Разработка многостраничного сайта на PHP

ТЕМА 4.2 РАСШИРЕННЫЕ МЕТОДЫ
ЗАНЯТИЕ № 3 - ЛЕКЦИЯ

Тема занятия – Расширенные методы

Цель –

Изучить методы , которые повышают производительность, использование и безопасность приложений

Актуализация

На прошлых занятиях мы изучили архитектуру приложений, шаблоны проектирования в PHP и решали большое количество практических задач.

Теперь нам предстоит познакомится с расширенными методами, используемые в PHP для улучшения веб-сайта.

Содержание

- Введение
- PSR - основные стандарты, принципы, пример использования
- CLI – как использовать в PHP, пример использования
- PSR6 – общий интерфейс для кэширования, пример использования
- Примеры проектов с использованием расширенных методов PHP
- Заключение
- Список рекомендуемой литературы

Введение

PHP — один из самых популярных языков программирования, предназначенный для создания веб-сайтов и приложений. Однако PHP не ограничивается только базовыми функциями и возможностями. В PHP есть расширенные методы, которые значительно повышают производительность, использование и безопасность приложений.

В этой презентации мы познакомимся с несколькими расширенными методами PHP, таких как PSR, CLI и PSR6. Мы рассмотрим, что они пересылают, зачем они нужны и как их можно использовать в своих проектах. Мы также рассмотрим примеры проектов, используя эти методы, и представляем примеры кода, чтобы вы могли лучше понять, как использовать их в своих приложениях.

PSR

PSR - это набор оснований и эскизов для организации кода на PHP.

Все стандарты имеют префикс PSR и нумеруются по мере их разработки.

Основные стандарты, которые вы можете встретить, это:

1) PSR-1,
2) PSR -2,
3) PSR -3,
4) PSR -4 и
5) PSR P-12.

PSR

PSR-1 решает, как организовать код на PHP, включая имена, функции и классы, отступы и комментарии.

Например, следуя PSR-1, вы можете именовать классы в формате CamelCase, а методы и свойства - в формате camelCase:

PSR

PSR-2 определяет правила форматирования кода на PHP, такие как использование отступов, перенос строк и т.д.

Например, следуя PSR-2, вы можете использовать 4 пробела для отступов и разбивать длинные строки на несколько строк:

PSR

PSR-3 определяет стандарты для логирования на PHP. Этот стандарт соответствует правилам использования интерфейса LoggerInterface.

Например, вы можете использовать стандартный интерфейс логирования для регистрации сообщений в приложении:

```php
<?php
use Psr\Log\LoggerInterface;

class MyClass {
    private $logger;

    public function __construct(LoggerInterface $logger) {
        $this->logger = $logger;
    }

    public function myMethod() {
        $this->logger->info('My message');
    }
}


?>
```

PSR

PSR-4 определяет стандарты для автозагрузки классов на PHP. Этот стандарт определяется тем, как должны быть организованы файлы с классами и пространствами имен.

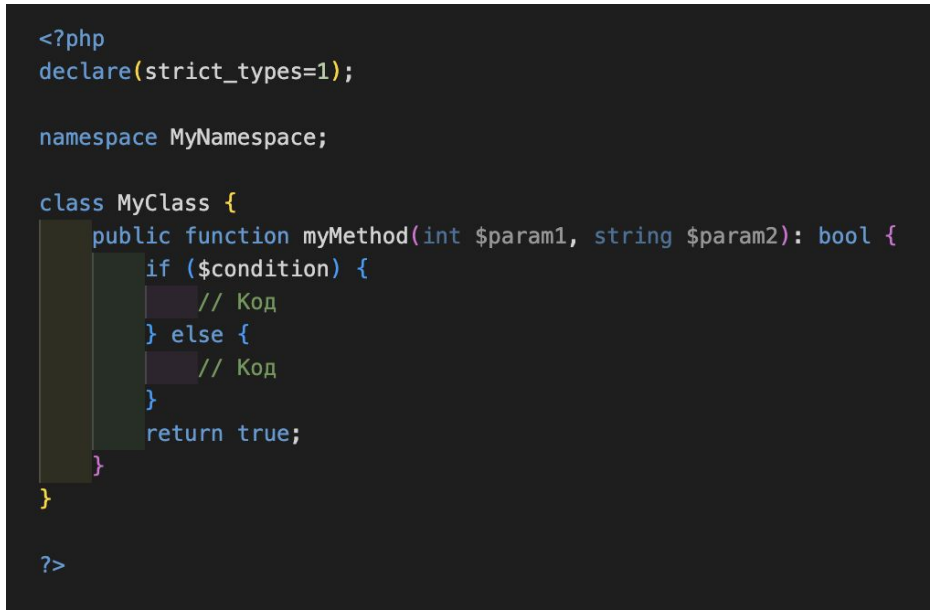
Например, вы можете использовать стандарт PSR-4 для автозагрузки классов в приложении:

```php
<?php
use MyNamespace\MyClass;

$obj = new MyClass();

?>
```

PSR

PSR-12 - это недавно созданный стандарт, который включает рекомендации PSR-1 и PSR-2, а также включает новые рекомендации, такие как использование строгой типизации и др.

Вот несколько примеров, как следовать PSR-12:



```php
<?php
declare(strict_types=1);

namespace MyNamespace;

class MyClass {
    public function myMethod(int $param1, string $param2): bool {
        if ($condition) {
            // Код
        } else {
            // Код
        }
        return true;
    }
}

?>
```

CLI

CLI (интерфейс командной строки) - это способ взаимодействия пользователя с приложением через командную строку.

В PHP это реализуется с помощью встроенной библиотеки php-cli.

Для использования CLI в PHP необходим PHP-скрипт, который будет вводить пользователь через командную строку. Вот пример такого скрипта:

```php
<?php
// Получаем аргументы командной строки
$arg1 = $argv[1];
$arg2 = $argv[2];

// Выводим результат работы
echo "Аргумент 1: $arg1\n";
echo "Аргумент 2: $arg2\n";
?>
```

CLI

Для запуска скрипта через CLI необходимо открыть терминал и выполнить команду:

```
php script.php arg1 arg2
```

Где script.php- название файла скрипта, arg1и arg2- аргументы, допустимые скрипту.

CLI

CLI в PHP используется для запуска скриптов из командной строки, для решения задач, для запуска тестов, для настройки приложений и многих других задач.

Пример использования CLI в PHP — это PHPUnit, популярный фреймворк для юнит-тестирования PHP-кода.

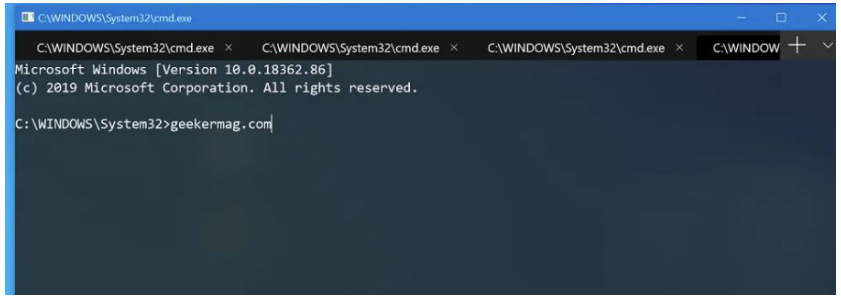
PHPUnit может воспроизводиться из командной строки и автоматически запускаться с набором тестов, написанных с помощью PHPUnit.

CLI : терминал

Чтобы запустить PHP-скрипт через CLI, необходимо открыть терминал (или командную строку) на компьютере. Как открыть терминал, зависит от системы.

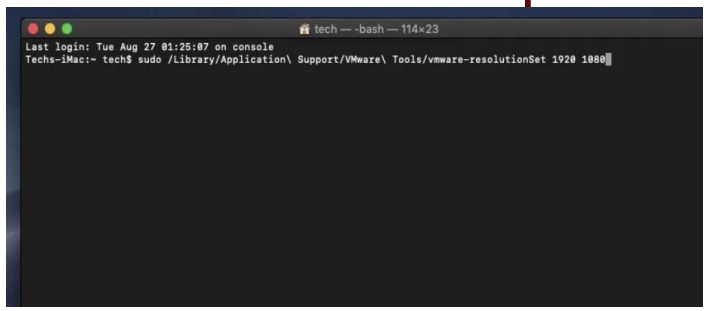
Вот несколько примеров:

- В ОС Windows: можно найти терминал через меню Пуск или сочетание клавиш Win + R и включение команды «cmd». Также можно использовать сторонние терминалы, такие как Git Bash или PowerShell.

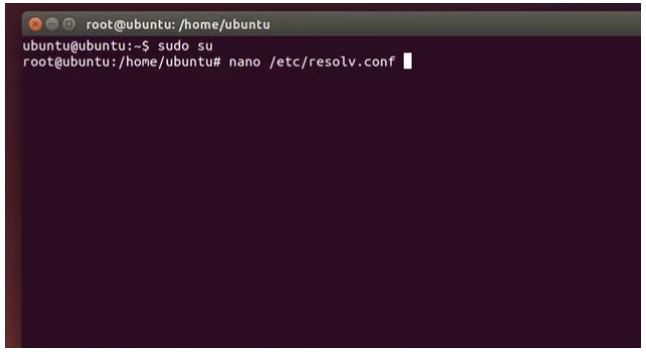


CLI : терминал

- На ОС Mac OS: можно найти терминал в папке "Утилиты" или сочетание клавиш Command + Пробел и вывод запроса "терминал".



- На ОС Linux: разные дистрибутивы Linux имеет различные терминалы. Например, в Ubuntu можно использовать сочетание клавиш Ctrl + Alt + T, чтобы открыть терминал.



CLI : терминал

Когда терминал, нужно открыть в каталоге, где находится PHP скрипт, используя команду cd(Change Directory), и затем использует скрипт, используя команду php и указав имя файла скрипта:

```
cd /path/to/script/
php script.php
```

Здесь /path/to/script/- это путь к директории, где находится скрипт, а script.php- имя файла скрипта.

PSR-6

PSR-6 - это стандарт обнаружения по внешнему унифицированному интерфейсу кэширования для PHP-приложений. Он описывает общий набор методов и интерфейсов, которые должны быть реализованы любой библиотекой кэширования, чтобы обеспечить совместимость и возможность переиспользования.

Зачем нужен PSR-6?

Кэширование - это один из инструментов для улучшения производительности веб-приложений. Он позволяет получить результаты вычисления или поступления в память или на диске и использовать их повторно, вместо того, чтобы каждый раз выполнять операцию операции. PSR-6 помогает сделать кэширование более универсальным, позволяя разработчикам выбирать и переключаться между различными реализациями кэширования без необходимости перезаписи кода.

PSR-6

Основные методы, которые должны быть реализованы в соответствии с PSR-6, включают:

- get($key): значения из кэша по ключу
- set($key, $value, $ttl = null): сохранение значения в кэше под ключом с опциональным временем жизни (Time-To-Live)
- delete($key): удаление значения из кэша по ключу
- clear(): очистка всего кэша

PSR-6

Пример использования PSR-6 в PHP может выглядеть следующим образом:

```php
<?php
// Подключение реализации кэширования
use Symfony\Component\Cache\Adapter\FilesystemAdapter;

// Создание экземпляра кэша
$cache = new FilesystemAdapter();

// Запись значения в кэш с ключом "my_key" и временем жизни 60 секунд
$cache->set('my_key', 'my_value', 60);

// Получение значения из кэша по ключу "my_key"
$value = $cache->get('my_key');

// Удаление значения из кэша по ключу "my_key"
$cache->delete('my_key');

// Очистка всего кэша
$cache->clear();
?>
```

PSR-6

```php
<?php
// Подключение реализации кэширования
use Symfony\Component\Cache\Adapter\FilesystemAdapter;

// Создание экземпляра кэша
$cache = new FilesystemAdapter();

// Запись значения в кэш с ключом "my_key" и временем жизни 60 секунд
$cache->set('my_key', 'my_value', 60);

// Получение значения из кэша по ключу "my_key"
$value = $cache->get('my_key');

// Удаление значения из кэша по ключу "my_key"
$cache->delete('my_key');

// Очистка всего кэша
$cache->clear();
?>
```

Здесь мы используем кэширование FilesystemAdapter библиотек Symfony. Мы создаем экземпляр кэша, записываем значение в кэше с ключом "my_key" и временем жизни 60.

Примеры использования.
Пример № 1. Кэширование результатов поступления к API

Представим, что наш проект делает множество форумов к API внешнего сервиса, чтобы получить данные, необходимые для отображения на странице. Однако эти запросы могут занимать много времени, особенно если у нас большой трафик на сайте.

Чтобы увеличить загрузку страниц, мы можем использовать кэширование результатов. PSR6 предоставляет общий интерфейс для кэширования, который можно использовать для создания простого кэширования.

Пример № 1

```php
<?php
use Psr\Cache\CacheItemPoolInterface;

class ApiData {
    private $cache;

    public function __construct(CacheItemPoolInterface $cache) {
        $this->cache = $cache;
    }

    public function getApiData($url) {
        $cacheKey = md5($url);

        $cacheItem = $this->cache->getItem($cacheKey);
        if ($cacheItem->isHit()) {
            return $cacheItem->get();
        }

        $data = file_get_contents($url);

        $cacheItem->set($data);
        $cacheItem->expiresAfter(3600);

        $this->cache->save($cacheItem);

        return $data;
    }
}
?>
```

Пример № 1

Разбор кода:

Данный код представляет собой класс ApiData, который использует интерфейс CacheItemPoolInterface из стандарта PSR-6 для кэширования результатов запросов к API.

В конструкторе класса используется экземпляр кэша, который должен реализовать интерфейс CacheItemPoolInterface. Это может быть любая реализация кэша, целевой интерфейс.

Метод getApiData() используется для входа по URL-адресу, откуда необходимо получить данные. Пример для этого URL-адреса использует ключ кэширования с помощью функции md5(), что приводит к тому, что уже есть данные в кэше для данного ключа. Если есть, то они возвращаются из кэша, а если нет, то высокий запрос к API, данные в кэше и возвращаются.

Пример №1

Разбор кода (продолжение):

В кэше данных находятся в виде объекта CacheItemInterface, который представляет собой отдельный элемент кэша. Методы set() и expiresAfter() требуют значения элемента и времени его жизни соответственно. Метод save() сохраняет элемент в кэше.

Такой подход позволяет избежать встреч с API при вызовах метода getApiData() для одного и того же URL. Также он повышает производительность приложений и вычисляет результаты на API-сервере.

Пример №1

В этом случае мы использовали CacheItemPoolInterface PSR6 для кэширования результатов. Мы сохраняем данные в кэше один раз, чтобы не перенаправлять внешний сервис лишними запросами.

Пример №2. Использование Composer для управления зависимостями

Composer - это менеджер зависимостей для PHP. Он позволяет управлять зависимостями вашего проекта и легко добавлять новые пакеты.

Вот пример файла composer.json, который определяется в зависимости от проекта:

Пример №2

Мы наблюдаем две зависимости: monolog/monologи league/csv. ^2.0и ^8.0- это использование версии, которые говорят Composer, какие версии пакетов мы хотим.

Когда мы устанавливаем эти зависимости с помощью Composer, он загружает эти пакеты и устанавливает их в нашем проекте. Мы можем использовать их в нашем коде, предполагаемую строку:

```php
<?php
require __DIR__ . '/vendor/autoload.php';

use Monolog\Logger;
use Monolog\Handler\StreamHandler;
use League\Csv\Reader;

$log = new Logger('name');
$log->pushHandler(new StreamHandler('path/to/your.log', Logger::WARNING));

$csv = Reader::createFrom
?>
```

Пример № 2

Разбор кода:

Код, представленный в этом появлении, использует две библиотеки - Monolog и League\Csv. Первый используется для логирования, а второй - для чтения данных из CSV-файлов.

Первые две нагрузки загружают автозагрузчик классов Composer, который позволяет загружать классы, зависимые от зависимостей приложений.

Далее мы создаем экземпляр класса Monolog\Logger, который будет вестись логированием сообщений. В качестве первого параметра мы передаем имя регистратора (в случае использования 'name'). Затем мы добавили обработчик Monolog\Handler\StreamHandler, который записывает сообщения с улучшением Logger::WARNING в файл журнала, указанный во втором параметре.

Пример № 2

Разбор кода(продолжение):

Затем мы создаем объект League\Csv\Reader со статистическим методом createFrom, который использует путь к файлу CSV в качестве параметра. Объект Reader позволяет нам читать итераторы строк CSV-файла.

В этом смысле код не завершен, и она не может быть осуществлена. Однако после объекта Reader мы можем использовать различные методы, такие как getHeader(), fetchOne(), fetchAssoc() и т.д., для создания данных из CSV-файла.

Пример № 3. Использование PSR-4

Необходимо создать стандартную структуру папоротников и классов в соответствии с PSR-4, настроить каталог src/и в ней каталог MyApp/.

В этой директории создайте файл MyClass.phpc классом MyApp\MyClass. Добавьте метод greet()в класс, который будет возвращать статью "Hello, World!".

Пример № 3

Файл composer.json должен выглядеть так:

```
{
    "autoload": {
        "psr-4": {
            "MyApp\\": "src/"
        }
    }
}
```

Файл src/MyApp/MyClass.php должен выглядеть так:

```
<?php
namespace MyApp;

class MyClass {
    public function greet() {
        return "Hello, World!";
    }
}
?>
```

Пример № 3

Затем выполните composer install и создайте файл index.php, который будет использовать класс MyClass:

```php
<?php
require __DIR__ . '/vendor/autoload.php';

use MyApp\MyClass;

$obj = new MyClass();
echo $obj->greet();
?>
```

Пример № 4. Использование PSR-6

Необходимо создать класс, который будет использовать PSR-6 для кэширования данных, настроить класс CacheManager, который будет использовать объект, реализующий интерфейс CacheItemPoolInterface, для кэширования данных.

Добавьте метод get()в класс, который будет получать данные из кэша, если они там есть, иначе - из источника данных. Добавьте метод set()в класс, который будет запоминать данные в кэше.

Пример № 4

```php
<?php
use Psr\Cache\CacheItemPoolInterface;

class CacheManager {
    private $cache;

    public function __construct(CacheItemPoolInterface $cache) {
        $this->cache = $cache;
    }

    public function get($key, callable $callback) {
        $cacheItem = $this->cache->getItem($key);
        if ($cacheItem->isHit()) {
            return $cacheItem->get();
        }

        $value = $callback();
        $cacheItem->set($value);
        $this->cache->save($cacheItem);

        return $value;
    }

    public function set($key, $value) {
        $cacheItem = $this->cache->getItem($key);
        $cacheItem->set($value);
        $this->cache->save($cacheItem);
    }
}
?>
```

Пример №4

Разбор кода:

Данный код представляет собой класс CacheManager, который использует интерфейс PSR-6 для кэширования.

В конструкторе происходит инъекция в зависимости от объекта, реализующего интерфейс CacheItemPoolInterface. Это позволяет получить любой объект кэширования, который соответствует стандарту PSR-6, например, объекту класса RedisCache.

Пример №4

Разбор кода(продолжение):

Метод get позволяет получить значение из кэша по ключу. Если значение уже есть в кэше, то метод вернет его, в данном случае будет вызвана переданная аргументом функция callback, которая должна вернуть значение для кэширования. Полученное значение будет сохранено в кэше и возвращено методом.

Метод set позволяет установить значение в кэше по заданному ключу. Если такое значение с ключом уже существует, то оно будет перезаписано.

Пример № 4

Далее можно использовать классификацию CacheManager:

```php
<?php
use Symfony\Component\Cache\Adapter\FilesystemAdapter;

$cache = new FilesystemAdapter();
$cacheManager = new CacheManager($cache);

$data = $cacheManager->get('my_data', function() {
    // Код для получения данных из источника данных
    return $data;
});
?>
```

Пример № 4

Разбор кода:

В этом случае мы создаем объект кэширования на основе класса FilesystemAdapter, затем создаем объект класса CacheManager, пересылаем в него объект кэширования. Далее вызывается метод get, который пытается получить значение из кэша по ключу key. Если значение имеет значение в кэше, то оно будет возвращено методом, в случае использования будет функция getDataFromDatabase, которая предоставит данные из данных и вернет их. Полученные данные сохраняются в кэше и возвращаются методом. После этого вызывается метод setсохранения данных в кэше.

Заключение

В заключении можно сказать, что с помощью расширенных методов в PHP получено большое количество приложений и выпущена их производительность. Библиотеки, реализующие стандарты PSR, облегчают разработку, улучшают качество кода и обеспечивают совместимость между бесплатными проектами.

CLI-интерфейс в PHP позволяет управлять приложением через командную строку, что удобно для автоматизации рутинных задач и испытаний.

Понимание и использование расширенных методов и приложений PHP является важным требованием для профессиональной деятельности и позволяет создавать более масштабируемые приложения.

Рефлексия

Сегодня на уроке мы обсудили новую тему : “Расширенные методы в PHP”, узнали многие методы, но это еще не весь список. Другие методы мы узнаем с вами на практических занятиях.

А пока ответьте на несколько вопросов, вспоминая этот урок:

1) Какие расширенные методы вы запомнили?
2) Какой метод вам понравился больше всего?
3) Какой метод вам показался наиболее сложным и непонятным?
4) И, конечно же, какие у вас остались вопросы?

Список используемой литературы

1. «PHP: The Right Way» — официальное руководство по полезности и рекомендации в PHP.
2. "Modern PHP: New Features and Good Practices" Джоша Локхарта - книга о новых возможностях PHP и современных подходах к разработке.
3. «Объекты, шаблоны и практика PHP», автор Matt Zandstra — книга о сокровищах объектно-ориентированного программирования в PHP.
4. "Learning PHP Design Patterns" Уильяма Сандерса - книга о применении паттернов проектирования в PHP.
5. "Mastering PHP Design Patterns" Джунад Али - книга о продвинутых технологиях использования паттернов проектирования в PHP.
6. "The Joy of PHP" от Алана Форбса - книга о расширенных возможностях PHP и примерах использования.
7. "PHP 7 в простых шагах" от Mike McGrath - книга о новых функциях и возможностях в PHP 7.
8. "Pro PHP Refactoring" от Francesco Trucchia - книга о техниках рефакторинга кода на PHP.

Спасибо за внимание