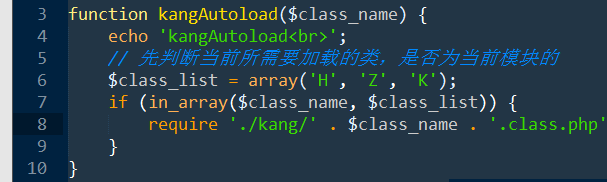
需要自动加载，自己注册：



可以同时注册多个自动加载函数。

而且这些自动加载函数，都是可以被自动调用的。

要求：用户在定义自动加载函数时，必须要判断当前所加载的类，是否是当前自动加载函数，所需要处理的类。



# 魔术方法

特别的方法：

在特定的时机，完成自动调用的一些对象方法。

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 常规目的 |  |
| \_\_construct() | 实例化过程中 | 初始化对象操作 |  |
| \_\_destruct() | 对象被销毁过程中 | 释放对象占用的额外资源，收尾性工作 |  |
| \_\_clone() | 克隆对象 | 重新初始化克隆出来的新对象 |  |
| \_\_toString() | 对象转换成字符串 |  |  |
| \_\_set() | 重载 |  |  |
| \_\_get() | .. |  |  |
| \_\_isset() | .. |  |  |
| \_\_unset() | .. |  |  |
| \_\_call() | .. |  |  |
| Static \_\_callStatic | .. |  |  |
| \_\_sleep() | 序列化过程中 | 指定需要被序列化的对象属性列表 |  |
| \_\_wakeup() | 反序列化过程中。 | 苏醒后的第一件事 | 需要加载对象所属类，也可以自动加载 |
| \_\_invoke() |  |  | PHP匿名函数的实现机制 |

\_\_toString();



# 重载

重新加载，当操作对象的不可访问的成员时，会自动触发一些魔术方法，去处理该情况。

不可访问：

1. 不存在
2. 受限于访问修饰限定符

## 属性重载

重载属性的默认处理：

1. 受限的，就不能处理
2. 不存在的，自动增加



此时用户，需要定制重载的操作，PHP提供魔术方法供使用。

属性重载：

\_\_set(); 重载属性设置，自动调用。

\_\_get(); 重载属性设置，自动调用。

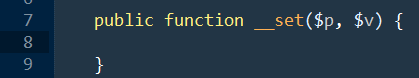
\_\_isset(); 重载属性设置，自动调用。

\_\_unset(); 重载属性设置，自动调用。

例子一：

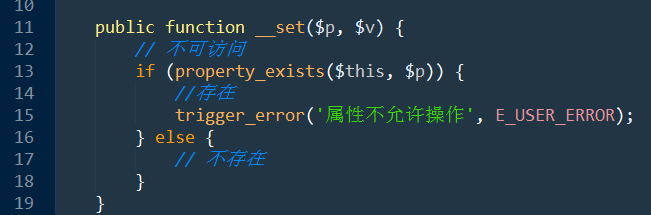
不允许操作不存在的属性：

一个空的\_\_set():



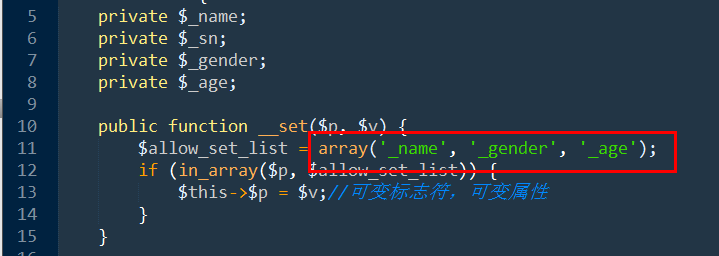
例子一：

不允许操作不存在的属性，如果不可访问，触发一个错误：



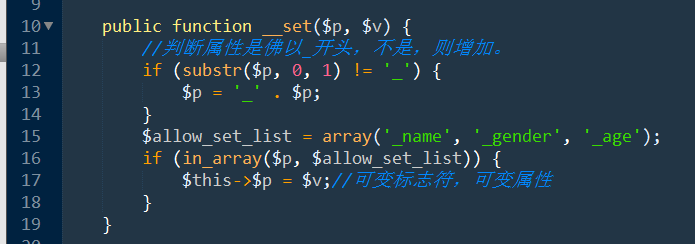
例子三：

允许某些属性可以设置。



例子四：

批量管理同时容错：



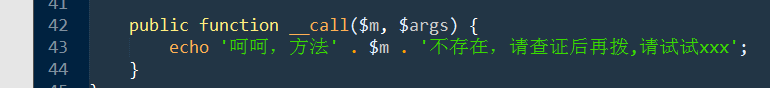
## 方法重载

\_\_call()

Static \_\_callStatic()

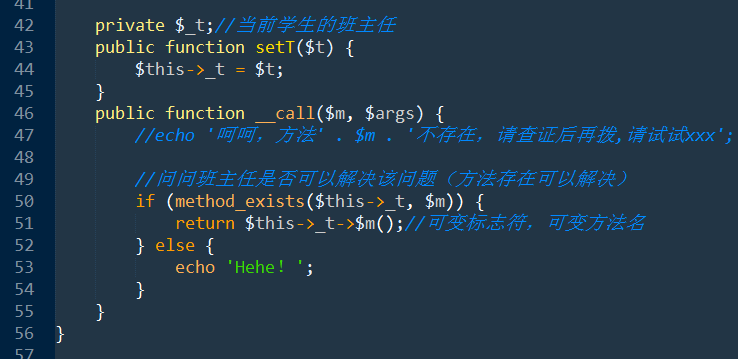
实例一：

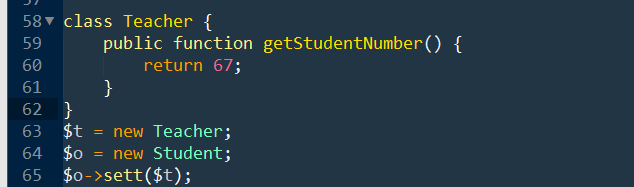
友好的错误处理：



实例二：

一个对象没有解决，另一个对象帮着解决。





问学生：

