Урок №7,8,9

Пространства имен

Что такое пространства имен?

В PHP пространства имен используются для решения двух проблем:

1. Конфликт имен между вашим кодом и сторонними, либо внутренними классами/функциями/константами PHP.

2. Возможность создавать псевдонимы (или сокращения) для Ну_Очень_Длинных_Имен, чтобы облегчить первую проблему и улучшить читаемость исходного кода.

Пространства имен PHP – предоставляют возможность группировать логически связанные *классы*, *интерфейсы*, *функции* и *константы*.

По умолчанию, все имена констант, классов и функций размещены в глобальном пространстве – как это и было до того, как PHP стал поддерживать пространства имен.

Как определить пространство имен?

Пространства имен объявляются с помощью зарезервированного слова namespace.

Файл, содержащий пространство имен, должен содержать его объявление в начале перед любым другим кодом, кроме зарезервированного слова declare.

```php
namespace MyProject;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
```

Пространства имен PHP позволяют создавать иерархию имен. Таким образом, имя пространства может быть определено с подуровнями.

```php
namespace MyProject\Sub\Level;

class Connection { /* ... */ }
```

Нескольких пространств имен в одном файле

Несколько пространств имен также можно описать в одном файле с помощью двух допустимых синтаксических конструкций.

```
namespace MyProject1;
# PHP code for the MyProject1 namespace

namespace MyProject2;
# PHP code for the MyProject2 namespace

# Alternative syntax
namespace MyProject3 {
    # PHP code for the MyProject3 namespace
}
```

Данный синтаксис не рекомендуется для комбинирования пространств имен в одном файле.

Вместо этого рекомендуется использовать альтернативный синтаксис со скобками.

Классификация имен пространства имен

Неполное имя – это идентификатор без разделителя пространств имен.

Например, Foo.

Полное имя – это идентификатор с разделителем пространств имен.

Например, Foo\Bar.

Абсолютное имя – это идентификатор с разделителем пространств имен, который начинается с разделителя пространств имен.

Например, \Foo\Bar.

Пространство имен \Foo также является абсолютным именем.

Как использовать пространства имен?

```php
namespace Foo\Bar;

const FOO = 2;
function foo() {}
class foo {
    static function staticmethod() {}
}

/* Неполные имена */
foo();                 # определяется как функция Foo\Bar\foo
foo::staticmethod(); # определяется как класс Foo\Bar\foo::staticmethod
echo FOO;              # определяется как константа Foo\Bar\FOO

/* Полные имена */
# определяется как функция Foo\Bar\subnamespace\foo
subnamespace\foo();
# определяется как класс Foo\Bar\subnamespace\foo с методом staticmethod
subnamespace\foo::staticmethod();
# определяется как константа Foo\Bar\subnamespace\FOO
echo subnamespace\FOO;

/* Абсолютные имена */
# определяется как функция Foo\Bar\foo
\Foo\Bar\foo();
# определяется как класс Foo\Bar\foo с методом staticmethod
\Foo\Bar\foo::staticmethod();
# определяется как константа Foo\Bar\FOO
echo \Foo\Bar\FOO;
```

Импорт/Создание псевдонима имени

Создание псевдонима имени выполняется с помощью оператора use.

Ключевое слово use должно быть указано в самом начале файла (в глобальной области) или внутри объявления пространства имен.

Пространства имен PHP поддерживают три вида создания псевдонима имени или импорта:

1. создание псевдонима для имени класса;
2. создание псевдонима для имени интерфейса;
3. создание псевдонима для имени пространства имен.

Импорт функций или констант не поддерживается.

```php
namespace foo;
use My\Full\Classname as Another;

# это тоже самое, что и использование My\Full\NSname как NSname
use My\Full\NSname;

# импортирование глобального класса
use ArrayObject;

$obj = new namespace\Another(); # создает экземпляр класса foo\Another

$obj = new Another(); # создает объект класса My\Full\Classname

NSname\subns\func();  # вызывает функцию My\Full\NSname\subns\func

$a = new ArrayObject(array(1)); # создает объект класса ArrayObject
# без выражения "use ArrayObject" мы создадим объект класса
foo\ArrayObject
```

```php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another(); # создает объект класса My\Full\Classname
NSname\subns\func();  # вызывает функцию My\Full\NSname\subns\func
```

Классы и объекты
Вторая часть

Как объявить абстрактный класс?

Абстрактный метод – это метод для которого отсутствует реализация. Определяется с помощью ключевого слова abstract.

```
abstract class ShopProduct {
    abstract public function getPrice();
}
```

Абстрактный класс – класс, на основе которого нельзя создать экземпляр объекта. Определяется с помощью ключевого слова abstract.

Класс, который содержит по крайней мере один абстрактный метод, должен быть определен как абстрактный.

В любом классе, который расширяет абстрактный класс, должны быть реализованы все абстрактные методы либо сам класс должен быть объявлен абстрактным.

Пример

```
abstract class CParser {
    abstract public function parse();
}

class XMLParser extends CParser {
    public function parse() {
        # Используем средства для парсинга XML
    }
}

class CSVParser extends CParser {
    public function parse() {
        # Используем функции для парсинга CSV
    }
}
```

Мы гарантируем, что каждая конкретная реализация парсера будет иметь метод parse, который вернет данные в оговоренном формате, при этом мы ничего не знаем об исходном формате данных.

Зачем нужны интерфейсы?

Интерфейс – объявляется также как и обычный класс, но с использованием ключевого слова interface.

Состоит только из абстрактных методов и констант.

```
interface IUser {
    public function authenticate();
    public function getId();
    public function getName();
}
```

Все методы, определенные в интерфейсе должны быть публичными.

Часто термин *наследование* заменяют термином *реализация*.

Для реализации интерфейса используется оператор implements.

Классы могут реализовывать более одного интерфейса.

```
class CUser implements IUser, ArrayAccess {
    # Здесь реализация
}
```

Как запретить переопределение метода?

Чтобы предотвратить переопределение метода в дочерних классах, необходимо использовать ключевое слово final перед объявлением метода.

Такие методы называются *завершенными*.

```
abstract class CParser {
    abstract public function parse();

    final public function toArray() {
        return $this->parse();
    }

    final public function toJson() {
        return json_encode($this->parse());
    }
}
```

Как запретить наследование?

Если класс определяется с ключевым словом final, то он называется *завершенным*.

Для завершенного класса нельзя создать подкласс.

```
# Класс объявлен завершенным

final class XMLParser extends CParser {
    public function parse() {}
}

# PHP выдаст фатальную ошибку
# Класс не может быть унаследован от завершенного класса
class ExtXMLParser extends XMLParser {}
```

Что такое магические методы?

Определение

Магические методы – это методы, вызов которых происходит неявно, когда удовлетворяются соответствующие условия.

Название этих методов начинается с *двух символов подчеркивания*.

Все магические методы должны быть объявлены *публичными* (public).

Конструктор и Деструктор

void __construct ([mixed args [, ...]])

Вызывается автоматически при создании экземпляра объекта.

void __destruct (void)

Вызывается автоматически при освобождении всех ссылок на определенный объект или при завершении скрипта.

```php
class MyDestructableClass {
    public function __construct() {
        $this->name = 'MyDestructableClass';
    }

    public function __destruct() {
        echo 'Уничтожается ' . $this->name . "\n";
    }
}

$obj = new MyDestructableClass();
unset($obj);
```

Что такое перегрузка?

Перегрузка[1] – в PHP означает возможность динамически "создавать" свойства и методы.

Эти динамические сущности обрабатываются с помощью *"волшебных" методов*.

Методы перегрузки вызываются при взаимодействии с теми свойствами и методами, которые не были объявлены или не видны в текущей области видимости.

[1] Интерпретация "перегрузки" в PHP отличается от остальных ОО языков. Традиционно перегрузка означает возможность иметь множество одноименных методов с разным количеством или различными типами аргументов.

Перегрузка свойств

mixed __get (string name)

Выполняется при обращении к неопределенному свойству.

void __set (string name , string value)

Выполняется когда неопределенному свойству присваивается значение.

bool __isset (string name)

Выполняется при вызове функции isset() для неопределенного свойства.

void __unset (string name)

Выполняется при вызове функции unset() для неопределенного свойства.

```php
class PropertyTest {
    private $data = array();

    public function __get($name) {
        return isset($this->data[$name]) ? $this->data[$name] : null;
    }

    public function __set($name, $value) {
        $this->data[$name] = $value;
    }

    public function __isset($name) {
        return isset($this->data[$name]);
    }

    public function __unset($name) {
        unset($this->data[$name]);
    }
}

$obj = new PropertyTest();

$obj->a = 1;
echo $obj->a;

var_dump(isset($obj->a));
unset($obj->a);
```

Перегрузка методов

mixed __call (string name , array arguments)

Выполняется в контексте объекта при обращении к неопределенному методу.

static mixed __callStatic (string name , array arguments)

Выполняется в статическом контексте при обращении к неопределенному методу.

Пример

```php
class MethodTest {
    public function __call($name, $arguments) {
        echo "Вызов метода '$name' "
             . implode(', ', $arguments) . "\n";
    }

    static public function __callStatic($name, $arguments) {
        echo "Вызов статического метода '$name' "
             . implode(', ', $arguments) . "\n";
    }
}

$obj = new MethodTest();

$obj->runTest('в контексте объекта');
$obj::runTest('в статическом контексте');
```

Как преобразовать объект в строку?

string __toString (void)

Выполняется при преобразовании объекта в строку.

Нельзя бросить исключение из метода __toString()

Попытка это сделать закончится фатальной ошибкой

```php
class ShopProduct {
    public $title;
    private $price = 0;

    public function __toString() {
        return $this->name . ': ' . $this->price . ' руб.';
    }
}

$product = new ShopProduct();
$product->title = 'Расческа для усов';
$product->price = 666;

echo $product; # Выведет 'Расческа для усов: 666 руб.'
```

Как клонировать объекты?

Объекты всегда передаются в функцию или присваиваются переменной по ссылке. Копию объекта можно создать с помощью оператора *clone*.

PHP создает *плоскую копию* объекта – любые свойства, являющиеся ссылками на другие переменные, останутся ссылками.

void __clone (void)

Вызывается у *свежесозданной копии* объекта для изменения всех необходимых свойств.

Пример

```php
class SubObject {
    public $instance;
    static private $instances = 0;

    public function __construct() {
        $this->instance = ++self::$instances;
    }

    public function __clone() {
        $this->instance = ++self::$instances;
    }
}

$obj = new SubObject();
$obj2 = clone $obj;

var_dump($obj->instance);  # Выведет '1'
var_dump($obj2->instance); # Выведет '2'
```

Как сериализовать объекты?

string serialize (mixed value)

Возвращает строковое представление любого значения, которое может быть сохранено в PHP.

mixed unserialize (string str)

Возвращает значение сериализованной ранее переменной.

array __sleep (void)

Выполняется перед любой операцией сериализации serialize().

void __wakeup (void)

Выполняется после любой операции десериализации unserialize().

Для того, чтобы корректно выполнить unserialize() для объекта нужно чтобы класс этого объекта был определен заранее

```php
class Connection {
    protected $pdo;
    private $server, $username, $password, $db;

    public function __construct($server, $username, $password, $db) {
        $this->server = $server;
        $this->username = $username;
        $this->password = $password;
        $this->db = $db;
        $this->connect();
    }

    public function connect() {
        $dsn = 'mysql:dbname=' . $this->db . ';host=' . $this->server;
        $this->pdo = new PDO($dsn, $this->username, $this->password);
    }

    public function __sleep() {
        return array('server', 'username', 'password', 'db');
    }

    public function __wakeup() {
        $this->connect();
    }
}
```

Автозагрузка

Что такое автозагрузка классов?

Большинство разработчиков ОО приложений используют соглашение: каждый класс хранить в отдельно созданном для него файле, где имя файла совпадет с именем класса.

Минус – большой объем подключаемых файлов.

Плюс – можно организовать простую автозагрузку классов в приложение, которая сработает при использовании ранее неопределенного класса или интерфейса.

```
boll spl_autoload_register (
    callable autoload_function [, bool throw [, bool prepend ]]
)
```

Регистрирует заданную функцию в spl стеке метода __autoload().

Создает очередь из функций автозагрузки в порядке их определения в исходном коде.

Стандарт PSR-4

Стандарт PSR-4 описывает различные способы и содержит требования, которым необходимо следовать для совместимости автозагрузчиков.

1. Термин class относится к классам, интерфейсам, трейтам и другим подобным структурам.
2. Полное имя класса имеет следующий вид:
   \<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>
   1. каждое пространство имен должно иметь пространство верхнего уровня (известно как «Производитель»);
   2. может иметь столько подпространств, сколько необходимо;
   3. завершается именем класса;
   4. подчеркивания не имеют специального значения в любой части полного имени класса;
   5. состоит из любых алфавитно-цифровых символов;
   6. именуются в CamelCase нотации.

Стандарт PSR-4

1. В момент автозагрузки файла:
   1. существует по крайней мере одна директория, которая соответствует имени пространства имен верхнего уровня;
   2. каждый разделитель пространств имен будет заменен на DIRECTORY_SEPARATOR, когда будет загружаться из файловой системы. Регистр символов важен!;
   3. файлы с классами имеют расширение .php.
2. Автозагрузчик не должен генерировать исключения, либо другие ошибки, а также не должен возвращать значений.

Standart PSR-4 on GitHub

Исключения

Что такое исключения?

Исключения – это специальное средство, позволяющее передать в вызывающий код возникшие ошибки или исключительные ситуации.

Если код встречает неожиданную ситуацию и не знает, как ее обработать, то он генерирует исключение.

«Я не знаю, что с этим делать.
Надеюсь кто-нибудь другой знает, как на это реагировать!»

Как сгенерировать исключение?

Сгенерировать исключение можно при помощи оператора throw.

Поймать (обработать) исключение можно с помощью оператора catch.

Код генерирующий исключение, должен быть окружен блоком try.

В PHP >= 5.5 после блока catch, можно использовать блок finally.
Блок finally всегда выполняется после блоков try и catch.

Пример

```php
function inverse($x) {
    if (!$x) {
        throw new Exception('Деление на ноль!');
    }
    return 1/$x;
}

try {
    echo inverse(5) . "\n";
} catch (Exception $e) {
    echo 'Поймано исключение: ', $e->getMessage(), "\n";
} finally {
    echo "Блок finally.\n";
}
```

Как создать свой тип исключения?

Исключение в PHP – это специальный объект, который является экземпляром встроенного класса Exception или его производного класса.

Объекты типа Exception предназначены для хранения информации об ошибках и выдачи сообщений о них.

Методы объекта Exception

getMessage() – возвращает текст исключения (сообщение об ошибке).

getCode() – возвращает код ошибки.

getFile() – возвращает имя файла, в котором было брошено исключение.

getLine() – возвращает номер строки, в которой было брошено исключение.

getTrace() – возвращает многомерный массив, отслеживающий вызовы метода, которые привели к исключению, включая имя метода, класса, файла и значение аргумента.

Конец