继承

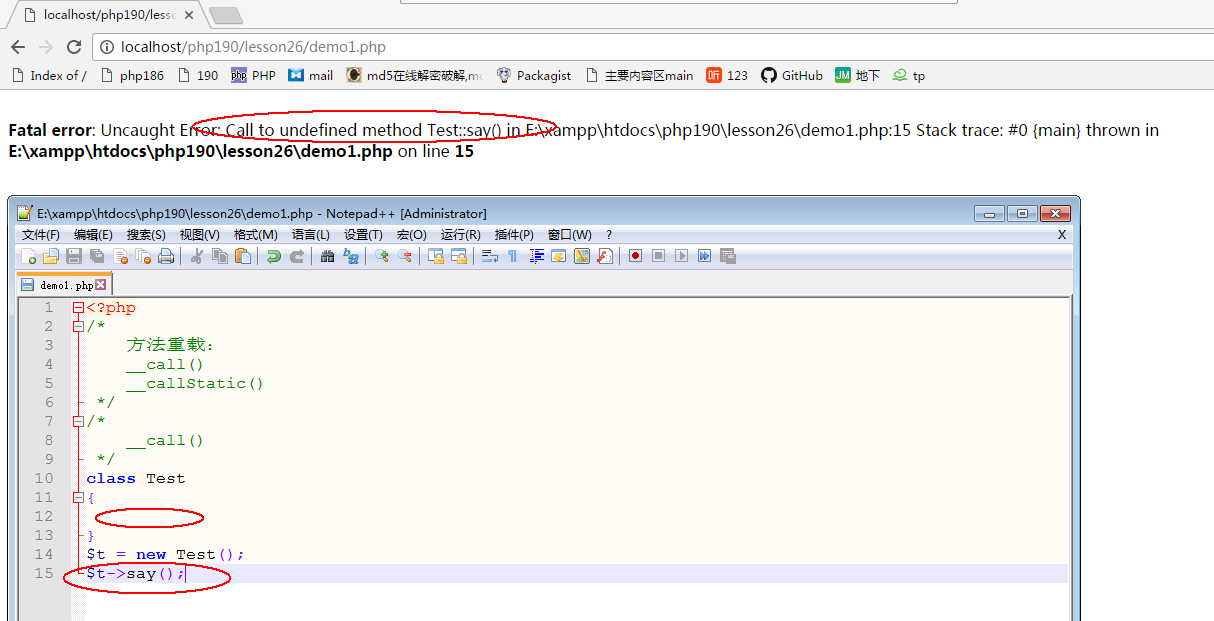
* + 关键字
    - Final
      * 修饰类，方法，表示最终
      * 最终类：不能继承
      * 最终方法：不能重写，允许普通类中的方法是最终方法
    - Static
      * 静态
      * 好处：不用实例化对象，就可以玩类中的属性，方法
      * 注意：静态属性，静态方法都是属于类的，而不是具体某个对象
      * 要求：
        + 静态方法中不能存在$this，如果想玩静态属性，需要self::
      * 调用：
        + 类外：类名::静态属性名|静态方法名()
        + 类内：self::
* 类常量
  + 定义：const 常量名 = ‘常量值’
  + 使用：
    - 类外：类名::常量名
    - 类内：self::常量名
    - 方法和静态属性一样，只不过他是常量，值不能修改，也不能重名
* 检测对象是否属于某个类
  + 对象 Instanceof 类
  + 结果：bool
  + 注意：如果属于子类，肯定属于父类
* 克隆
  + 意义：想复制对象，但是对象之间没关系
  + 如果使用常规的引用赋值，直接赋值，结果对象都是一个
  + $a = clone $b
  + 默认，浅克隆，也就是说：只复制外层的对象
  + 魔术方法：
    - \_\_clone()
      * 不需要传值，也没有返回值
      * 特点：使用clone时自动调用
      * 可以在克隆的时候对值进行操作
* 自动加载
  + 特点：不需要挨个把需要的类做require,而是指定路径，然后php自己找，自己引入
  + 方法1：
    - \_\_autoload()
  + 方法2：
    - Spl\_autoload\_register()

# 方法重载

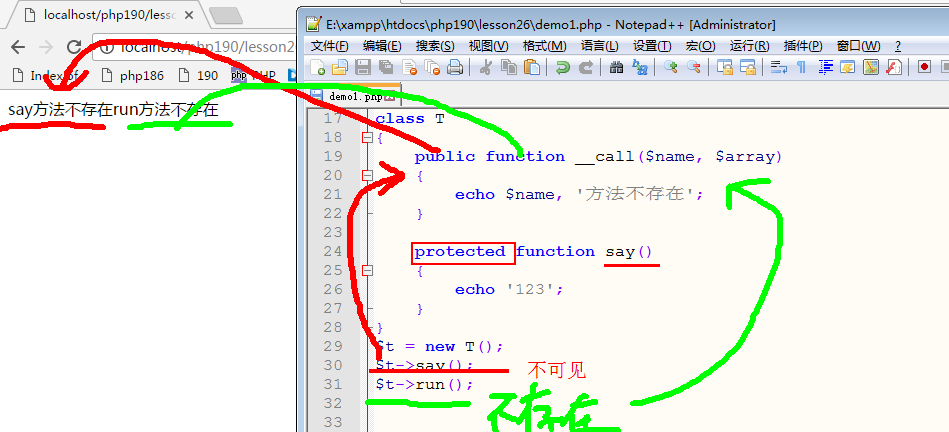


## \_\_call(方法名，参数数组)

* 在对象中调用不存在的方法，自动调用

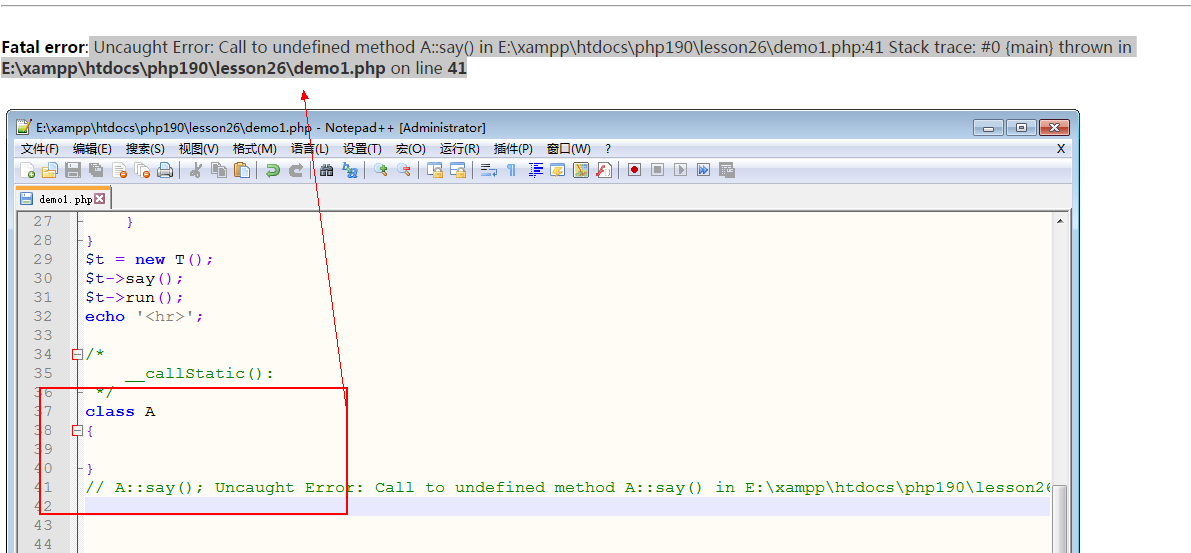


* 注意：
  + \_\_call()中的函数体其实是固定的，只要是不可见或者不存在的方法被调用时，都会自动调用，结果无非就是说：方法不存在啦，你掉了XX方法，他的参数是XX。这么做，看起来没什么意义，但是起码不会给你一脸的error

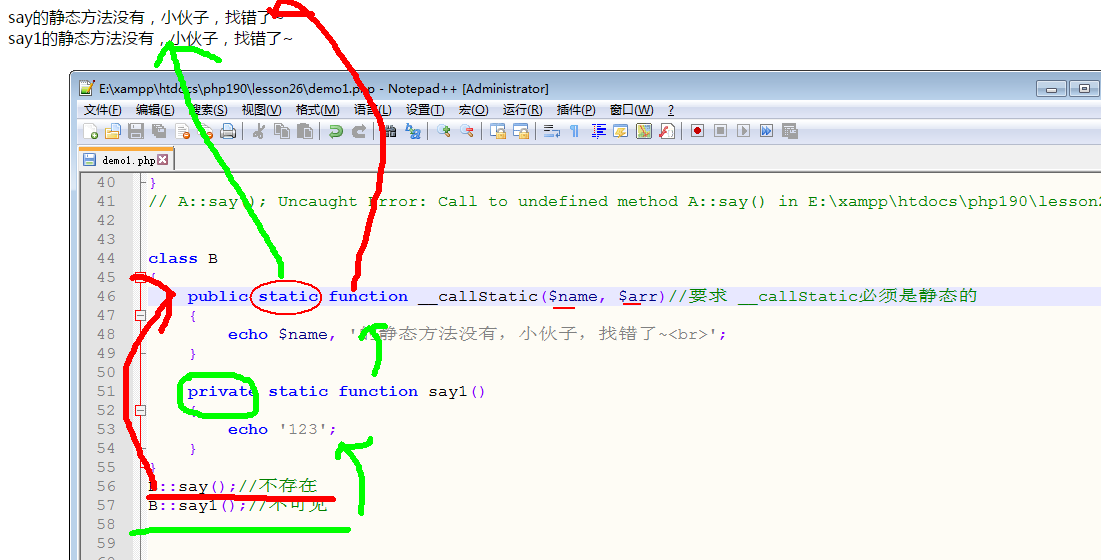


## \_\_callStatic(静态方法名，参数数组)

* 和\_\_call()一样的操作，只不过此处调用的是静态方法而已



* 示例：



# 序列化

* 说白了：就是把别的类型转换为字符串型，用来做数据的存储和传输。现在用的多不多？还真不多，不过可以作为替代方法。更多的是json

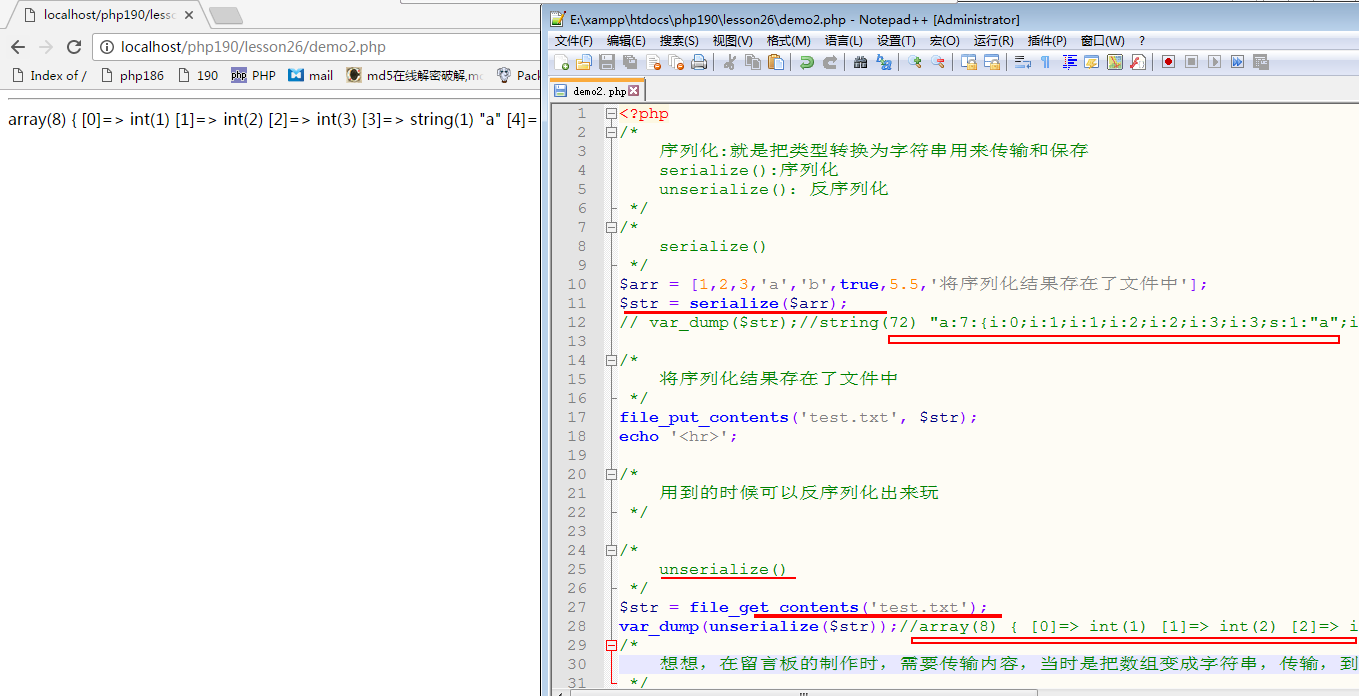
## Serialize():序列化



### Unserialize()：反序列化



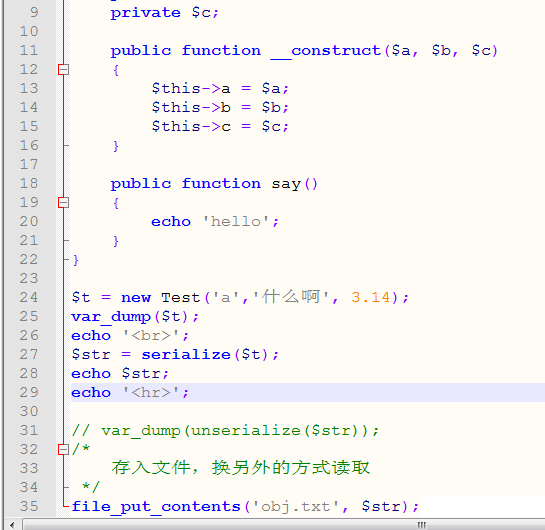
示例1:数组序列化，数组中有：所有标量还带中文



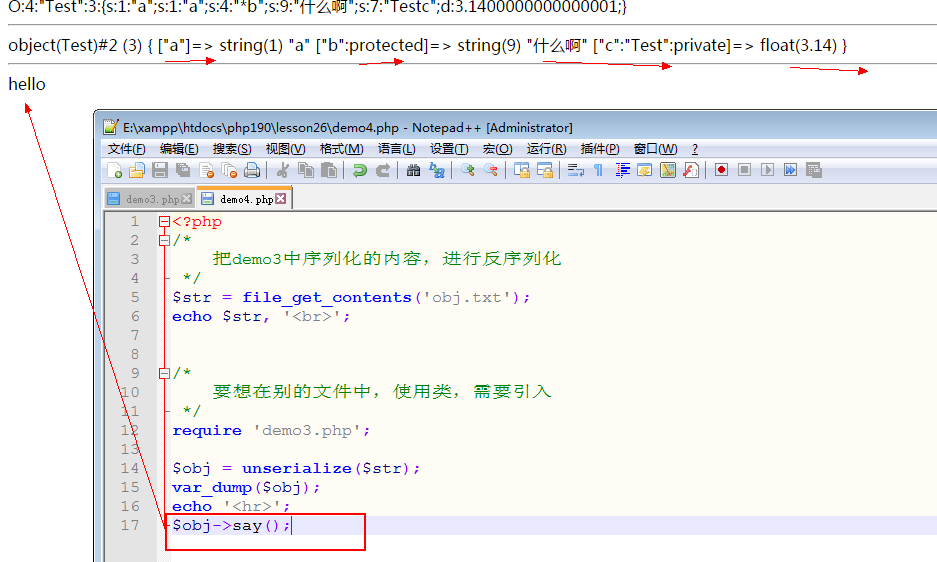
示例2：对象的序列化

**注意：对象的序列化，只会存在所有的属性，而不会存储方法**

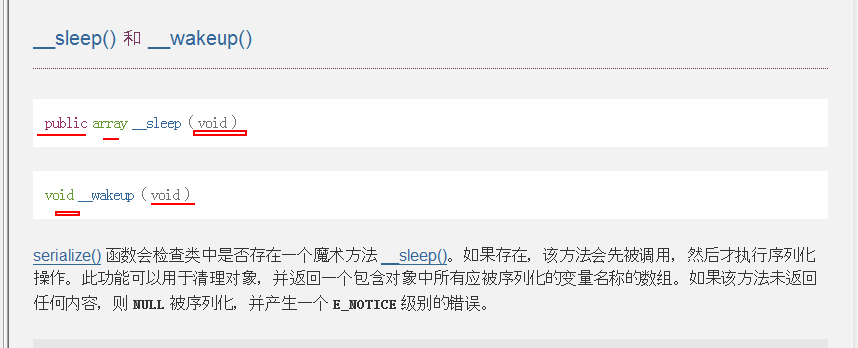
* 1.序列化



2.反序列化

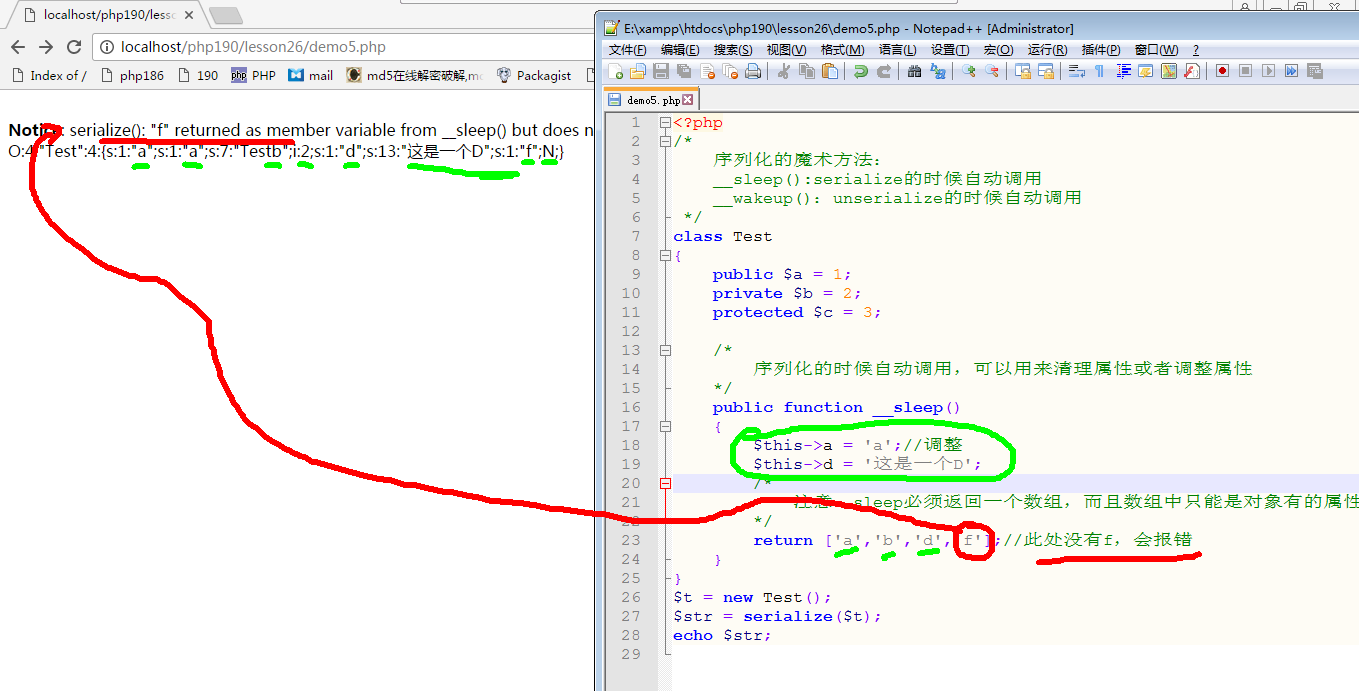


## 魔术方法



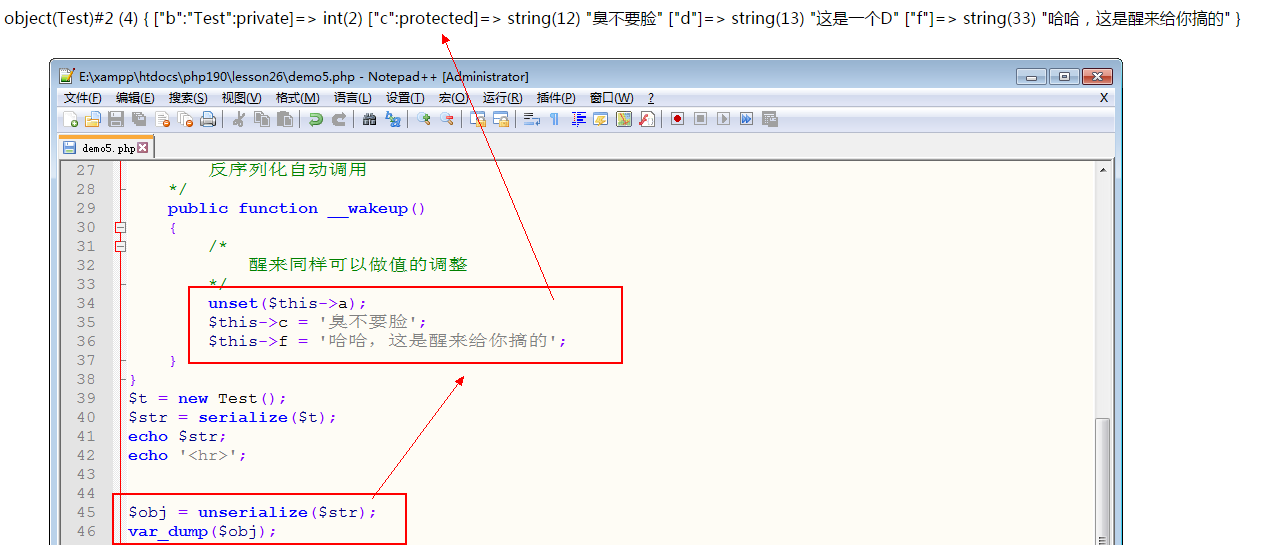
### \_\_sleep()

* 注意：
  + **1.不需要传值**
  + **2，必须返回数组，并且，数组中的值只能是对象有的属性**
  + **3，可以在内部进行值的调整**



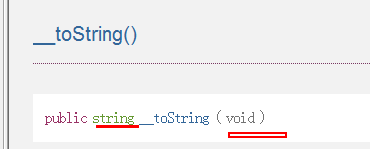
### \_\_wakeup()

* 注意：
  + **1.不需要传值**
  + **2.没有返回值**
  + **3.同样可以对属性做值的调整**

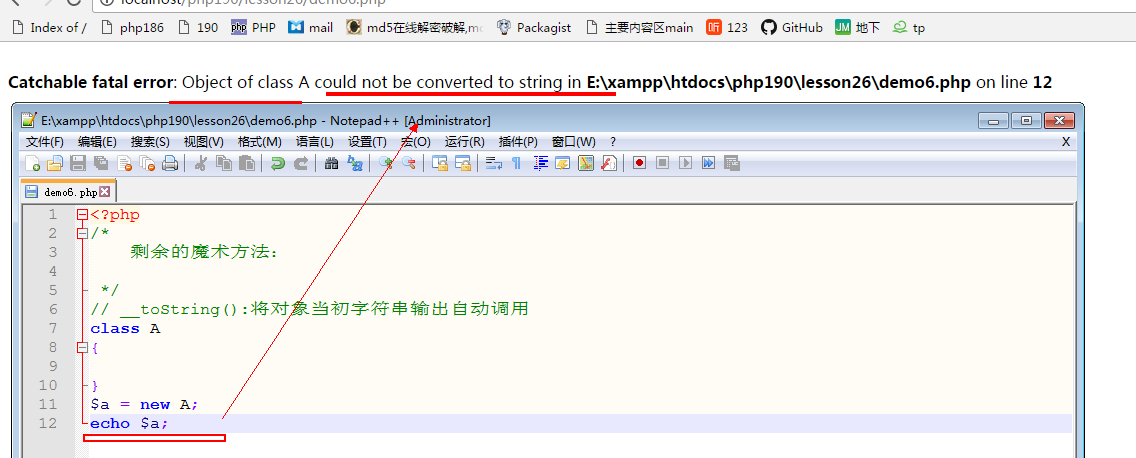


# 剩余的魔术方法

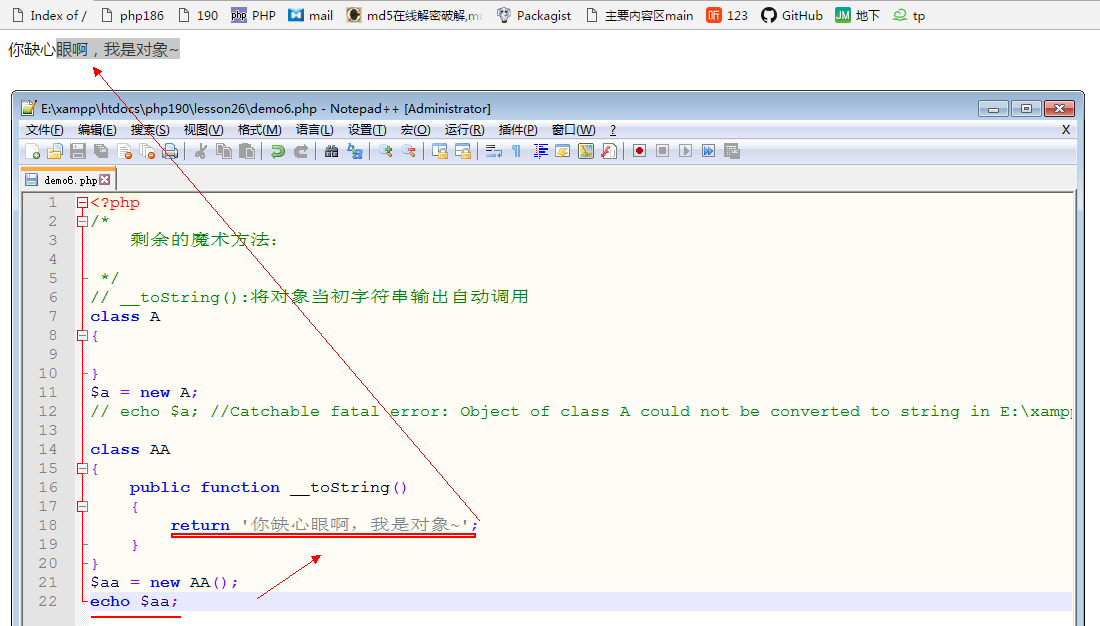
## \_\_toString()



* 将对象当字符串输出自动调用
* 示例：
  + 默认echo 对象会报错：不能把对象当字符串一样输出

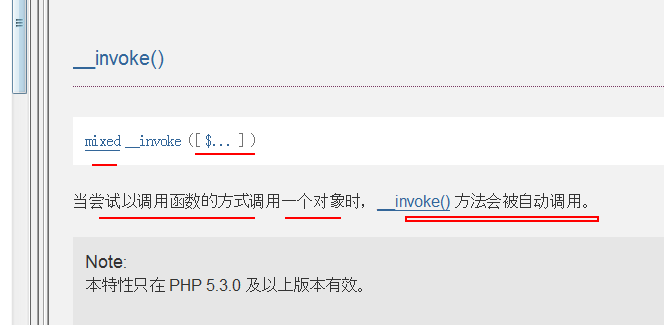


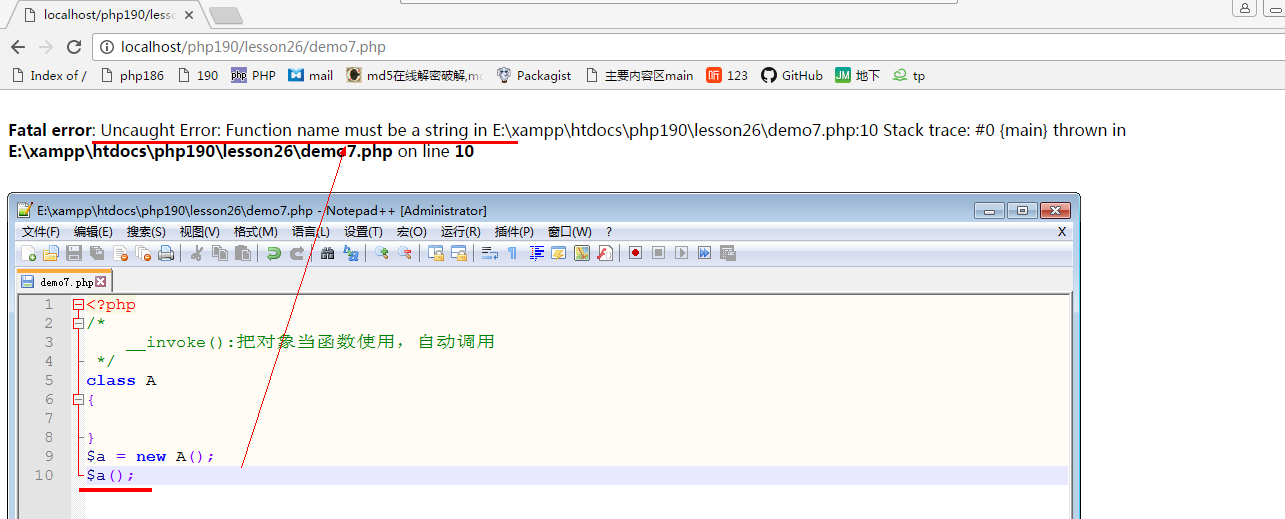
* 正确的示例



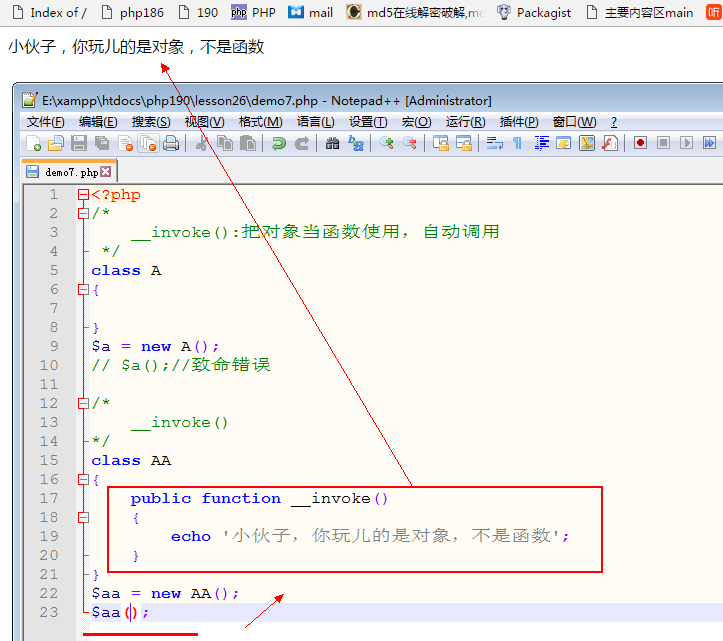
## \_\_invoke()

* 将对象当函数调用自动调用



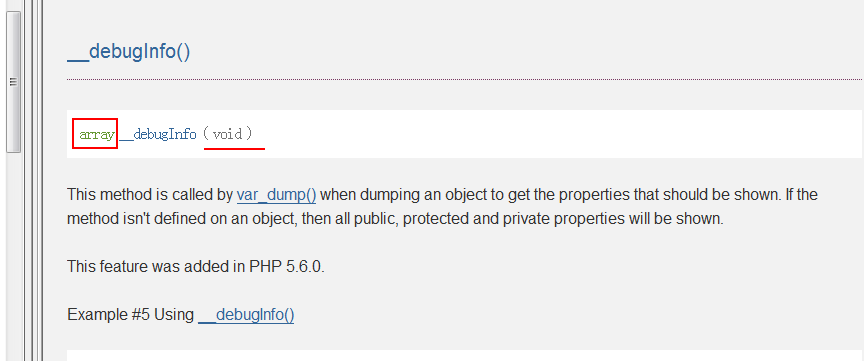


示例：

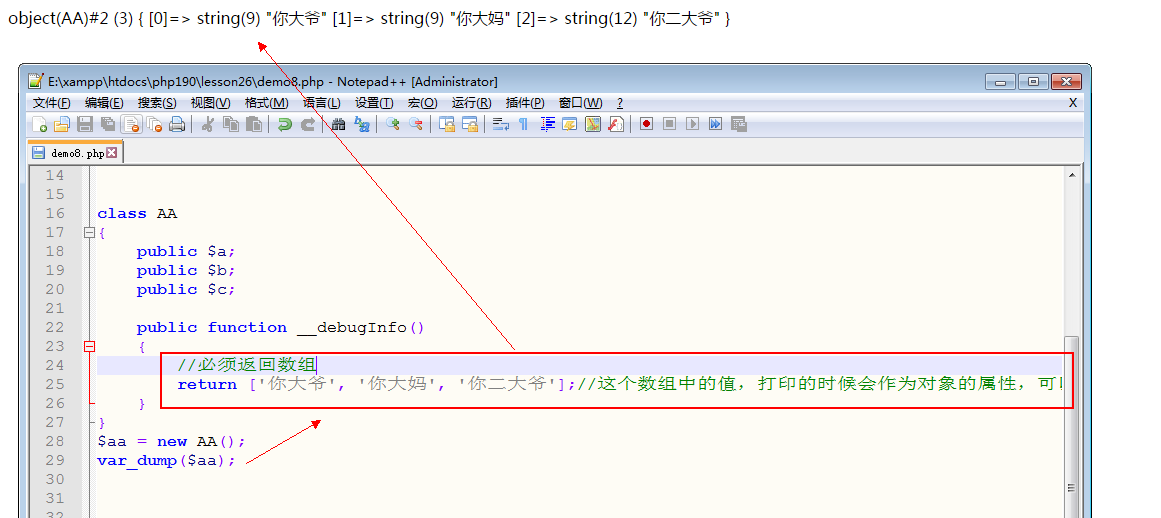


## \_\_debugInfo()

* Var\_dump()对象的时候自动调用



示例：

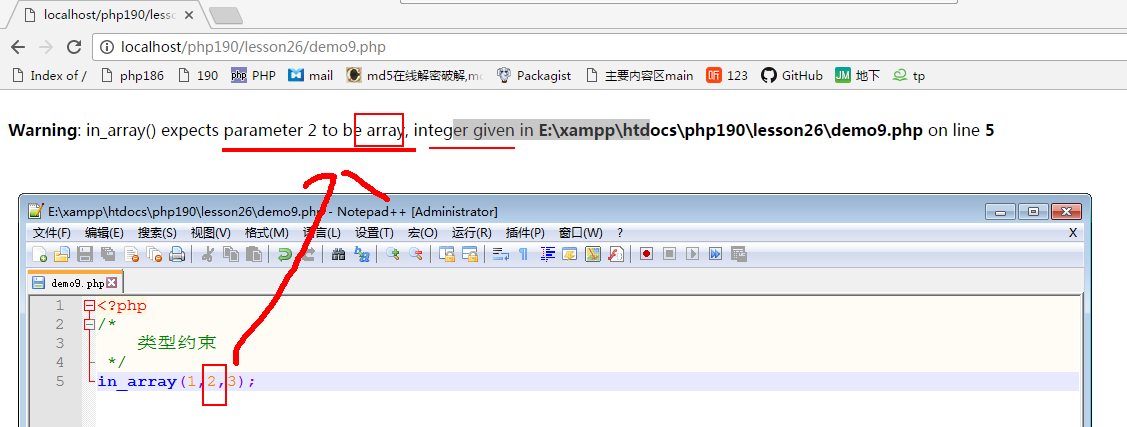


# 类型约束

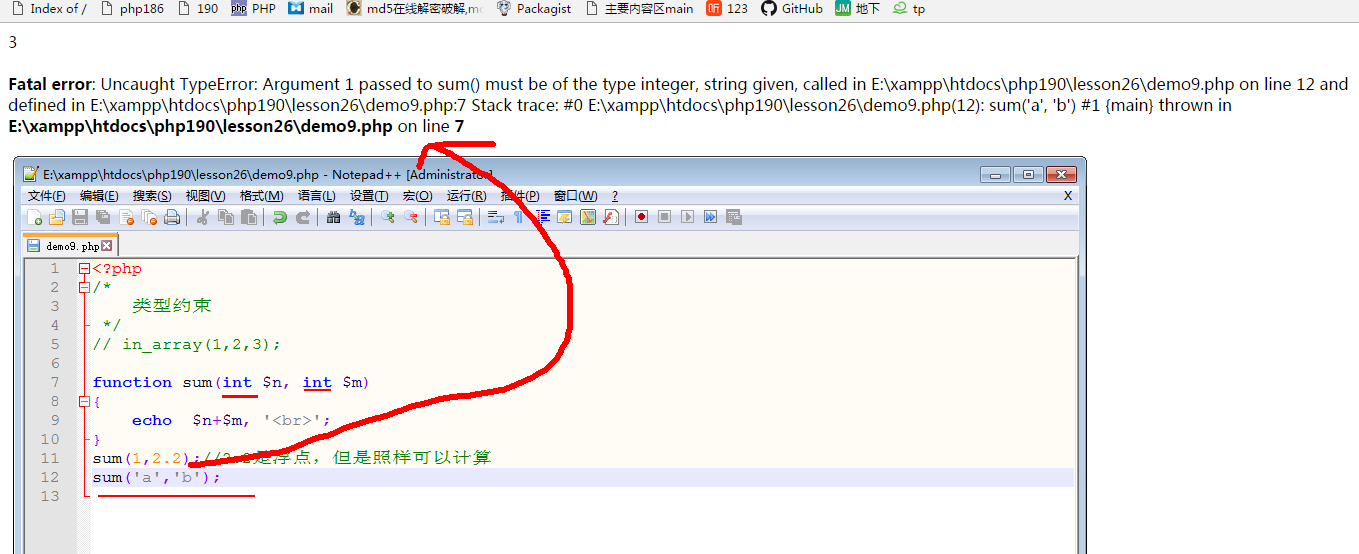
* **类型约束起始很早就见过了。在函数中，参数列表的前面都有个数据类型，这个类型就表示传入的参数，必须是对应的类型，否则会出错。在PHP7之前，只能是非标量的类型约束，7之后，允许所有类型的约束**



* **涉及3个伪类型**
  + **Number:数字，也就是整型+浮点**
  + **Callback：回调函数**
  + **Mixed:任意类型，但不是所有的类型**

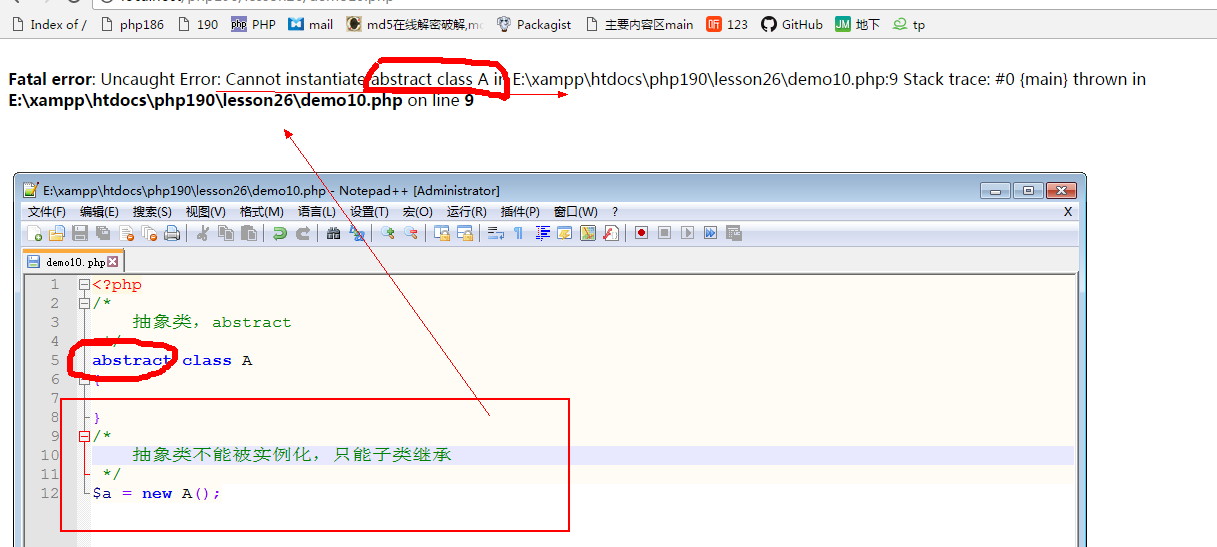


**自定义的类型约束**

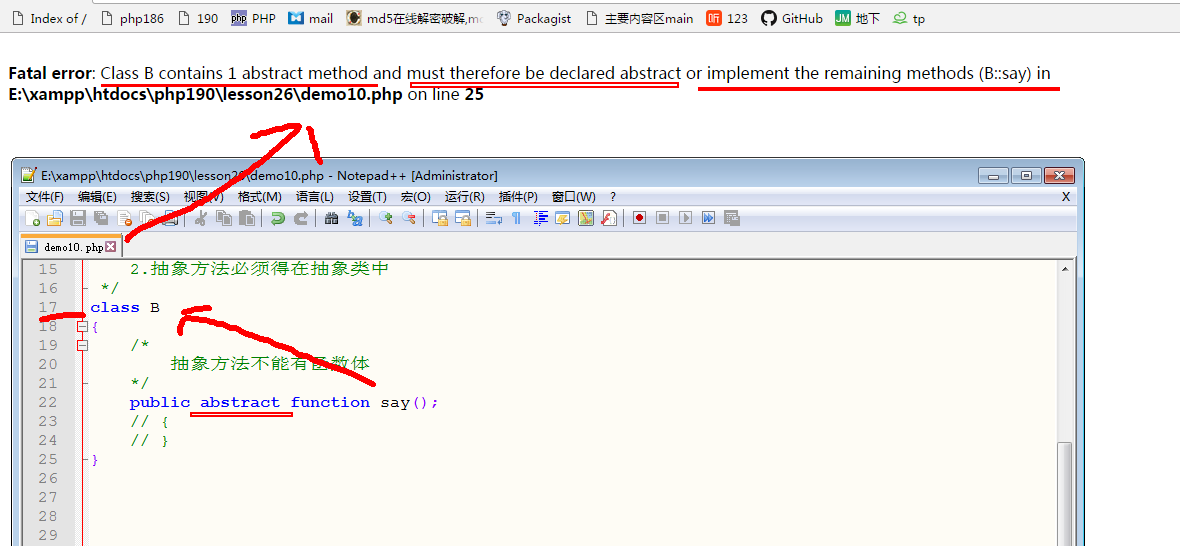


# 抽象类abstract

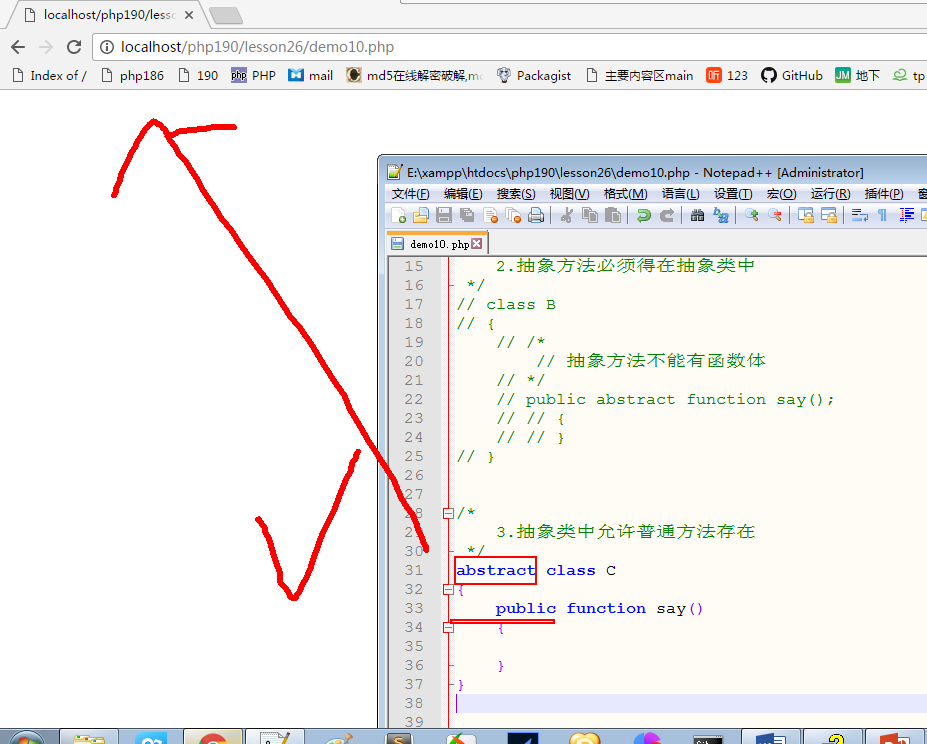
* 抽象类其实就是对类进行抽象
* 关键词 ： abstract
* 作用范围：类，方法
* 特点：
  + **1.抽象类不能实例化**



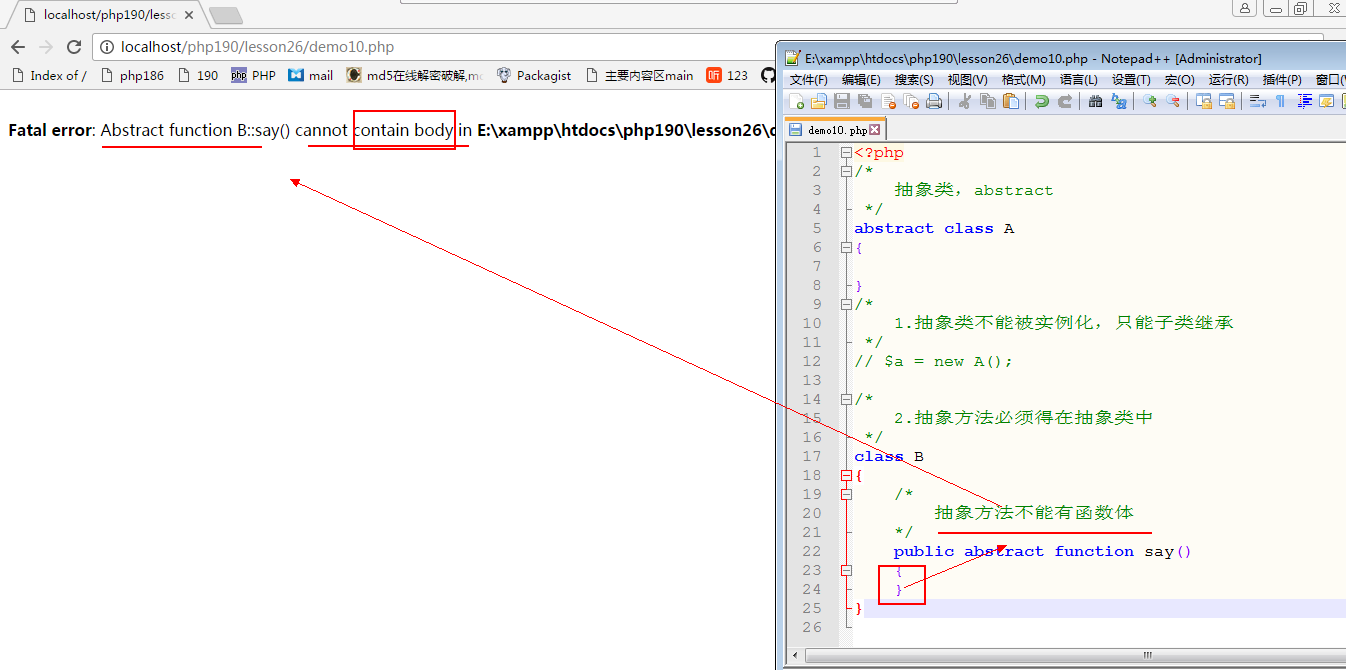
* + **2.抽象方法必须得在抽象类中，只要有抽象方法的类就是抽象类，反过来，所有的抽象类一定有抽象方法吗？不是的**
    - **抽象方法必须得声明在抽象类中**



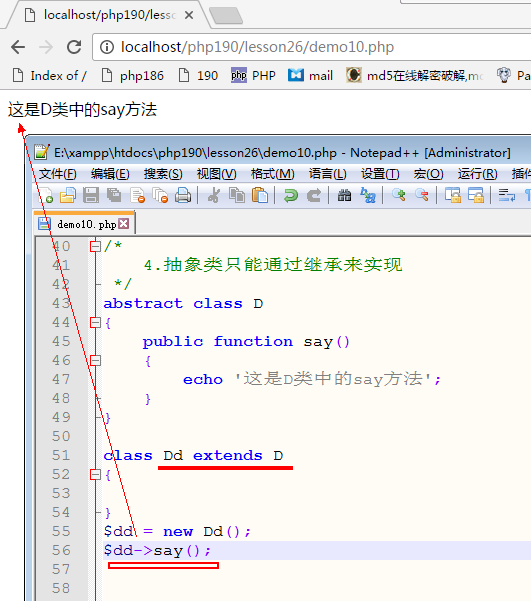
* **抽象类中允许普通方法存在**



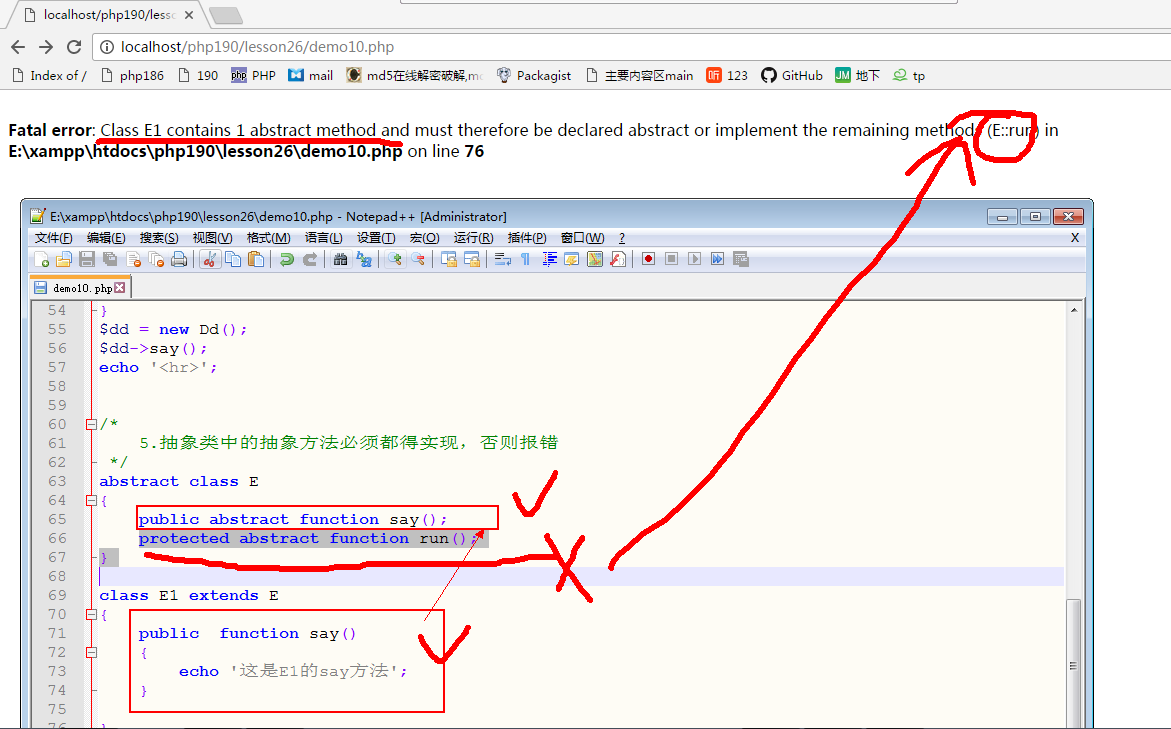
* + **3.抽象方法不能有函数体,而且后面必须有分号**



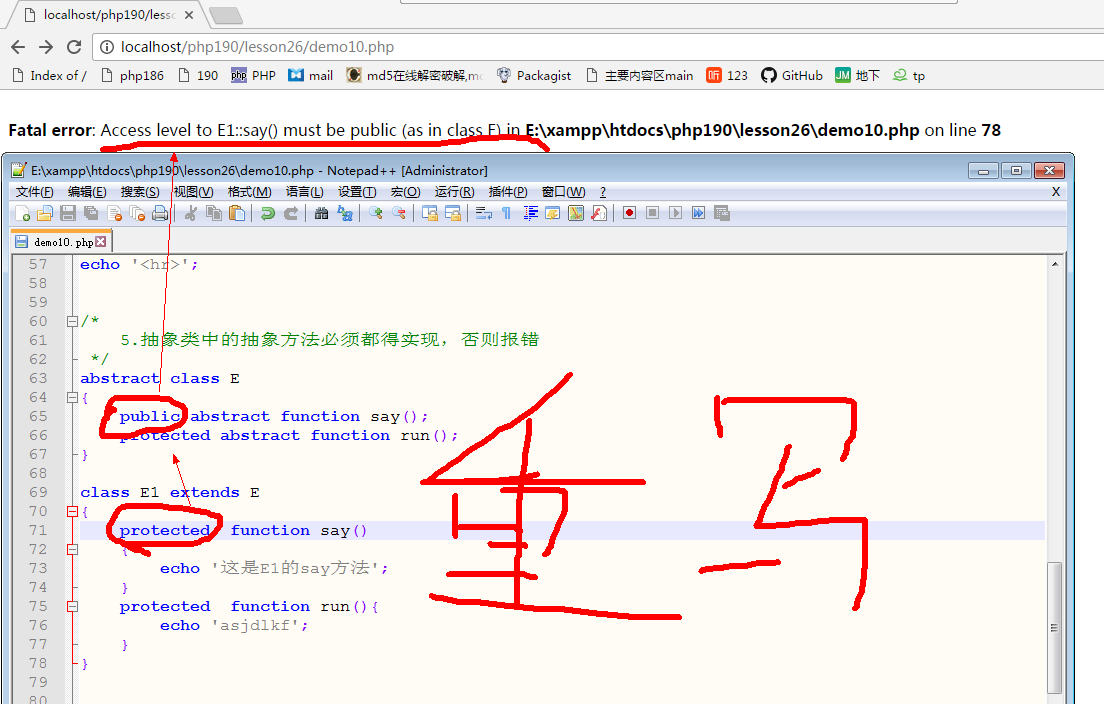
* + **4.抽象类只能通过子类继承来实例化**



* + **5.抽象类中的所有的抽象方法必须都得实现，否则报错**

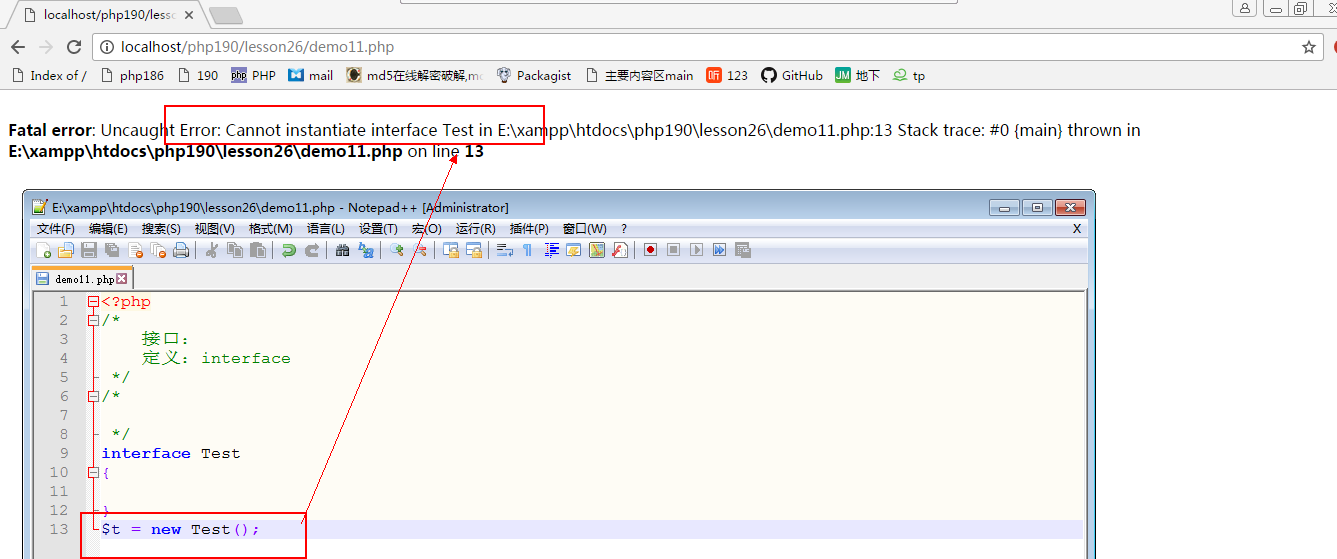


* + **6.子类实现抽象类的方法，也就是重写的时候，遵守重写的规则。1.方法名一致，2，参数数目一致，3，访问的范围子>=父**

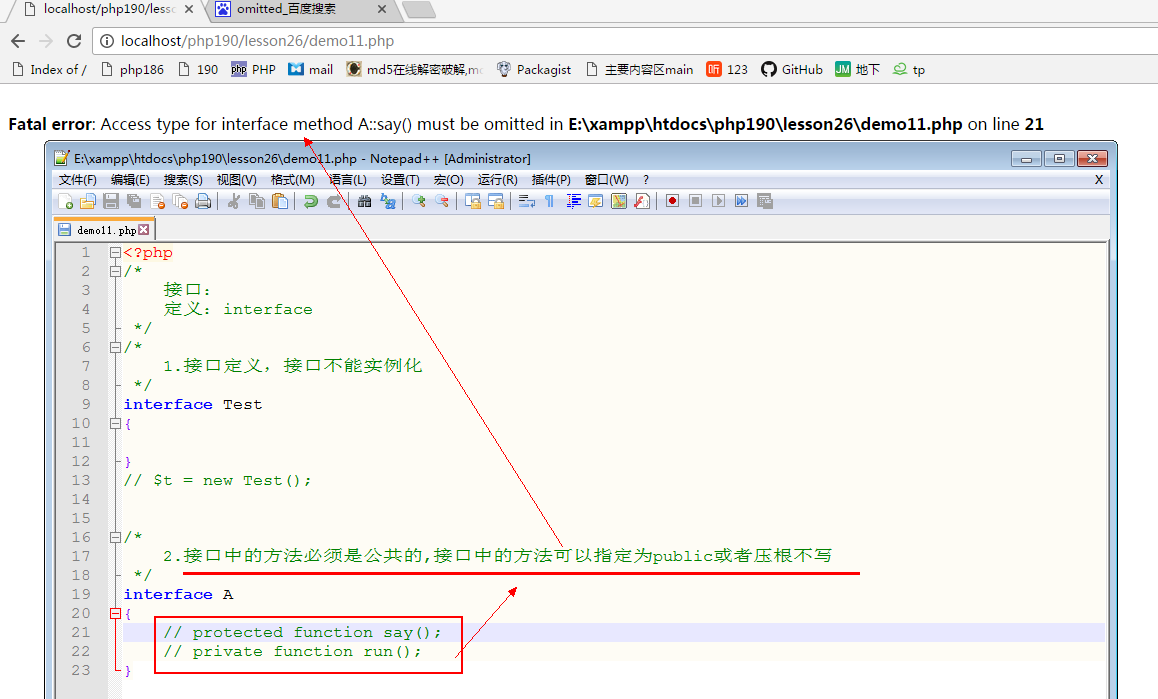


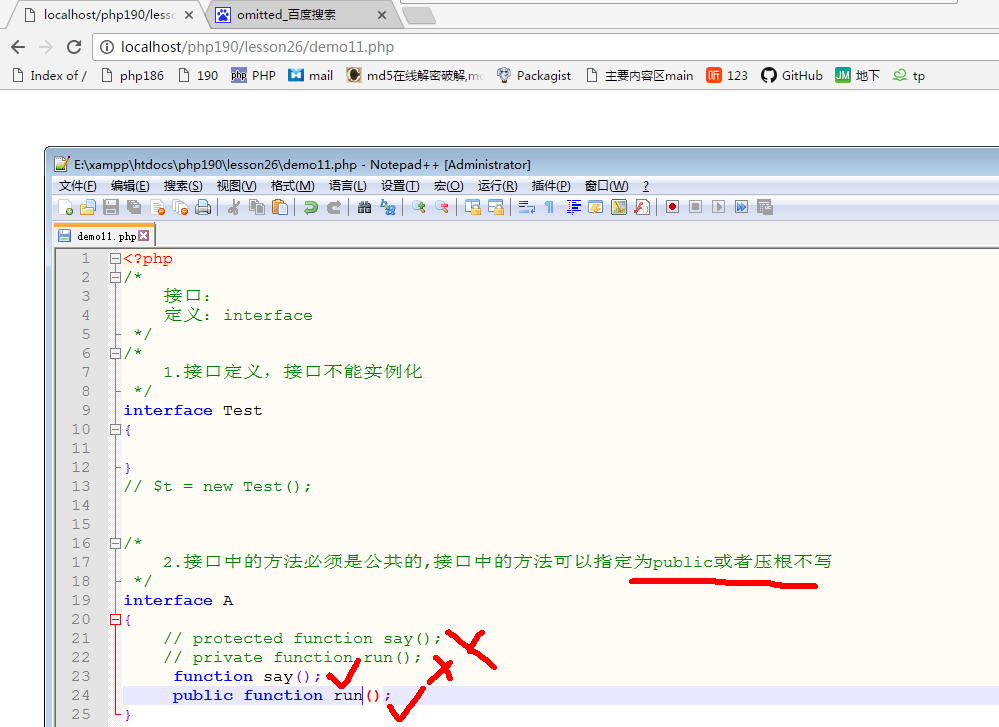
# 接口

* 定义：
  + 比抽象类更抽象的类，就是接口
* 用处：
  + 说白了，就是通过接口来定一个大纲，然后，别的类实现大纲，就实现了功能，给一个规范，或者一个指导性的意见
* 特点：
  + **1.关键词 interface,替代class**
    - **接口同样不能直接实例化，而是需要实现**

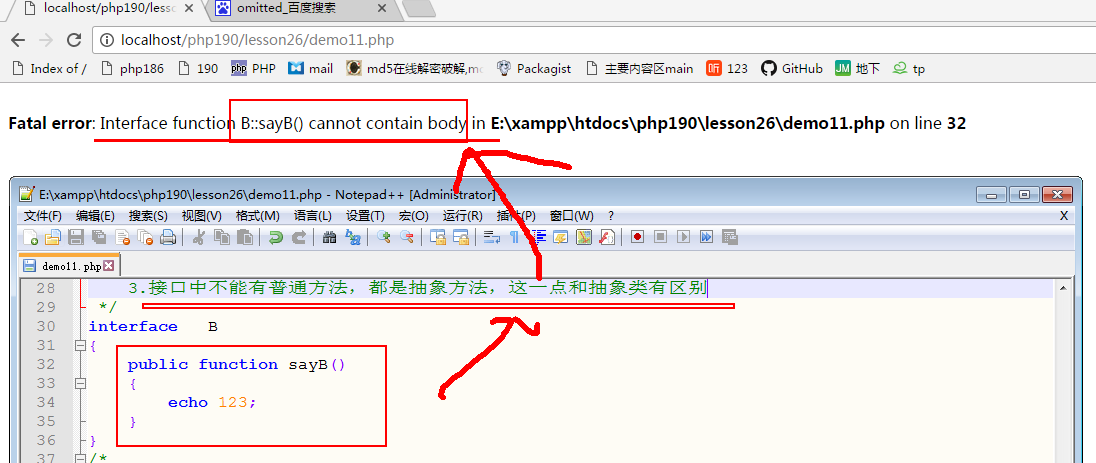


* + **2.接口中的方法，必须是公共的。或者不写**





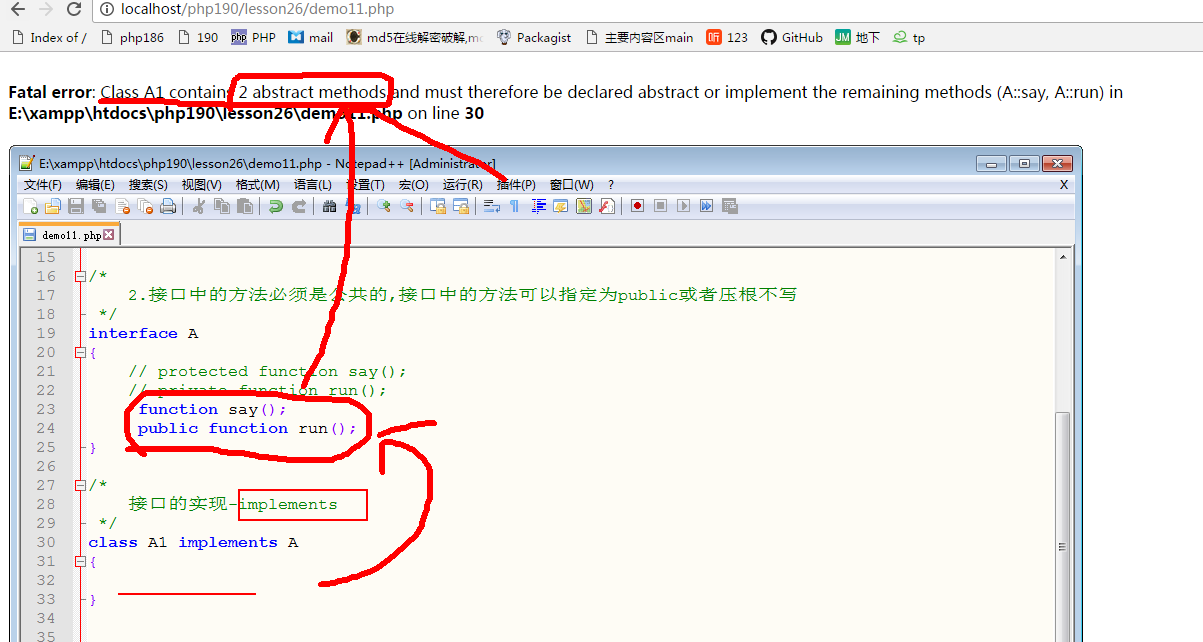
* + **3.接口中能不能有普通方法？不能，必须都得是抽象方法**



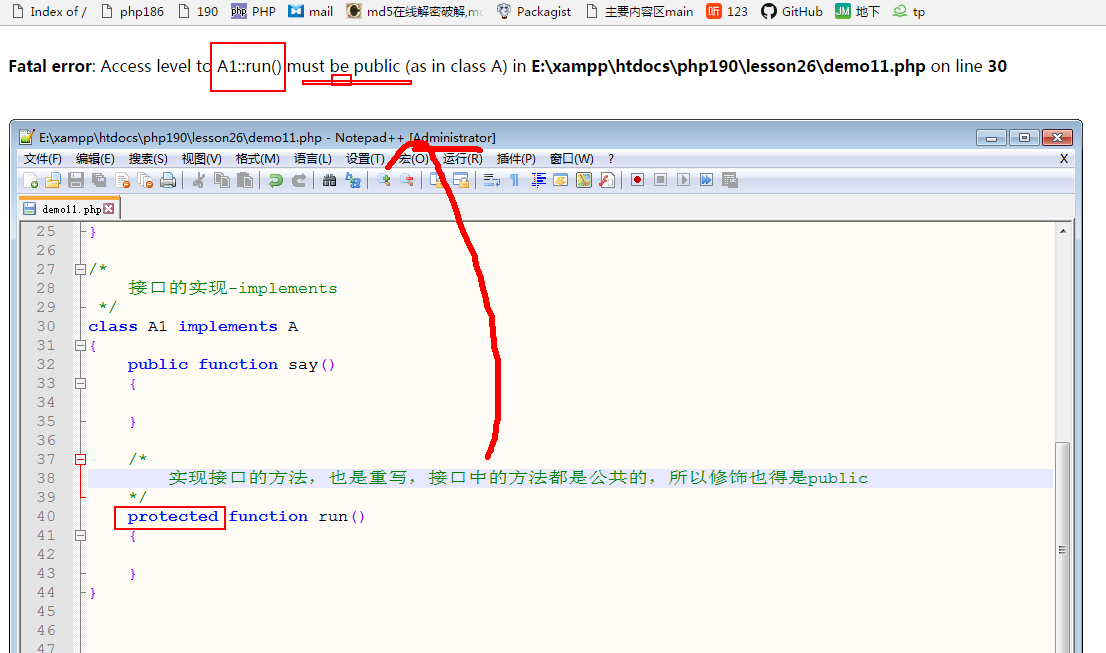
* + **4.接口必须得通过实现（继承）来完成对应的功能**

## 接口的实现-implements

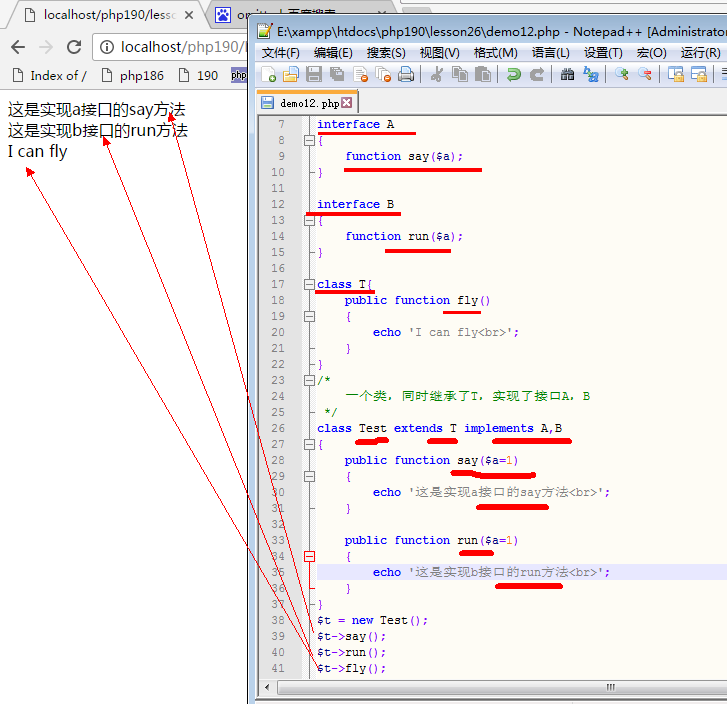
* **1.接口中定义的方法，必须都得实现**



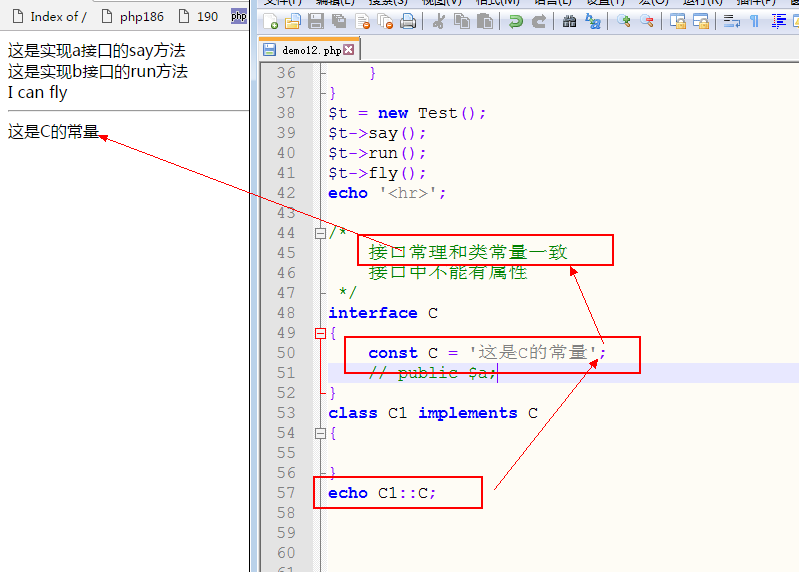
* **2.其他类实现接口时，必须得用public修饰**



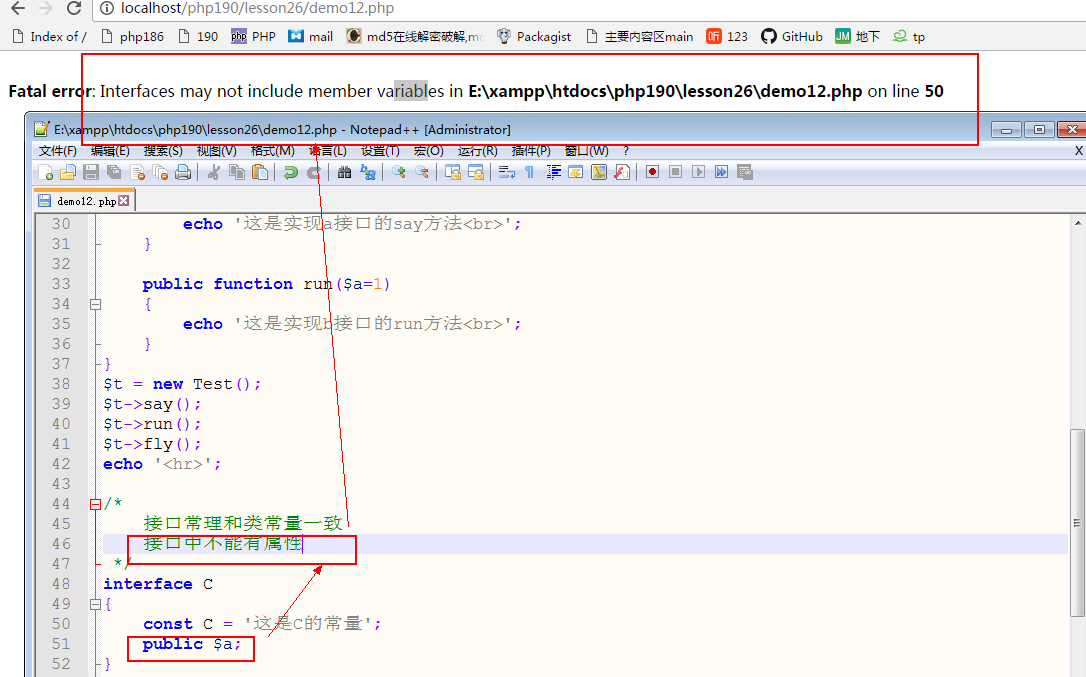
* **3.可以同时实现多个接口，多个接口之间使用逗号隔开**



* **4.如果一次性实现多个接口，接口中的方法不能重名**
* **5.接口当中允许有接口常量，方法，**
  + **接口常量和类常量一致，不能重名，定义方式也一样**



* + **能否有属性？不能有属性**



### 小结：

* 抽象类和接口对比：
  + **1.抽象类允许有普通方法存在，接口不允许**
  + **2.抽象类中的方法允许是任意类型，接口只能是公共**
  + **3.抽象类的实现通过继承，接口的实现通过实现来实现**
  + **4.一个子类只能继承一个抽象类，但是可以实现多个接口**
  + **5.抽象类也好，接口也好，对方法的实现其实都遵守重写的规则：**
    - **1.范围**
    - **2.参数数目**