1、__construct

1、__construct()，类的构造函数
2、__destruct()，类的析构函数
3、__call()，在对象中调用一个不可访问方法时调用
4、__callStatic()，用静态方式中调用一个不可访问方法时调用
5、__get()，获得一个类的成员变量时调用
6、__set()，设置一个类的成员变量时调用
7、__isset()，当对不可访问属性调用isset()或empty()时调用
8、__unset()，当对不可访问属性调用unset()时被调用。
9、__sleep()，执行serialize()时，先会调用这个函数
10、__wakeup()，执行unserialize()时，先会调用这个函数
11、__toString()，类被当成字符串时的回应方法
12、__invoke()，调用函数的方式调用一个对象时的回应方法
13、__set_state()，调用var_export()导出类时，此静态方法会被调用。
14、__clone()，当对象复制完成时调用
15、__autoload()，尝试加载未定义的类
16、__debugInfo()，打印所需调试信息

一、 __construct()，类的构造函数

php中构造方法是对象创建完成后第一个被对象自动调用的方法。在每个类中都有一个构造方法，如果没有显示地声明它，那么类中都会默认存在一个没有参数且内容为空的构造方法。

1、 构造方法的作用

通常构造方法被用来执行一些有用的初始化任务，如对成员属性在创建对象时赋予初始值。

2、 构造方法的在类中的声明格式

```
function __constrct([参数列表]){

  方法体 //通常用来对成员属性进行初始化赋值
}
```

3、 在类中声明构造方法需要注意的事项

1、在同一个类中只能声明一个构造方法，原因是，PHP不支持构造函数重载。

2、构造方法名称是以两个下画线开始的__construct()

二、__destruct()，类的析构函数

通过上面的讲解，现在我们已经知道了什么叫构造方法。那么与构造方法对应的就是析构方法。

析构方法允许在销毁一个类之前执行的一些操作或完成一些功能，比如说关闭文件、释放结果集等。

析构方法是PHP5才引进的新内容。

析造方法的声明格式与构造方法 __construct() 比较类似，也是以两个下划线开始的方法 __destruct() ，这种析构方法名称也是固定的。

1、 析构方法的声明格式

```
function __destruct()
{
 //方法体
}
```

注意：析构函数不能带有任何参数。

三、 __call()，在对象中调用一个不可访问方法时调用。

该方法有两个参数，第一个参数 $function_name 会自动接收不存在的方法名，第二个 $arguments 则以数组的方式接收不存在方法的多个参数。

1、 __call() 方法的格式：

?

```
function __call(string $function_name, array $arguments)
{
  // 方法体
}
```

2、 __call() 方法的作用：

为了避免当调用的方法不存在时产生错误，而意外的导致程序中止，可以使用 __call() 方法来避免。

该方法在调用的方法不存在时会自动调用，程序仍会继续执行下去。

四、 __callStatic()，用静态方式中调用一个不可访问方法时调用

此方法与上面所说的 __call() 功能除了 __callStatic() 是未静态方法准备的之外，其它都是一样的。

请看下面代码：

```php
<?php
class Person
{
  function say()
  {

    echo "Hello, world!<br>";
  }

  /**
   * 声明此方法用来处理调用对象中不存在的方法
   */
  public static function __callStatic($funName, $arguments)
  {
    echo "你所调用的静态方法：" . $funName . "(参数：" ; // 输出调用不存在的方法名
    print_r($arguments); // 输出调用不存在的方法时的参数列表
    echo ")不存在! <br>\n"; // 结束换行
  }
}
$Person = new Person();
$Person::run("teacher"); // 调用对象中不存在的方法，则自动调用了对象中的__call()方法
$Person::eat("小明", "苹果");
$Person->say();
```

运行结果如下：

你所调用的静态方法：run(参数：Array ([0] => teacher))不存在!

你所调用的静态方法：eat(参数：Array ([0] => 小明 [1] => 苹果))不存在!

Hello, world!

五、 __get()，获得一个类的成员变量时调用

在 php 面向对象编程中，类的成员属性被设定为 private 后，如果我们试图在外面调用它则会出现“不能访问某个私有属性”的错误。那么为了解决这个问题，我们可以使用魔术方法 __get()。

魔术方法__get()的作用

在程序运行过程中，通过它可以在对象的外部获取私有成员属性的值。

我们通过下面的 __get() 的实例来更进一步的连接它吧：

```php
<?php
class Person
{
  private $name;
  private $age;

  function __construct($name="", $age=1)
  {
    $this->name = $name;
    $this->age = $age;
  }

  /**
   * 在类中添加__get()方法，在直接获取属性值时自动调用一次，以属性名作为参数传入并处理
   * @param $propertyName
   *
   * @return int
   */
  public function __get($propertyName)
  {
    if ($propertyName == "age") {
      if ($this->age > 30) {
        return $this->age - 10;
      } else {
        return $this->$propertyName;
      }
    } else {
      return $this->$propertyName;
    }
  }
}
$Person = new Person("小明", 60);  // 通过Person类实例化的对象，并通过构造方法为属性赋初值
echo "姓名：" . $Person->name . "<br>";  // 直接访问私有属性name，自动调用了__get()方法可以间接获取
echo "年龄：" . $Person->age . "<br>";  // 自动调用了__get()方法，根据对象本身的情况会返回不同的值
```

运行结果：

姓名：小明

年龄：50

六、 __set()，设置一个类的成员变量时调用

__set() 的作用：

__set($property, $value)` 方法用来设置私有属性， 给一个未定义的属性赋值时，此方法会被触发，传递的参数是被设置的属性名和值。

请看下面的演示代码：

?

```php
<?php
class Person
{
  private $name;
  private $age;

  public function __construct($name="", $age=25)
  {
    $this->name = $name;
    $this->age = $age;
  }

  /**
   * 声明魔术方法需要两个参数，真接为私有属性赋值时自动调用，并可以屏蔽一些非法赋值
   * @param $property
   * @param $value
   */
  public function __set($property, $value) {
    if ($property=="age")
    {
      if ($value > 150 || $value < 0) {
        return;
      }
    }
    $this->$property = $value;
  }

  /**
   * 在类中声明说话的方法，将所有的私有属性说出
   */
  public function say(){
    echo "我叫".$this->name."，今年".$this->age."岁了";
  }
}

$Person=new Person("小明", 25); //注意，初始值将被下面所改变
//自动调用了__set()函数，将属性名name传给第一个参数，将属性值”李四”传给第二个参数
$Person->name = "小红";   //赋值成功。如果没有__set()，则出错。
//自动调用了__set()函数，将属性名age传给第一个参数，将属性值26传给第二个参数
$Person->age = 16; //赋值成功
$Person->age = 160; //160是一个非法值，赋值失效
$Person->say(); //输出：我叫小红，今年16岁了
```

运行结果：

我叫小红，今年16岁了

七、 __isset()，当对不可访问属性调用isset()或empty()时调用

在看这个方法之前我们看一下isset()函数的应用，isset()是测定变量是否设定用的函数，传入一个变量作为参数，如果传入的变量存在则传回true，否则传回false。

那么如果在一个对象外面使用isset()这个函数去测定对象里面的成员是否被设定可不可以用它呢？

分两种情况，如果对象里面成员是公有的，我们就可以使用这个函数来测定成员属性，如果是私有的成员属性，这个函数就不起作用了，原因就是因为私有的被封装了，在外部不可见。那么我们就不可以在对象的外部使用isset()函数来测定私有成员属性是否被设定了呢？当然是可以的，但不是一成不变。你只要在类里面加上一个__isset()方法就可以了，当在类外部使用isset()函数来测定对象里面的私有成员是否被设定时，就会自动调用类里面的__isset()方法了帮我们完成这样的操作。

__isset()的作用：当对不可访问属性调用 isset() 或 empty() 时，__isset() 会被调用。

请看下面代码演示：

```php
<?php
class Person
{
  public $sex;
  private $name;
  private $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  /**
   * @param $content
   *
   * @return bool
   */
  public function __isset($content) {
    echo "当在类外部使用isset()函数测定私有成员{$content}时，自动调用<br>";
    echo isset($this->$content);
  }
}

$person = new Person("小明", 25); // 初始赋值
echo isset($person->sex),"<br>";
echo isset($person->name),"<br>";
echo isset($person->age),"<br>";
```

运行结果如下：

1 // public 可以 isset()

当在类外部使用isset()函数测定私有成员name时，自动调用 // __isset() 内 第一个echo

1 // __isset() 内第二个echo

当在类外部使用isset()函数测定私有成员age时，自动调用 // __isset() 内 第一个echo

1 // __isset() 内第二个echo

八、 __unset()，当对不可访问属性调用unset()时被调用。

看这个方法之前呢，我们也先来看一下 unset() 函数，unset()这个函数的作用是删除指定的变量且传回true，参数为要删除的变量。

那么如果在一个对象外部去删除对象内部的成员属性用unset()函数可以吗？

这里自然也是分两种情况：

1、 如果一个对象里面的成员属性是公有的，就可以使用这个函数在对象外面删除对象的公有属性。

2、 如果对象的成员属性是私有的，我使用这个函数就没有权限去删除。

虽然有以上两种情况，但我想说的是同样如果你在一个对象里面加上__unset()这个方法，就可以在对象的外部去删除对象的私有成员属性了。在对象里面加上了__unset()这个方法之后，在对象外部使用“unset()”函数删除对象内部的私有成员属性时，对象会自动调用__unset()函数来帮我们删除对象内部的私有成员属性。

请看如下代码：

2

```php
<?php
class Person
{
  public $sex;
  private $name;
  private $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  /**
   * @param $content
   *
   * @return bool
   */
  public function __unset($content) {
    echo "当在类外部使用unset()函数来删除私有成员时自动调用的<br>";
    echo isset($this->$content);
  }
}

$person = new Person("小明", 25); // 初始赋值
unset($person->sex);
unset($person->name);
unset($person->age);
```

运行结果：

当在类外部使用unset()函数来删除私有成员时自动调用的

1当在类外部使用unset()函数来删除私有成员时自动调用的

九、 __sleep()，执行serialize()时，先会调用这个函数

serialize() 函数会检查类中是否存在一个魔术方法 __sleep()。如果存在，则该方法会优先被调用，然后才执行序列化操作。

此功能可以用于清理对象，并返回一个包含对象中所有应被序列化的变量名称的数组。

如果该方法未返回任何内容，则 NULL 被序列化，并产生一个 E_NOTICE 级别的错误。

注意：

__sleep() 不能返回父类的私有成员的名字。这样做会产生一个 E_NOTICE 级别的错误。可以用 Serializable 接口来替代。

作用：

__sleep() 方法常用于提交未提交的数据，或类似的清理操作。同时，如果有一些很大的对象，但不需要全部保存，这个功能就很好用。

具体请参考如下代码：

```
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男'
)
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  /**
   * @return array
   */
  public function __sleep() {
    echo "当在类外部使用serialize()时会调用这里的__sleep()方
法<br>";
    $this->name = base64_encode($this->name);
    return array('name', 'age'); // 这里必须返回一个数值，里
边的元素表示返回的属性名称
  }
}

$person = new Person('小明'); // 初始赋值
echo serialize($person);
echo '<br/>';
```

代码运行结果：

当在类外部使用serialize()时会调用这里的__sleep()方法

0:6:"Person":2:{s:4:"name";s:8:"5bCP5pi0";s:3:"age";i:25;}

十、 __wakeup()，执行unserialize()时，先会调用这个函数

如果说 __sleep() 是白的，那么 __wakeup() 就是黑的了。

那么为什么呢？

因为：

与之相反，`unserialize()` 会检查是否存在一个 `__wakeup()` 方法。如果存在，则会先调用 `__wakeup` 方法，预先准备对象需要的资源。

作用：

__wakeup() 经常用在反序列化操作中，例如重新建立数据库连接，或执行其它初始化操作。

还是看代码：

```php
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  /**
   * @return array
   */
  public function __sleep() {
    echo "当在类外部使用serialize()时会调用这里的__sleep()方法<br>";
    $this->name = base64_encode($this->name);
    return array('name', 'age'); // 这里必须返回一个数值，里边的元素表示返回的属性名称
  }

  /**
   * __wakeup
   */
  public function __wakeup() {
    echo "当在类外部使用unserialize()时会调用这里的__wakeup()方法<br>";
    $this->name = 2;
    $this->sex = '男';
    // 这里不需要返回数组
  }
}

$person = new Person('小明'); // 初始赋值
var_dump(serialize($person));
var_dump(unserialize(serialize($person)));
```

运行结果：

当在类外部使用serialize()时会调用这里的__sleep()方法

string(58) "O:6:"Person":2:{s:4:"name";s:8:"5bCP5piO";s:3:"age";i:25;}" 当在类外部使用serialize()时会调用这里的__sleep()方法

当在类外部使用unserialize()时会调用这里的__wakeup()方法

object(Person)#2 (3) { ["sex"]=> string(3) "男" ["name"]=> int(2) ["age"]=> int(25) }

十一、 __toString()，类被当成字符串时的回应方法

作用：

__toString() 方法用于一个类被当成字符串时应怎样回应。例如 `echo $obj;` 应该显示些什么。

注意：

此方法必须返回一个字符串，否则将发出一条 `E_RECOVERABLE_ERROR` 级别的致命错误。

警告：

不能在 __toString() 方法中抛出异常。这么做会导致致命错误。

代码：

```php
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  public function __toString()
  {
    return 'go go go';
  }
}

$person = new Person('小明'); // 初始赋值
echo $person;
```

结果：

go go go

那么如果类中没有 __toString() 这个魔术方法运行会发生什么呢？让我们来测试下：

代码：

```php
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

}

$person = new Person('小明'); // 初始赋值
echo $person;
```

结果：

Catchable fatal error: Object of class Person could not be converted to string in D:\phpStudy\WWW\test\index.php on line 18

很明显，页面报了一个致命错误，这是语法所不允许的。

十二、 __invoke()，调用函数的方式调用一个对象时的回应方法

作用：

当尝试以调用函数的方式调用一个对象时，__invoke() 方法会被自动调用。

注意：

本特性只在 PHP 5.3.0 及以上版本有效。

直接上代码：

```php
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  public function __invoke() {
    echo '这可是一个对象哦';
  }

}

$person = new Person('小明'); // 初始赋值
$person();
```

查看运行结果：

这可是一个对象哦

当然，如果你执意要将对象当函数方法使用，那么会得到下面结果：

Fatal error: Function name must be a string in D:\phpStudy\WWW\test\index.php on line 18

十三、 __set_state()，调用var_export()导出类时，此静态方法会被调用。

作用：

自 PHP 5.1.0 起，当调用 var_export() 导出类时，此静态方法会被自动调用。

参数：

本方法的唯一参数是一个数组，其中包含按 array('property' => value, ...) 格式排列的类属性。

下面我们先来看看在没有加 __set_state() 情况按下，代码及运行结果如何：

上代码：

?

```php
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男'
)
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

}

$person = new Person('小明'); // 初始赋值
var_export($person);
```

看结果：

Person::__set_state(array('sex' => '男', 'name' => '小明', 'age' => 25,))

很明显，将对象中的属性都打印出来了

加了 __set_state() 之后：

继续上代码：

```php
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  public static function __set_state($an_array)
  {
    $a = new Person();
    $a->name = $an_array['name'];
    return $a;
  }

}

$person = new Person('小明'); // 初始赋值
$person->name = '小红';
var_export($person);
```

继续看结果：

Person::__set_state(array('sex' => '男', 'name' => '小红', 'age' => 25,))

十四、 __clone()，当对象复制完成时调用

在多数情况下，我们并不需要完全复制一个对象来获得其中属性。但有一个情况下确实需要：如果你有一个 GTK 窗口对象，该对象持有窗口相关的资源。你可能会想复制一个新的窗口，保持所有属性与原来的窗口相同，但必须是一个新的对象（因为如果不是新的对象，那么一个窗口中的改变就会影响到另一个窗口）。还有一种情况：如果对象 A 中保存着对象 B 的引用，当你复制对象 A 时，你想其中使用的对象不再是对象 B 而是 B 的一个副本，那么你必须得到对象 A 的一个副本。

作用：

对象复制可以通过 clone 关键字来完成（如果可能，这将调用对象的 __clone() 方法）。对象中的 __clone() 方法不能被直接调用。

语法：

$copy_of_object = clone $object;

注意：

当对象被复制后，PHP 5 会对对象的所有属性执行一个浅复制（shallow copy）。所有的引用属性 仍然会是一个指向原来的变量的引用。

当复制完成时，如果定义了 __clone() 方法，则新创建的对象（复制生成的对象）中的 __clone() 方法会被调用，可用于修改属性的值（如果有必要的话）。

看代码：

```php
<?php
class Person
{
  public $sex;
  public $name;
  public $age;

  public function __construct($name="", $age=25, $sex='男')
  {
    $this->name = $name;
    $this->age = $age;
    $this->sex = $sex;
  }

  public function __clone()
  {
    echo __METHOD__."你正在克隆对象<br>";
  }

}

$person = new Person('小明'); // 初始赋值
$person2 = clone $person;

var_dump('persion1:');
var_dump($person);
echo '<br>';
var_dump('persion2:');
var_dump($person2);
```

看结果：

Person::__clone你正在克隆对象

string(9) "persion1:" object(Person)#1 (3) { ["sex"]=> string(3) "男" ["name"]=> string(6) "小明" ["age"]=> int(25) }

string(9) "persion2:" object(Person)#2 (3) { ["sex"]=> string(3) "男" ["name"]=> string(6) "小明" ["age"]=> int(25) }

克隆成功。

十五、__autoload()，尝试加载未定义的类

作用：

你可以通过定义这个函数来启用类的自动加载。

在魔术函数 __autoload() 方法出现以前，如果你要在一个程序文件中实例化100个对象，那么你必须用include或者require包含进来100个类文件，或者你把这100个类定义在同一个类文件中 —— 相信这个文件一定会非常大，然后你就痛苦了。

但是有了 __autoload() 方法，以后就不必为此大伤脑筋了，这个类会在你实例化对象之前自动加载制定的文件。

还是通过例子来看看吧：

先看看以往的方式：

```php
/**
 * 文件non_autoload.php
 */

require_once('project/class/A.php');
require_once('project/class/B.php');
require_once('project/class/C.php');

if (条件A) {
  $a = new A();
  $b = new B();
  $c = new C();
  // … 业务逻辑
} else if (条件B) {
  $a = newA();
  $b = new B();
  // … 业务逻辑
}
```

看到了吗？不用100个，只是3个看起来就有点烦了。而且这样就会有一个问题：如果脚本执行“条件B”这个分支时，C.php这个文件其实没有必要包含。因为，任何一个被包含的文件，无论是否使用，均会被php引擎编译。如果不使用，却被编译，这样可以被视作一种资源浪费。更进一步，如果C.php包含了D.php，D.php包含了E.php。并且大部分情况都执行“条件B”分支，那么就会浪费一部分资源去编译C.php,D.php,E.php三个“无用”的文件。

那么如果使用 __autoload() 方式呢？

```
/**
 * 文件autoload_demo.php
 */
function __autoload($className) {
  $filePath = "project/class/{$className}.php";
  if (is_readable($filePath)) {
    require($filePath);
  }
}

if (条件A) {
  $a = new A();
  $b = new B();
  $c = new C();
  // … 业务逻辑
} else if (条件B) {
  $a = newA();
  $b = new B();
  // … 业务逻辑
}
```

ok,不论效率怎么用，最起码界面看起来舒服多了，没有太多冗余的代。

再来看看这里的效率如何，我们分析下：

当php引擎第一次使用类A，但是找不到时，会自动调用 __autoload 方法，并将类名“A”作为参数传入。所以，我们在 __autoload() 中需要的做的就是根据类名，找到相应的文件，并包含进来，如果我们的方法也找不到，那么php引擎就会报错了。

注意：

这里可以只用require，因为一旦包含进来后，php引擎再遇到类A时，将不会调用__autoload，而是直接使用内存中的类A，不会导致多次包含。

扩展：

其实php发展到今天，已经有将 `spl_autoload_register` — 注册给定的函数作为 __autoload 的实现了，但是这个不在啊本文讲解之内，有兴趣可以自行看手册。

十六、__debugInfo()，打印所需调试信息

注意：

该方法在PHP 5.6.0及其以上版本才可以用，如果你发现使用无效或者报错，请查看啊你的版本。

看代码：

```
<?php
class C {
  private $prop;

  public function __construct($val) {
    $this->prop = $val;
  }

  /**
   * @return array
   */
  public function __debugInfo() {
    return [
      'propSquared' => $this->prop ** 2,
    ];
  }
}

var_dump(new C(42));
```

结果：

object(C)#1 (1) { ["propSquared"]=> int(1764) }

再次注意：

这里的 `**` 是乘方的意思，也是在PHP5.6.0及其以上才可以使用，详情请查看PHP手册