Объектно-ориентированное программирование в PHP для начинающих

Доброго времени суток, уважаемые читатели нашего блога! Добро пожаловать на первый урок из серии уроков по **объектно-ориентированному программированию в PHP**. На нашем блоге уже были уроки о том, что такое PHP, также было рассказано о работе с сылками в PHP и о многом другом.

А в этой серии уроков вы сможете изучить различные **понятия объектно-ориентированного программирования** и узнать, как быстро и легко создавать надежные объектно-ориентированные PHP приложения. В этой статье мы начнем с рассмотрения некоторых базовых принципов объектно-ориентированного программирования.

Содержание

Понятие объектно-ориентированного программирования в PHP

Вы когда-нибудь работали с PHP? Тогда вы знаете, как создавать

и использовать функции в PHP. Наверняка вы знакомы с процедурным программированием в PHP. Посредством процедурного программирования обычно создаются структуры данных — номеров, строк, массивов и так далее — для хранения ваших данных, а затем эти структуры данных передаются в функции, которые работают с этими данными.

Объектно-ориентированное программирование, или ООП, **хранит, как структуры данных так и функции, которые работают с этими данными, в едином месте, известном как объект**. Вместо того, чтобы передавать данные в вызовы функций, вы загружаете данные, с которыми вы хотите работать, в объект, а затем вызываете функции объекта для управления объектными данными и получения желаемого результата.

Часто объекты, которые вы создаете с помощью ООП, отражают реальные объекты. Например, если бы вы писали свой веб-сайт, вы бы создали объект *User*, который содержит всю информацию о том или ином пользователе вашего сайта (псевдоним, имя пользователя, адрес электронной почты, пароль, и так далее), а также функции для работы с данными (регистрация, имя для входа, выход из системы, запрет и так далее).

Для чего нужно объектно-ориентированное программирование в PHP?

Как вы уже поняли можно использовать процедурное программирование или ООП, чтобы делать одинаковые вещи. Сказать, что одно превосходит другое будет неверным. Каждый выбирает сам, что ему лучше использовать. Конечно, можно программировать, используя и тот и другой способ. Например, в одном и том же примере программирования можно комбинировать два этих подхода.

Тем не менее у ООП есть ряд преимуществ:

1. **Написание модульного кода**: С помощью ООП вы можете написания код в отдельных модулях. Разделяя код на отдельные модули, вы можете сделать код более управляемым, и простым в отладке и расширении.
2. **Может отражать, то что происходит в реальном мире**: Как я уже говорил ранее, объекты могут легко отображаться на реальных «объектах», таких как животные, транспорт, архитектура и так далее. Благодаря этому задача упрощается, когда вы приступаете к проектированию ваших объектно-ориентированных приложений, поскольку цель каждого объекта, а также связь между объектами, и без того уже достаточно ясны.
3. **Использование кода для многократного применения**: Код многократного использования поможет вам сэкономить много времени при написании приложений, и с течением времени вы можете создать библиотеку кодов многократного использования, которые вы можете использовать в различных приложениях. ООП облегчает написание кода многократного использования, поскольку структуры данных и функции красиво содержатся в одном объекте многократного использования. Также можно воспользоваться уже существующим объектом и расширить его для того, чтобы добавить новые функции для конкретного приложения, что в свою очередь облегчает повторное использование объектов.

Основополагающие понятия ООП в PHP

Прежде чем перейти к написанию объектно-ориентированного кода на PHP, важно понимать 4 основных понятия ООП: **Классы, объекты, свойства и методы**.

Классы

Класс является основой для объектов. Это блок кода, который определяет:

Типы данных, которые объекты, созданные из класса, будут хранить и функции, которые эти объекты будут содержать.
При создании объектно-ориентированного приложения, вы обычно создаете один или более классов, представляющих различные типы объектов в вашем приложении. Например, если вы пишете сайт, вы можете создать классы, называемые *Site*, *Article*, *Category* и *User*.

Объекты

Объект — это **специальный тип переменной, который создается из класса**. Он содержит фактические данные, и вы можете вызывать объектные функции, чтобы работать с этими данными. Из одного класса вы можете создать сколько угодно объектов. Каждый объект функционирует независимо от других, даже если все они из одного и того же класса.

Для примера:

- Класс можно сравнить с чертежом здания. Он определяет, как будет выглядеть и какие материалы потребуются для здания, которое создано по этому чертежу, но все же это просто абстрактное понятие.
- Объект можно сравнить с реальным зданием, созданным по чертежу. Он имеет реальные свойства (например, «высота») и реальные материалы из которого сделано здание (например, «цемент» и «сталь»»).

Часто говорят, что объект — это экземпляр класса, и процесс создания объекта из класса называется созданием экземпляра.

Свойства

Значения данных, которые содержит объект, хранятся в специальных переменных, известных как свойства. Свойства объекта тесно связаны с объектом. Несмотря на то, что все объекты, созданные из данного класса, имеют одни и те же свойства, свойства одного объекта могут иметь различные

значения для свойств другого объекта.

Методы

Функции, которые определены в пределах класса и используются в объекте, известны как методы. Во многих отношениях, они подобны обычным функциям; им можно присваивать значения, они могут содержать локальные переменные и возвращать значения. Однако методы объекта обычно работают в основном со свойствами объекта. Например, метод *signin()* для входа на сайт может установить свойство *SignIn* для пользователя сайта в значение *true*.

Создание классов в PHP

Теперь, когда вы знаете, какие существуют классы, объекты, свойства и методы, самое время для того, чтобы приступить к работе и **создать некоторые классы и объекты с помощью PHP кода**.

Во-первых, давайте рассмотрим создание класса. Базовый синтаксис для создания класса в PHP выглядит так:

```
<?php

  class ИмяКласса
  {
    // (Здесь находится описание класса)
  }

?>
```

Например, если бы вы создали класс *User* на своём веб-сайте, тогда бы вы написали следующий код:

```
<?php

class User
```

```
{
  // (Здесь находится описание класса)
}

?>
```

Довольно просто не правда ли?! Конечно, этот класс ничего не может делать, до тех пор пока вы не добавите свойства и методы этого класса. Тем не менее, **приведенный выше код создает настоящий класс, пригодный для использования в PHP**.

Хорошим тоном в ООП программировании является то, что для каждого класса нужно создавать свой собственный файл, с тем же именем, что и имя класса. Например, вы могли бы записать указанный выше код класса *User* в файл *User.php* и сохранить файл в папку *classes*.

Создание объектов в PHP

Вы можете создать объект класса с помощью ключевого слова **new**, следующим образом:

```
<?php

 new Имя_класса()

?>
```

Эта конструкция создает объект на базе класса «Имя_класса» . Обычно, чтобы хранить вновь созданный объект, нужно присвоить его переменной, которую можно использовать позже для доступа к объекту. Например, давайте создадим новый объект *User* и сохраним его в переменную *$User*:

```
<?php

$user = new User();

?>
```

Также мы можем создать еще один объект в этом же классе:

```
<?php

$user2 = new User();

?>
```

Даже если мы создали эти классы из одного и того же класса *User*, то эти объекты *$User* и *$User2* абсолютно не зависят друг от друга.

Свойства класса

Теперь, когда мы знаем, **как создавать классы и объекты PHP**, давайте рассмотрим **добавление свойств**. Существует 3 типа свойств, которые можно добавить в класс:

- Свойства с ключевым словом *public* они будут доступны везде. Эти свойства можно прочитать или изменить с помощью любого кода в вашем сценарии, будь этот код внутри или вне класса.
- Свойства с ключевым словом *private* доступны только с помощью методов в пределах класса. Если это возможно, то лучше всего делать ваши свойства недоступными извне, с помощью ключевого слова *private*, поскольку это помогает сохранить объекты отделенными от остальной части вашего кода.
- Свойства с ключевым словом *protected* могут быть доступны с помощью методов внутри класса, а также в классах, которые являются наследниками родительского класса. Чтобы добавить в класс свойство, нужно перед ним написать одно из ключевых слов *public*, *private* или *protected*:

```
<?php

class ИмяКласс
```

```
{
  public $ИмяСвойства;
  private $ИмяСвойства;
  protected $ИмяСвойства;
}

?>
```

Добавим в класс *User* свойство для хранения имени пользователя:

```
<?php

class User
{
      public $username = "";
}

?>
```

Обратите внимание, что при создании свойства, мы присваиваем ему пустую строку – «» . Это означает, что, когда новое свойство объекта *$username* будет создано, то ему будет присвоена пустая строка. Как и в случае с обычными переменными, при создании свойств им не обязательно присваивать какие-то значения, но часто это очень хорошая идея. Если вы не присвоите свойству значие, то ему по умолчанию будет установлено значение *null*.

Доступ к свойствам

Чтобы получить доступ к объекту нужно использовать оператор «->», следующим образом:

```
<?php

 $Объект -> ИмяСвойства

?>
```

Попробуйте так сделать. Давайте создадим скрипт, который

объявляет класс *User* и его свойства, создает новые объекты *User*, а затем устанавливает и считывает свойство *username*:

```
<?php

class User
{
  public $username = "";
}

$User = new User();
$User->username = "Петя";
echo $User->username; // Выводит "Петя"

?>
```

Если вы запустите указанный выше код, то в результате будет выведена строка «Петя», которая является значением свойства *$User->username*. Как можно заметить, свойство объекта используется обычная переменная. Можно прочитать его значение и присвоить ему новое.

Методы класса

Что можно сказать о добавлении методов класса? Как уже упоминалось ранее, **методы это обычные функции, которые являются частью класса**. Поэтому нет ничего удивительного в том, что можно создавать методы, как любые другие функции с помощью ключевого слова *function*. Единственное различие заключается в том, что вам следует также добавлять ключевые слова *public*, *private* или *protected* перед объявлением метода, также как в случае свойств:

```
<?php

 class ИмяКласса
{
  public function ИмяМетода() {
    // (Запишите код вашего метода здесь)
```

```
  }

  private function ИмяМетода() {
    // (Запишите код вашего метода здесь)
  }

  protected function ИмяМетода() {
    // (Запишите код вашего метода здесь)
  }
}

?>
```

Как и в случае со свойствами, методы с ключевым словом *public* могут быть **использованы в любом месте**. Методы с ключевым словом *private* могут в**ызываться только с помощью других методов внутри класса**. И методы с ключевым словом *protected* **могут быть вызваны методами внутри класса и в любых производных классах**, то есть те, которые являются потомками родительского класса.

Добавим некоторые методы для нашего класса *User*:

- Свойство *$SignIn* с ключевым словом private со значением, которое указывает, вошёл ли пользователь на сайт или нет.
- Метод, называемый *signin()*, который «пропускает» пользователя на сайт, устанавливая значение свойства *$SignIn* в значение *true*.
- Метод, называемый *signout()*, который «выпускает» пользователя с сайта, устанавливая значение свойства *$SignIn*в значение *false*.
- Метод, называемый *isSignIn()* возвращает значение свойства *$SignIn*.

Вот наш код:

```
<?php

class User
```

```
{
  public $username = "";
  private $SignIn = false;

  public function signin() {
    $this->SignIn = true;
  }

  public function signout() {
    $this->SignIn = false;
  }

  public function isSignIn() {
    return $this->SignIn;
  }
}

?>
```

Обратите внимание на то, что мы использовали новое ключевое слово *$this* внутри наших методов. В контексте метода объект ссылается на самого себя с помощью специальной переменной *$this*. С помощью *$this* метод может получить доступ к любому другому свойству и методу объекта. Подробнее о том, как в случае *$this* работает механизм ссылок в PHP.

Например, метод *signin()* имеет доступ к свойству *$SignIn* этого же объекта, используя конструкцию *$this->SignIn*.

Кстати, свойство *$SignIn* является недоступным (*private*), поэтому доступ к нему осуществляется только через методы *signin()*, *signout()* и *isSignIn()*. **Это хороший подход, поскольку он сохраняет внутреннюю информацию об объекте**. Он запрещает доступ во внешнем коде к тому, как происходит запись входа или выхода пользователя на сайт. Использование недоступных (*private*) свойств, там где это возможно, **позволяет сохранить объекты самостоятельными, переносимыми и безопасными**.

Свойство *$username* сделано доступным (*public*) главным образом

для того, чтобы показать, как это работает. В реальных условиях вы, возможно, захотите сделать *$username* недоступным и добавите доступные (*public*) методы для чтения и записи имени пользователя, если это необходимо в вашем случае.

Примеры работы с методами в ООП

Для вызова объекта, нужно использовать оператор «->»:

```
<?php

 $Объект -> Имя_Метода()

?>
```

Этот код работает как обычный вызов функции. Вы можете передавать аргументы в круглых скобках (при условии, что метод может принимать аргументы). Кроме того метод может возвращать значения, которые вы можете использовать по своему усмотрению. Теперь добавим некоторые методы в наш класс *User* и попробуем использовать их:

```
<?php

class User
{
  public $username = "";
  private $SignIn = false;

  public function signin() {
    $this->SignIn = true;
  }

  public function signout() {
    $this->SignIn = false;
  }

  public function isSignIn() {
    return $this->SignIn;
  }
```

```
}

$User = new User();
$User->username = "Петя";
echo $User->username . " " .
( $User->isSignIn() ? "вошел в систему" :  "вышел из системы"
) .  "
";
$User->signin();
echo $User->username . " " .
( $User->isSignIn() ? "вошел в систему" :  "вышел из системы"
) .  "
";
$User->signout();
echo $User->username . " " .
( $User->isSignIn() ? "вошел в систему" :  "вышел из системы"
) .  "
";

?>
```

Указанные выше скрипт отображает следующее:

Петя вышел из системы

Петя вошел в систему

Петя вышел из системы

Вот как работает скрипт:

- После определения класса *User* мы создаем новый объект *User* и храним его значение в переменной *$User*. Установим имя пользователя, как «Петя».
- Затем вызовем метод *$User->isSignIn()*, чтобы определить, вошёл ли пользователь на сайт. Метод *IsSignIn()* возвращает значение свойства *$SignIn*. Пока свойство *$SignIn* по умолчанию *false* , метод *isSignIn()* возвращает значение *false* в вызывающий код, в результате чего появляется сообщение «Петя вышел из системы».

- Далее мы вызываем метод *signin()*. Он устанавливает свойство *$SignIn* в значение *true* .
  Теперь, когда мы вызываем метод *isSignIn()* он возвращает значение *true* и выводит строку «Петя вошел в систему».
- Затем вызываем метод *signout()*, который устанавливает *$SignIn* в значение *false*.
  И наконец, мы вызываем метод *isSignIn()* в третий раз. В этот раз метод возвращает значение *false*, так как свойство *$SignIn* имеет значение *false*. Поэтому ещё раз появляется строка «Петя вышел из системы».

В случае, если вы не знакомы с тернарным оператором «?:», который используется в примере, то можете больше узнать о нём. Это компактный вариант условного оператора «if … else».

Успешного вам освоения языка программирования PHP на реальных примерах!

Источник: www.elated.com