План:

1.**Введение**

Хотя PHP обладает общими объектно-ориентированными возможностями, он не является полноценным ОО-языком (например, таким, как C++ или Java). В частности, в PHP не поддерживаются следующие объектно-ориентированные возможности:

– множественное наследование;

– автоматический вызов конструкторов (если необходимо, чтобы при конструировании объекта производного класса вызывался конструктор базового класса, придется вызвать его явно);

– абстрактные классы;

– перегрузка методов;

– перегрузка операторов (это связано с тем, что PHP является языком со свободной типизацией, — за дополнительной информацией обращайтесь к главе 2);

– закрытый и открытый доступ, виртуальные функции;

– деструкторы;

– полиморфизм.

Но и без всего перечисленного все равно можно извлечь пользу из объектно-ориентированных возможностей, поддерживаемых PHP.

2.Классы, объекты и объявления методов

Классы образуют синтаксическую базу ООП. Их можно рассматривать как своего рода «контейнеры» для логически связанных данных и функций (обычно называемых методами — см. ниже). Класс представляет собой шаблон, по которому создаются конкретные экземпляры, используемые в программе. Экземпляры классов называются объектами.

Общий формат классов PHP приведен в ниже

Объявление классов в PHP

```
class Class_name {
var $attribute_1;
...
```

```
var $attribute_N;
function function1() {
...
}
...
function functionN() {
...
} // end Class_name
```

Подведем итоги: объявление класса должно начинаться с ключевого слова class (подобно тому, как объявление функции начинается с ключевого слова function). Каждому объявлению атрибута, содержащегося в классе, должно предшествовать ключевое слово var. Атрибуты могут относиться к любому типу данных, поддерживаемых в PHP. После объявлений атрибутов следуют объявления методов, очень похожие на типичные объявления функций.

По общепринятым правилам имена классов ООП начинаются с прописной буквы. При ссылках на атрибуты внутри методов используется специальная переменная $this. Синтаксис методов продемонстрирован в следующем примере:

```
<?
class Webpage {
van $bgcolor;
function setBgColor($color) {
$this->bgcolor = $color;
}
function getBgColor() {
return $this->bgcolor;
}
}
?>
```

Переменная $this ссылается на экземпляр объекта, для которого вызывается метод. Поскольку в любом классе может существовать несколько экземпляров объектов, уточнение $this необходимо для ссылок на атрибуты, принадлежащие текущему объекту. При использовании этого синтаксиса обратите внимание на два обстоятельства:

- атрибут, на который программист ссылается в методе, не нужно передавать в виде параметра функции;
- знак доллара ($) ставится перед переменной $this, но не перед именем атрибута (как у обычной переменной).

3.Создание объектов и работа с ними

Объекты создаются оператором new. Например, объект класса Webpage создается следующей командой:

```
$home_page = new Webpage;
```

Новый объект с именем $some_page обладает собственным набором атрибутов и методов, перечисленных в классе Webpage. Для изменения значения атрибута $bgcolor, принадлежащего этому конкретному объекту, можно воспользоваться определенным в классе методом setBgColor( ):

```
$some_page->setBgColor("black");
```

Следует помнить, что PHP также позволяет явно получить значение атрибута указанием имен объекта и атрибута:

```
$some_page->bgcolor;
```

Однако второй способ противоречит принципу инкапсуляции, и при работе с ООП поступать так не следует.

4.Конструкторы

Довольно часто при создании объекта требуется задать значения некоторых атрибутов. Конструктор представляет собой метод, который задает значения некоторых атрибутов (а также может вызывать другие методы). Конструкторы вызываются автоматически при создании новых объектов. Имя метода-конструктора должно совпадать с именем класса, в котором он содержится. Пример конструктора приведен ниже.

Использование конструктора

```
<?
class Webpage {
var $bgcolor;
function Webpage($color) {
$this->bgcolor = $color;
}
}
// Вызвать конструктор класса Webpage
$page = new Webpage("brown");
?>
```

Деструкторы

Как упоминалось ранее, в PHP отсутствует непосредственная поддержка деструкторов. Тем не менее, можно легко имитировать работу деструктора, вызывая функцию PHP unset(). Эта функция уничтожает содержимое переменной и возвращает занимаемые ресурсы системе. С объектами unset() работает так же, как и с переменными. Допустим, ведется работа с объектом $Webpage. После завершения работы с этим конкретным объектом вызывается функция

```
unset($Webpage);
```

Эта команда удаляет из памяти все содержимое $Webpage. Действуя в духе инкапсуляции, можно поместить вызов unset( ) в метод с именем destroy( ) и затем вызвать его:

```
$Website->destroy( );
```

Необходимо отметить, что необходимость в вызове деструкторов возникает лишь при работе с объектами, использующими большой объем ресурсов, поскольку все переменные и объекты автоматически уничтожаются по завершении сценария.

```
<?
class Webpage {
```

```
        var $bgcolor;
        function Webpage($color) {
                $this->bgcolor = $color;
        }

        function get_Color(){
                return $this->bgcolor;
        }
        function set_Color($newColor){
                $this->bgcolor = $newColor;
        }
        function destroy(){
                unset ($this);
        }
}
// Вызвать конструктор класса Webpage
$page = new Webpage("brown");
$page->set_Color("white");
echo $page->get_Color();
$page->destroy();
?>
```

5.Простое и иерархическое наследование

Наследование является исключительно полезным средством программирования, поскольку его применение предотвращает копирование кода, совместно используемого структурами данных. В общем случае синтаксис наследования характеристик другого класса в PHP выглядит так:

```
class Class_name2 extends Class_name1 {
объявления атрибутов;
объявления методов;
}
```

Ключевое слово extends говорит о том, что класс Class_name2 наследует все характеристики класса Class_name1.

Помимо возможности многократного использования кода, наследование обладает еще одним важным преимуществом — снижается вероятность ошибок при модификации программы.

Пример представления различных типов транспортных средств с использованием наследования

```
<?
// Транспортное средство
class Vehicle {
var $model;
var $current_speed;
function setSpeed($mph) {
$this->current_speed = $mph;
}
function getSpeed() {
return $this->current_speed;
}
}
// Автомобиль
class Auto extends Vehicle {
var $fue1_type;
function setFuelType($fuel) {
$this->fuel_type = $fuel;
}
function getFuelType() {
return $this->fuel_type;
}
}
```

```
// Самолет
class Airplane extends Vehicle {
var $wingspan;
function setWingSpan($wingspan) {
$this->wingspan = $wingspan;
}
function getWingSpan() {
return $this->wingspan;
}
}
?>
```

Объекты этих классов создаются следующим образом:

```
$tractor = new Vehicle;
$gulfstream = new Airplane;
```

Приведенные команды создают два объекта. Первый объект, $tractor, относится к классу Vehicle. Второй объект, $gulfstream, относится к классу Airplane и потому обладает как общими характеристиками класса Vehicle, так и уточненными характеристиками класса Airplane.

Ситуация, при которой класс наследует свойства нескольких родительских классов, называется множественным наследованием. К сожалению, в PHP множественное наследование не поддерживается. Например, следующая конструкция невозможна в PHP:

```
class Airplane extends Vehicle extends Building...
```

Многоуровневое наследование

При многоуровневом наследовании класс наследует свои свойства от других классов, которые, в свою очередь, будут наследовать от третьих классов и т. д. Многоуровневое наследование развивает модульную структуру программы, обеспечивая простоту сопровождения и более четкую логическую структуру.

Пример многоуровневого наследования

```
<?
class Vehicle {
Объявления атрибутов...
Объявления методов...
}
class Land extends Vehicle {
Объявления атрибутов...
Объявления методов...
}
class Car extends Land {
Объявления атрибутов...
Объявления методов...
}
$nissan = new Car;
?>
```

Объект $nissan содержит все атрибуты и методы классов Car, Land и Vehicle. Программа получается исключительно модульной. Допустим, когда-то в будущем возникнет необходимость добавления классу Land нового атрибута. Для этого достаточно внести соответствующие изменения в класс Land, и этот атрибут немедленно становится доступным для классов Land и Car, не влияя на функциональность других классов. Таким образом, модульность кода и его гибкость относятся к числу основных преимуществ ООП.

Хотя класс наследует свои характеристики от цепочки родителей, конструкторы родительских классов не вызываются автоматически при создании объектов класса-наследника. Для этого конструкторы могут вызываться классом-наследником в виде методов.

6.Абстрактные классы

Иногда целесообразно создать класс, объекты которого никогда не будут созданы (данный класс нужен всего лишь как базовый для создания

производных классов). В данном случае класс называется абстрактными. Абстрактные классы применяются тогда, когда разработчик программы хочет обеспечить обязательную поддержку некоторых функциональных возможностей всеми классами, производными от абстрактного базового класса.

В PHP отсутствует прямая поддержка абстрактных классов, однако существует простое обходное решение — достаточно определить в «абстрактном» классе конструктор и включить в него вызов die().

Пример создания абстрактных классов

```
<?
class Vehicle {
Объявления атрибутов...
function Vehicle() }
die ("Cannot create Abstract Vehicle class!");
}
Объявления других методов...
}
class Land extends Vehicle {
Объявления атрибутов...
function Land() {
die ("Cannot create Abstract Land class!");
}
Объявления других методов.
}
class Car extends Land {
Объявления атрибутов...
Объявления методов...
}
?>
```

Попытка создания экземпляра описанных выше абстрактных классов приведет к выдаче сообщения об ошибке и завершению программы.

7.Перегрузка методов

Перегрузкой методов это определение нескольких методов с одинаковыми именами, но разным количеством или типом параметров. В PHP эта возможность не поддерживается, но существует простое обходное решение, приведенное ниже.

Пример перегрузки методов

```
<?
class Page {
var $bgcolor;
var $textcolor;
function Page() {
// Определить количество переданных аргументов и вызвать метод с нужным именем
$name = "Page".func_num_args();
// Call $name with correct number of arguments passed in
if ( func_num_args() == 0 ) :
$this->$name();
else :
$this->$name(func_get_arg(0));
endif;
}
function Page0() {
$this->bgcolor = "white";
$this->textcolor = "black";
print "Created default page";
}
function Page1($bgcolor) {
$this->bgcolor = $bgcolor;
```

```
$this->textcolor = "black";
print "Created custom page";
}
}
$html_page = new Page("red");
echo "\n" . $html_page->bgcolor;
$html_page = new Page();
echo "\n" . $html_page->bgcolor;
?>
```

В примере при создании нового объекта с именем $html_page передается один аргумент. Поскольку в классе был определен конструктор по умолчанию (Page( )), вызывается именно он. Однако конструктор по умолчанию всего лишь выбирает, какому из конструкторов (Page0( ) или Page1( )) следует передать управление. При выборе конструктора используются функции func_num_args( ) и func_get_arg( ), которые, соответственно, определяют количество аргументов и читают эти аргументы.

8.Функции для работы с классами и объектами

Данные функции используются в процессе разработки интерфейса, администрирования кода и диагностики ошибок.

get_class_methods( )

Функция get_class_methods( ) возвращает массив имен методов класса с заданным именем. Синтаксис функции get_class_methods( ):

array get_class_methods (string имя_класса)

Пример получение списка методов класса

```
<?
class Vehicle {
    var $model;
    var $current_speed;
    function setSpeed($mph) {
        $this->current_speed = $mph;
```

```
        }
        function getSpeed() {
                return $this->current_speed;
        }
}
class Airplane extends Vehicle {
        var $wingspan;
        function setWingSpan($wingspan) {
        $this->wingspan = $wingspan;
        }
        function getWingSpan() {
        return $this->wingspan;
        }
}
$cls_methods = get_class_methods("Airplane");
foreach ($cls_methods as $method_name) {
 echo "$method_name\n";
}


?>
```

Результат:

setspeed

getspeed

setwingspan

getwingspan

Как видно из примера, функция get_class_methods() позволяет легко получить информацию обо всех методах, поддерживаемых классом.

get_class_vars( )

Функция get_class_vars( ) возвращает массив имен атрибутов класса с заданным именем. Синтаксис функции get_class_vars( ):

array get_class_vars (string имя_класса)

```
<?
class Vehicle {
var $model;
var $current_speed = 100;
}
class Airplane extends Vehicle {
var $wingspan;
}
$attribs = get_class_vars("Airplane");
foreach ($attribs as $name => $value) {
        echo "$name : $value\n";
}
?>
```

model:

current_speed : 100

wingspan: 100

Массив $attribs заполняется именами всех атрибутов класса Airplane.

get_object_vars( )

Функция get_object_vars( ) возвращает ассоциативный массив с информацией обо всех атрибутах объекта с заданным именем. Синтаксис функции get_object_vars( ):

array get_object_vars (object имя_обьекта)

Функция get_object_vars( ) позволяет быстро получить всю информацию об атрибутах конкретного объекта и их значениях в виде ассоциативного массива.

method_exists( )

Функция method_exists( ) проверяет, поддерживается ли объектом метод с заданным именем. Если метод поддерживается, функция возвращает TRUE, в противном случае возвращается FALSE. Синтаксис функции method_exists( ):

bool method_exists (object имя_обьекта. string имя_метода)

get_class( )

Функция get_class( ) возвращает имя класса, к которому относится объект с заданным именем. Синтаксис функции get_class( ):

string get_class(object имя_объекта);

get_parent_class( )

Функция get_parent_class( ) возвращает имя родительского класса (если он есть) для объекта с заданным именем. Синтаксис функции get_parent_dass():

string get_parent_class (object имя_обьекта);

is_subclass_of( )

Функция is_subclass_of( ) проверяет, был ли объект создан на базе класса, имеющего родительский класс с заданным именем. Функция возвращает TRUE, если проверка дает положительный результат, и FALSE в противном случае. Синтаксис функции is_subclass_of( ):

bool is_subclass_of (object объект, string имя_класса)

get_declared_classes( )

Функция get_declared_classes( ) возвращает массив с именами всех определенных классов. Синтаксис функции get_declared_classes( ):

array get_declared_classes( )

9.ЛИТЕРАТУРА

1. Ал. Олбах Все о PHP, Мн: ВШ, 2006
2. Холл М., Браун Л. Программирование для WEB. Вильямс, 1280, 2002