1. **ОПИСАНИЕ ЯЗЫКА PHP**

PHP (рекурсивный акроним словосочетания *PHP: Hypertext Preprocessor*) - язык программирования общего назначения с открытым исходным кодом. PHP разработан специально для ведения Web-разработок и его код может внедряться напрямую в HTML.

Язык PHP является серверным языком. Серверные языки во многом похожи на JavaScript. PHP отличается от JavaScript тем, что PHP-скрипты выполняются на сервере и генерируют HTML, который посылается клиенту. Клиент получает только результат выполнения скрипта, но не может узнать, какой именно код его произвел.

PHP довольно прост в освоении, но тем не менее способен удовлетворить запросы профессиональных программистов.

Можно выделить основные области применения PHP.

**Создание скриптов для выполнения на стороне сервера.** PHP традиционно и наиболее широко используется именно таким образом. Для этого вам будут необходимы три вещи. Интерпретатор PHP (в виде программы CGI или серверного модуля), веб-сервер и браузер. Для того чтобы можно было просматривать результаты выполнения PHP-скриптов в браузере, нужен работающий веб-сервер и установленный PHP. Просмотреть вывод PHP-программы можно в браузере, получив PHP-страницу, сгенерированную сервером. В случае, если вы просто экспериментируете, вы вполне можете использовать свой домашний компьютер вместо сервера.

**Создание скриптов для выполнения в командной строке.** Вы можете создать PHP-скрипт, способный запускаться без сервера или браузера. Все, что вам потребуется - парсер PHP. Такой способ использования PHP идеально подходит для скриптов, которые должны выполняться регулярно, например, с помощью cron (на платформах \*nix или Linux) или с помощью планировщика задач (Task Scheduler) на платформах Windows. Эти скрипты также могут быть использованы в задачах простой обработки текстов.

**Создание оконных приложений, выполняющихся на стороне клиента.** Возможно, PHP является не самым лучшим языком для создания подобных приложений, но, если вы очень хорошо знаете PHP и хотели бы использовать некоторые его возможности в своих клиентских приложениях, вы можете использовать PHP-GTK для создания таких приложений. Подобным образом вы можете создавать и кросс-платформенные приложения. PHP-GTK является расширением PHP и не поставляется вместе с основным дистрибутивом PHP.

Работа с PHP предоставляет свободу выбора операционной системы и веб-сервера. Кроме того, появляется выбор между использованием процедурного или объектно-ориентированного программирования (ООП) или же их сочетания.

Одним из главных преимуществ PHP является поддержка большого круга баз данных. Можно воспользоваться расширением, предназначенным для отдельной базы данных (например mysql) или воспользоваться уровнем абстракции от базы данных, при помощи PDO. Так же можно подсоединиться к любой базе данных, поддерживающей Открытый Стандарт Соединения Баз Данных (ODBC), при помощи одноименного расширения ODBC. Для других баз данных, таких как CouchDB, есть возможность воспользоваться cURL или сокетами.

1. **СИНТАКСИС ЯЗЫКА PHP**

Синтаксис языка PHP во многом похож на такие языки как JavaScript, C, C++, C#, Objective-C, Java, Perl и другие Си-подобные языки.

PHP-скрипт строится из наборов команд или выражений, которые в свою очередь представляют инструкцию, которая выполняется до того, как веб-сервер переходит к следующей команде. Все выражения в PHP должны заканчиваться точкой с запятой.

Весь PHP код заключается между тегами <?php и ?>. Если файл содержит только код PHP, предпочтительно опустить закрывающий тег в конце файла. Это помогает избежать добавлени случайных символов пробела или перевода строки после закрывающего тега PHP, которые могут послужить причиной нежелательных эффектов, так как PHP начинает выводить данные в буфер при отсутствии намерения у программиста выводить какие-либо данные в этой точке скрипта.

|  |
| --- |
| <?php echo "Hello world";  // ... еще код  echo "Последнее выражение";  // Скрипт заканчивается тут без закрывающего тега PHP |

Все, что находится вне пары открывающегося и закрывающегося тегов, игнорируется интерпретатором PHP, у которого есть возможность обрабатывать файлы со смешанным содержимым. Это позволяет PHP-коду быть встроенным в документы HTML, к примеру, для создания шаблонов.

|  |
| --- |
| <p>Это будет проигнорировано PHP и отображено браузером.</p> <?php echo 'А это будет обработано.'; ?> <p>Это тоже будет проигнорировано PHP и отображено браузером.</p> |

PHP поддерживает комментарии в стиле 'C', 'C++' и оболочки Unix (стиль Perl). Например:

|  |
| --- |
| <?php     echo "Это тест"; // Это однострочный комментарий в стиле c++     /\* Это многострочный комментарий        еще одна строка комментария \*/     echo "Это еще один тест";     echo "Последний тест"; # Это комментарий в стиле оболочки Unix ?> |

* 1. Типы данных

В PHP осуществлена поддержка восьми простых типов.

Из них:

- Четыре скалярных типа:

* Boolean
* Integer
* Float
* String

- Два смешанных типа:

* array
* object

- Два специальных типа:

* resource
* NULL

В PHP нет необходимости явно указывать тип объявляемой переменной. Тип переменной определяется в контексте ее использования.

* 1. Выражения

Выражения - это самые важные строительные элементы PHP. Практически все что пишется в программе – это выражения.

Самый простой пример выражения – это переменная или константа. Например:

|  |
| --- |
| $a = 5 |

Более сложными примерами выражений являются функции. Например:

|  |
| --- |
| <?php function foo () {     return 5; } ?> |

* + 1. Логические выражения

Логические выражения – это выражения которые могут принимать только два значения: ложь и истина (0 и 1). Эти выражение чаще всего возникают при использовании операторов >, <, == (равно), || (логическое ИЛИ), && (логическое И), ! (логическое НЕ) и другие. Например:

|  |
| --- |
| $less = 10 < 5; // $less - false |
| $equals = $b == 1; // $equals — true, если $b == 1 |
| $between - $b>=l && $b<=10 // $between - true, если $b от 1 до 10 |
| $x = !($b || $c) && $d; // true, если $b и $с ложны, a $d - истинно |

* + 1. Строковые выражения

В языке PHP строки являются одним из основных объектов. Они могут содержать текст вместе с символами форматирования или даже бинарные данные. Например:

|  |
| --- |
| $multiline = "Это текст, начинающийся на одной строке  и продолжающийся на другой,  третьей и т. д."; |

* 1. Операции

В PHP различают следующие типы операций:

- арифметические операции;

- строковые операции;

- операции присваивания;

- операции инкремента и декремента;

- битовые операции;

- операции сравнения;

- операция эквивалентности;

- логические операции;

- операция отключения предупреждений.

* 1. Конструкции языка
     1. Условный оператор if-else

Синтаксис условного оператора:

|  |
| --- |
| if {логическое\_выражение)  инструкция\_1;  else  инструкция\_ 2; |

Действие инструкции следующее: если логическое\_выражение истинно, то выполняется инструкция\_1, а иначе — инструкция\_2. Как и в любом другом языке, конструкция else может опускаться, в этом случае при получении ложного значения просто ничего не делается.

Пример:

|  |
| --- |
| if ($salary>=100 && $salary<=5000) echo "Вам еще расти и расти"  else echo "Ну и правильно — не в деньгах счастье." |

* + 1. Цикл с предусловием while

Осуществляет цикличное выполнение команд в теле цикла, включающее предварительную проверку, нужно ли это делать (истинно ли логическое выражение в заголовке). Если не нужно (выражение ложно), то конструкция заканчивает свою работу, иначе выполняет очередную итерацию и начинает все сначала. Выглядит цикл так:

|  |
| --- |
| while (логическое\_выражение)  инструкция; |

* + 1. Цикл с постусловием do-while

В отличие от цикла while, этот цикл проверяет значение выражения не до, а после каждого прохода. Таким образом, тело цикла выполняется хотя бы один раз.

Выглядит оператор так:

|  |
| --- |
| do {  команды;  } while (логическое\_выражение) ; |

После очередной итерации проверяется, истинно ли логическое\_выражение, и, если это так, управление передается вновь на начало цикла, в противном случае цикл обрывается.

* + 1. Цикл for

Формат конструкци:

|  |
| --- |
| for {инициализирующе\_команды  тело\_цикла  условие\_цикла  кома нды\_после\_про хода) |

Как только управление доходит до цикла, первым делом выполняются операторы, включенные в инициализируюшпе\_команды (слева направо). Эти команды перечисляются через запятую, например:

|  |
| --- |
| for ($i=0, $j=10, $k="Test!"; ...) |

* + 1. Цикл foreach

Данный тип цикла предназначен специально для перебора всех элементов массива и был добавлен только в четвертой версии языка РНР. (Массив — это набор так называемых ключей, каждому из которых соответствует некоторое значение.) Выглядит он следующим образом:

|  |
| --- |
| foreach {массив as $ключ=>$значение)  команды; |

Здесь команды циклически выполняются для каждого элемента массива, при этом очередная пара ключ=> значение оказывается в переменных $ключ и $ значение.

* + 1. Конструкция switch-case

Для того чтобы избежать использования нескольких идущих подряд конструкций if-eise целесообразно воспользоваться специальной конструкцией switch-case:

|  |
| --- |
| switch {выражение) {  case значение!: команды1; [break;]  case значение2: команды2; [break;]  case значением: командны; [break; ]  [default: команды\_по\_умолчанию; [break]]  } |

Делает она следующее: вычисляет значение выражения (пусть оно равно, например, v), а затем пытается найти строку, начинающуюся с case V:. Если такая строка обнаружена, выполняются команды, расположенные сразу после нее (причем на все последующие операторы case что-то внимания не обращается, как будто их нет, а код после них остается без изменения). Если же найти такую строку не удалось, выполняются команды после default (когда они заданы).

1. **РАБОТА С СУБД MySQL**

База данных — совокупность связанных между собой данных, сохраняемая в двумерных таблицах информационной системы. Программное обеспечение информационной системы, обеспечивающей создание, ведение и совместное использование баз данных, называется системой управления базами данных (СУБД).

Одна из самых популярных СУБД, которые используются в Web-программировании, — MySQL. Она предназначена для создания небольших (сравнительно, конечно — например, в районе 100 Мбайт) баз данных и поддерживает некоторое подмножество языка запросов SQL.

SQL (Structured Query Language, язык структурированных запросов) — это специально разработанный стандарт языка запросов к базам данных. В нем присутствуют такие команды, как:

- создание/удаление таблицы;

- создание новых записей в заданной таблице;

- поиск и группировка записей, в том числе сразу в нескольких связанных между собой таблицах;

- удаление записей, удовлетворяющих некоторому критерию;

- обновление некоторых полей в указанных записях.

MySQL — это программа-сервер, постоянно работающая на компьютере. Клиентские программы (например, сценарии) посылают ей специальные запросы через механизм сокетов (т. е. при помощи сетевых средств), она их обрабатывает и запоминает результат. Затем, также по специальному запросу клиента, весь этот результат или его часть передается обратно.

Механизм использования сокетов подразумевает технологию клиент-сервер, а это означает, что в системе должна быть запущена специальная программа — MySQL- сервер, которая принимает и обрабатывает запросы от программ. Так как вся работа происходит в действительности на одной машине, накладные расходы по работе с сетевыми средствами незначительны (установка и поддержание соединения с MySQL-сервером обходится довольно дешево).

Структура MySQL трехуровневая: базы данных — таблицы — записи. Один сервер MySQL способен поддерживать сразу несколько баз данных, доступ к которым может разграничиваться именем пользователя (login) и паролем (password). Зная эти регистрационные сведения, можно работать с конкретной базой данных. Например, можно создать или удалить в ней таблицу, добавить записи и т. д. Обычно имя-идентификатор и пароль назначаются хостинг-провайдерами, которые и обеспечивают поддержку MySQL для своих пользователей.

* 1. Соединение с базой данных

Прежде чем работать с базой данных, необходимо установить с ней сетевое

соединение, а также провести авторизацию пользователя. Для этого служит функция mysql\_connect().

|  |
| --- |
| int mysql\_connect ([ string $host] [,string $user] [, string $passvrord\) |

Функция mysql\_connect () устанавливает сетевое соединение с базой данных MySQL, расположенной на хосте $host (по умолчанию это localhost, т. е. текущий компьютер), и возвращает идентификатор открытого соединения. Вся дальнейшая работа ведется именно с этим идентификатором. При регистрации указывается имя пользователя $user и пароль $password (по умолчанию имя пользователя, от которого запущен текущий процесс, и пустой пароль). Строка $host также может включать в себя номер порта в формате: имя\_хоста: порт (если сервер MySQL настроен не на стандартный, а на какой-то другой порт, что делать, вообще говоря, не рекомендуется).

Соединение с MySQL-сервером будет автоматически закрыто по завершении работы сценария, либо же при вызове функции mysql\_ciose ().

|  |
| --- |
| int mysql\_select\_db (string $dbname [,int $link\_identifier]) |

До того как послать первый запрос серверу MySQL, необходимо указать, с какой базой данных мы собираемся работать. Для этого и предназначена описываемая функция. Она уведомляет РНР, что в дальнейших операциях с соединением $link\_identifier (или с последним открытым соединением, если указанный параметр не задан) будет использоваться база данных $dbname.

* 1. Запросы к базе данных

Для этого существует одна-единственная функция — mysql\_query() — и возвращает она идентификатор результирующего набора данных.

Любой запрос к MySQL-серверу представляет собой обычную строку, записанную на языке SQL. В этой строке указывается имя команды, а также данные (в апострофах), которые необходимо использовать для модификации записей или поиска в таблицах. Например, следующий SQL-запрос осуществляет поиск всех записей в таблице Music, для которых название композиции равно "Going Under":

|  |
| --- |
| SELECT \* FROM Music WHERE title='Going Under' |

|  |
| --- |
| int mysql\_query (string $ query [,int $link\_identifier]) |

Эта функция посылает MySQL-серверу запрос $query и возвращает идентификатор ответа, или результата. Параметр $query представляет собой строку, составленную по правилам языка SQL. Используется установленное ранее соединение $link\_identifier, а в случае его отсутствия — последнее открытое соединение.

* 1. Пример файла осуществляющего подключение к базе данных

|  |
| --- |
| <?php ## Подключение к СУБД MySQL.  $user = "root";  $pass = "";  $db = "spoon";  // Подключаемся к СУБД MySQL.  mysql\_connect("localhost", $user, $pass)  or die("Could not connect: ".mysql\_error());  // Создаем БД $db — это может делать только суперпользователь!  // Если БД уже существует, будет ошибка, но это не страшно.  @mysql\_query("CREATE DATABASE $db");  // Выбираем БД $db (только что созданную или уже существующую)  mysql\_select\_db($db)  or die("Could not select database: ".mysql\_error());  ?> |

* 1. Язык запросов СУБД MySQL

Язык SQL позволяет создавать довольно сложные запросы. Подчас разобраться в каком-нибудь особенно сложном запросе, затрагивающем сразу несколько таблиц, может только специалист. Составление SQL-запросов — это в некотором роде искусство, и от того, насколько опытен разработчик, часто зависит скорость выполнения запроса.

* + 1. Создание базы данных

|  |
| --- |
| CREATE DATABASE ИмяБазыДанних |

Создает новую базу данных с именем имяБазыДанных. Эта команда доступна только администратору сервера.

* + 1. Создание таблицы

|  |
| --- |
| CREATE [IF NOT EXISTS] TABLE ИмяТаблицы(ИмяПоля тип, ИмяПоля тип, . . .) |

Этой командой в базе данных создается новая таблица с колонками (полями), определяемыми своими именами (имяПоля) и указанными типами. После создания таблицы в нее можно будет добавлять записи, состоящие из перечисленных в данной команде полей.

* + - 1. Типы полей

**Целые числа**

|  |  |
| --- | --- |
| Тип | Описание |
| tinyint | Может хранить числа от -128 до +127 |
| SMALLINT | Диапазон от -32 768 до 32 767 |
| MEDIUMINT | Диапазон От -8 388 608 до 8 388 607 |
| INT | Диапазон от -2 147 483 648 до 2 147 483 647 |
| BIGINT | Диапазон от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 |

**Вещественные числа**

|  |  |
| --- | --- |
| Тип | Описание |
| Float | Число с плавающей точкой небольшой точности |
| Double | Число с плавающей точкой двойной точности |
| Real | Синоним для DOUBLE |
| Decimal | Дробное число, хранящееся в виде строки |
| Numeric | Синоним для DECIMAL |

**Строки**

|  |  |
| --- | --- |
| Тип | Описание |
| TINYTEXT | Может хранить максимум 255 символов |
| TEXT | Может хранить не более 65 535 символов |
| MEDIUMTEXT | Может хранить максимум 16 777 215 символов |
| LONGTEXT | Может хранить 4 294 967 295 символов |

**Бинарные данные**

|  |  |
| --- | --- |
| Тип | Описание |
| tinyblob | Может хранить максимум 255 символов |
| BLOB | Может хранить не более 65 535 символов |
| MEDIUMBLOB | Может хранить максимум 16 777 215 символов |
| LONGBLOB | Может хранить 4 294 967 295 символов |

**Дата и время**

|  |  |
| --- | --- |
| Тип | Описание |
| date | Дата в формате гггг-мм-дд |
| TIME | Время в формате чч: ММ: СС |
| datetime | Дата и время в формате гггг-мм-дц чч:ММ:СС |
| timestamp | Время и дата в формате Unix timestamp. Однако при получении значения поля оно отображается не в формате timestamp, а в виде ггггммдцччммес, что сильно умаляет преимущества его использования в РНР |

**Перечисления**

MySQL поддерживает еще несколько специфических типов данных, использовать

которые в РНР, откровенно говоря, вряд ли целесообразно. Например, тип перечисление (enum) задает, что значение соответствующего поля может быть не любой строкой или числом, а только одним из нескольких указанных при создании таблицы значений: vaiuel, vaiue2 и т. д. Вот как выглядит имя типа перечисления:

|  |
| --- |
| ENUM(vaiuel, value2, value3, . . .) |

**Множества**

В отличие от всех остальных типов данных, множества означают, что в соответствующем поле может содержаться не одно, а сразу несколько значений (vaiuel, vaiue2 и т. д., т. е. множество значений). Формат задания данных такого типа имеет следующий вид:

|  |
| --- |
| SET(value1,value2,value3,...) |

**Модификаторы и флаги типов**

|  |  |
| --- | --- |
| Модификатор | Описание |
| not NULL | Означает, что поле не может содержать неопределенное значение —  в частности, поле обязательно должно быть инициализировано при  вставке новой записи в таблицу (если не задано значение по  умолчанию) |
| primary key | Отражает, что поле является первичным ключом, т. е.  идентификатором записи, на который можно ссылаться |
| auto\_INCREMENT | При вставке новой записи поле получит уникальное значение, так что  в таблице никогда не будут существовать два поля с одинаковыми  номерами. (Мы поговорим об этом чуть позже.) |
| DEFAULT 'значение' | Задает значение по умолчанию для поля, которое будет  использовано, если при вставке записи поле не было проинициализировано явно |

* + 1. Удаление таблицы

|  |
| --- |
| DROP TABLE ИмяТаблицы |

Удаляет таблицу имятаблицы. Таблица не обязательно должна быть пустой, так что будьте внимательны, чтобы случайно не "аннулировать" нужную таблицу с данными.

* + 1. Вставка записи в таблицу

|  |
| --- |
| INSERT INTO ИмяТаблицы (ИмяПоля1, Имя11оля2, ...) VALUES ( 'зн1', 'зн2', . . .) |

Добавляет в таблицу ИмяТаблицы запись, у которой поля, обозначенные как имяполяы, установлены в значения соответственно знN. Те поля, которые в этой команде не перечислены, получают "неопределенные" значения.

* + 1. Удаление записи

|  |
| --- |
| DELETE FROM ИмяТаблицы WHERE выражение |

Удаляет из таблицы ИмяТаблицы все записи, для которых выполнено выражение. Параметр выражение — это просто логическое выражение, составленное "почти" по правилам РНР.

* + 1. Поиск и выборка записей

|  |
| --- |
| SELECT \* FROM ИмяТаблицы WHERE выражение [ORDER BY ИмяПоля [DESC] , . . . ] |

Команда select — одна из самых сложных в SQL. Существует огромное количество ее разновидностей, предназначенных как для выборки нескольких строк из одной таблицы, так и для выполнения сложных вычислений и группировки строк сразу из нескольких таблиц.

* + 1. Получение числа записей, удовлетворяющих выражению

|  |
| --- |
| SELECT COUNT (\*.) FROM ИмяТаблицы WHERE выражение |

Результатом этого запроса будет набор данных, состоящий из единственного целого числа — количества элементов в таблице ИмяТаблицы.

* + 1. Получение уникальных значений столбцов

|  |
| --- |
| SELECT DISTINCT country FROM ИмяТаблицы WHERE age>=30 |

Данный запрос сгенерирует результат, состоящий из одного столбца, в котором и будут перечислены искомые страны.

* + 1. Обновление записей

|  |
| --- |
| UPDATE ИмяТаблицы SET (ИмяПоля1= 'зн1', ИмяПоля1= 'зн2', . . .) WHERE выражение |

В таблице ИмяТаблицы для всех записей, удовлетворяющих выражению выражение, указанные поля устанавливаются в соответствующие значения. При этом остальные поля остаются без изменения! Эта команда часто выполняется, если не требуется обновлять сразу все поля какой-то записи, а нужно затронуть только некоторые.

* 1. Индексы

Сервер MySQL для каждой своей таблицы строит своеобразный "предметный указатель", который в терминах баз данных называют индексом. С помощью индекса MySQL сразу же находит положение записи в своих служебных файлах, если известно ее значение. Остается лишь считать данные из нужного файла (по аналогии с тем, как вы в конце открываете нужную страницу книги).

Для каждой таблицы может существовать не один, а сразу несколько индексов. Например, индекс автоматически создается для поля, которое помечено как primary key (в нашем случае id имеет тип auto\_increment primary key), так что выборка по id происходит мгновенно даже в таблице, содержащей десятки миллионов записей. При создании таблицы вы можете указать еще индексы, которые будут ускорять выборку по другим полям. Например, создав индекс по полю name в таблице people, вы заставите запросы вроде name= "Marcus chong" срабатывать практически мгновенно.

Можно создавать и так называемые составные индексы, которые охватывают сразу совокупность полей в таблице. Представим, что в таблице people у нас есть еще два поля: firstname (имя человека, например, "Вася") и age (его возраст). Добавив индекс по паре (firstname, age), вы сможете быстро извлекать записи запросами, в которых встречается выражение вроде "firstname=? and age=?".

* + 1. Создание индексов

Чтобы добавить индекс при создании новой таблицы, команду create можно записать так:

|  |
| --- |
| CREATE TABLE people (  id INT AUTO\_INCREMENT PRIMARY KEY,  firstname VARCHAR(20),  INDEX i\_firstname(firstname(10)),  name VARCHAR(40) ,  INDEX i\_name (name (10) ),  age INT,  INDEX i\_age(age)  ) |

* + 1. План выполнения запроса

Длина индекса влияет лишь на скорость выборки, но не на ее результат. Вообще, индексы — это способ повышения производительности сервера СУБД, на результирующий набор данных они никак не влияют.

При выполнении запроса MySQL самостоятельно решает, какие индексы и в какой момент оптимальнее всего использовать. Встроенный оптимизатор запросов работает очень хорошо, и нам на своей практике еще ни разу не приходилось встречать случаев, когда бы он ошибся. Посмотреть, в каком порядке извлекаются данные и какие индексы при этом используются, можно при помощи команды explain, например:

|  |
| --- |
| EXPLAIN SELECT \* FROM people WHERE age>20 AND firstname LIKE 'Bac%' |

Вместо того чтобы выполнять запрос select, MySQL теперь просто проанализирует его и вернет результирующий набор данных, хранящий информацию о предполагаемом плане выполнения.

* + 1. Недостатки индексов

У индексов имеется два недостатка:

- Наличие индексов увеличивает требование к дисковому пространству, занимаемому базой данных. Обычной практикой можно считать удвоение объема, занимаемого таблицей, при добавлении индекса по каждому ее полю. (Представьте, во что бы превратился предметный указатель этой книги, если бы в нем стояли ссылки на все слова, которые в ней встречаются.)

- Индексы несколько замедляют вставку и изменение записей в таблице. (Если бы текст данной книги часто изменялся, редактор вынужден был бы каждый раз модифицировать ее предметный указатель — это очень трудоемкая работа.)

Индексы следует использовать для тех полей таблицы, которые участвуют в where-предикатах (логических выражениях). Например, не имеет смысла делать индекс по текстовому полю гостевой книги — все равно оно ни с чем не сравнивается при выборке, а значит, не влияет и на результирующий набор данных. Если таблица обновляется очень часто, а извлечения из нее практически не происходят (например, таблица, хранящая все HTTP-запросы к текущему серверу), подумайте о том, чтобы вообще отказаться от индексов — это увеличит производительность и уменьшит занимаемый на диске объем базы данных.

1. **СЕССИИ**

Сессия — это некоторое место долговременной памяти (обычно часть на жестком диске и часть — в cookies браузера), которое сохраняет свое состояние между вызовами сценариев одним и тем же пользователем. Поместив в сессию переменную (любой структуры), мы при следующем запуске сценария получим ее в целости и сохранности.

* 1. Механизм работы

Для начала должен существовать механизм, который бы позволил РНР идентифицировать каждого пользователя, запустившего сценарий. То есть при следующем запуске РНР нужно однозначно определить, кто его запустил: тот же человек, или другой. Делается это путем присвоения клиенту так называемого уникального идентификатора сессии, Session ID. Чтобы этот идентификатор был доступен при каждом запуске сценария, РНР помещает его в cookies браузера.

Теперь, зная идентификатор (дальше для краткости мы будем называть его SID), РНР может определить, в каком же файле на диске хранятся данные пользователя.

Немного о том, как сохранять данные в сессии. Для этого существует глобальный массив $\_session, который РНР обрабатывает особым образом. Поместив в него некоторые данные, мы можем быть уверены, что при следующем запуске сценария тем же пользователем массив $\_session получит то же самое значение, которое было у него при предыдущем завершении программы. Это произойдет потому, что при завершении сценария РНР автоматически сохраняет массив $\_session во временном хранилище, имя которого хранится в SID.

* 1. Инициализация сессии

Прежде, чем работать с сессией, ее необходимо инициализировать. Делается это путем вызова специальной функции session\_start().

Эта функция инициализирует механизм сессий для текущего пользователя, запустившего сценарий. По ходу инициализации она выполняет ряд действий:

- если посетитель запускает программу впервые, у него устанавливается cookies с уникальным идентификатором, и создается временное хранилище, ассоциированное с этим идентификатором;

- определяется, какое хранилище связано с текущим идентификатором пользователя;

- если в хранилище имеются какие-то данные, они помещаются в массив

$\_SESSION;

- если параметр register\_globals из файла php.ini равен on, то все ключи в массиве $\_ SESSION и соответствующие им значения превращаются в глобальные переменные.

* 1. Уничтожение сессии

|  |
| --- |
| bool session\_destroy() |

Данная функция уничтожает хранилище сессии. При этом массив $\_session не очищается. Чтобы полностью удалить сессию, нужно выполнить следующую последовательность команд:

|  |
| --- |
| // Очистить данные сессии для текущего сценария.  $\_SESSION = array();  // Удалить cookie, соответствующую SID.  @unset($\_COOKIE[session\_name()])/  // Уничтожить хранилище сессии.  session\_destroy(); |

Очистка хранилища сессии полезна тем, что начиная с этого момента все страницы, использующие то же самое хранилище, получат пустую сессию.

1. **РАБОТА С ИЗОБРАЖЕНИЯМИ**
   1. Функция getimagesizef)

list getimagesize(string $ filename)

Эта функция предназначена для быстрого определения в сценарии размеров (в пикселах) и формата рисунка, имя файла которого ей передано. Она возвращает список из следующих элементов.

- Нулевой элемент (с ключом 0) хранит ширину картинки в пикселах.

- Первый элемент (с ключом 1) содержит высоту изображения.

- Ячейка массива с ключом 2 определяется форматом изображения: 1=GIF,

2 = JPG, 3 = PNG, 4 = SWF, 5 = PSD, 6 = BMP, 7 = TIFF (на Intel-процессорах), 8 = TIFF (на процессорах Motorola), 9 = JPC, 10 = JP2, ll=JPX, 12 = JB2, 13 = SWC, 14 = IFF, 15 = WBMP, 16 = XBM. Можно также использовать и константы вида imagetypexxx, встроенные в РНР, где ххх соответствует названию формата (только что мы перечислили все поддерживаемые форматы).

- Элемент, имеющий ключ 3, содержит строку примерно следующего вида: "height=sx width=sy", где sx и sy — соответственно ширина и высота изображения. Указанный элемент задумывался для того, чтобы облегчить вставку данных о размере изображения в тег <img>, который может быть сгенерирован сценарием.

- Ячейка с индексом 4 содержит число битов, используемых для хранения информации о каждом пикселе изображения.

- Элемент с ключом 5 содержит количество цветовых каналов, задействованных в изображении. Для JPEG-картинок в формате RGB он будет равен 3, а в формате CMYK - 4.

- Элемент со строковым ключом mime хранит MIME-тип изображения (например, image/gif, image/jpeg И Т. Д.). Его очень удобно использовать при выводе заголовка content-type, определяющего тип изображения.

В случае ошибки функция возвращает false и генерирует предупреждение.

* 1. Пример создания изображения

|  |
| --- |
| <?php ## Создание картинки "на лету".  // Получаем строку, которую нам передали в параметрах  $string = $\_SERVER['QUERY\_STRING'];  if (!$string) $string = "Hello, world!";  // Загружаем рисунок фона с диска.  $im = imageCreateFromGif("button.gif");  // Создаем в палитре новый цвет — черный.  $color = imageColorAllocate($im, 0, 0, 0) ;  // Вычисляем размеры текста, который будет выведен.  $рх = (imageSX($im)-6.5\*strlen($string))/2;  // Выводим строку поверх того, что было в загруженном изображении.  imageString($im, 3, $рх, 1, $string, $color);  // Сообщаем о том, что далее следует рисунок PNG.  Header("Content-type: image/png");  // Теперь - самое главное: отправляем данные картинки в  // стандартный выходной поток, т. е. в браузер.  imagePng($im);  //В конце освобождаем память, занятую картинкой.  imageDestroy($im);  ?> |

* 1. Загрузка изображения

|  |
| --- |
| resource imageCreateFromPng (string $ filename)  resource imageCreateFromJpeg (string $ filename)  resource imageCreateFromGif(string $filename) |

Эти функции загружают изображения из файла в память и возвращают его идентификатор. Как и после вызова функции imageCreate (), дальнейшая работа с картинкой возможна только через этот идентификатор. При загрузке с диска изображение распаковывается и хранится в памяти уже в неупакованном формате, для того чтобы можно было максимально быстро производить с ним различные операции, например, масштабирование, рисование линий и т. д. При сохранении на диск или выводе В браузер функцией imagePngO (ИЛИ, соответственно, imageJpegO И imageGif()) картинка автоматически упаковывается.

* 1. Определение параметров изображения

|  |
| --- |
| int imageSX(int $im) |

Функция возвращает горизонтальный размер изображения, заданного своим идентификатором, в пикселах.

|  |
| --- |
| int imageSY(int $im) |

Возвращает высоту картинки в пикселах.

|  |
| --- |
| int imageColorsTotal (int $im) |

Эту функцию имеет смысл применять только в случае работы с изображениями, "привязанными" к конкретной палитре — например, с файлами GIF или PNG. Она возвращает текущее количество цветов в палитре.

* 1. Сохранение изображения

|  |
| --- |
| int imagePng (resource $im [, string $ filename])  int imageJpeg(resource $im [,string $filename] [,int $ quality])  int imaged f (resource $im [, string $ filename]) |

Перечисленные функции сохраняют изображение, заданное своим идентификатором и находящееся в памяти, на диск, или же выводят его в браузер. Разумеется, вначале изображение должно быть загружено или создано при помощи функции imageCreate() (ИЛИ imageCreateTrueColor ()), т. е. нужно знать его идентификатор $im.

Если аргумент $ filename опущен (или равен пустой строке "» или null), to сжатые данные в соответствующем формате отправляются прямо в стандартный выходной поток, т. е. в браузер.

1. **ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА PHP**

Ключевым понятием ООП является класс. Класс можно рассматривать как тип некоторой переменной.

Переменная класса (далее объект класса) обычно имеет набор свойств (значений различных типов) и операций (или методов, функций), которые могут быть с ним проведены. Свойства и методы класса часто называют его членами.

Так же, как может существовать много переменных одного и того же типа (например, строкового), не связанных между собой, возможно и наличие в программе множества объектов одного и того же класса, различающихся своими свойствами.

* 1. Создание класса

Новый класс (тип данных) в программе описывается при помощи ключевого слова class. Внутри класса могут располагаться его свойства (переменные класса) и методы (функции-члены класса).

Пример:

|  |
| --- |
| <?php ## Пример класса,  class Math\_Complex {  // Свойства: действительная и мнимая части,  public $re, $im;  // Метод: добавить число к текущему значению. Число задаете;  // своей действительной и мнимой частью,  function add($re, $im) {  $this->re += $re;  $this->im += $im;  }  ?> |

* 1. Создание объекта класса

При создании переменных, имеющих пользовательский тип данных (иными словами, при создании объектов класса), применяется ключевое слово new, за которым следует имя класса:

|  |
| --- |
| $obj = new Math\_Complex |

Теперь $obj хранит все данные класса — в частности, содержит внутри себя отдельные значения $ге и $im.

* 1. Доступ к свойствам объекта

Каждый объект имеет свой собственный набор ассоциированных с ним свойств (значений, или переменных) и множество методов (функций-членов). Каждое свойство объекта доступно в программе по его имени. Можно присваивать значение свойству или получать его величину:

|  |
| --- |
| // Создаем новый объект класса Math\_Complex.  $obj = new Math\_Complex;  // Присваивает значение свойствам $ге и $im объекта $obj.  $obj->re = 6;  $obj->im - 101;  // Выводит значение свойства re объекта $obj.  echo $obj->re; |

Доступ к свойству осуществляется при помощи оператора -> (стрелка, символ -, за которым идет >).

* 1. Доступ к методам

|  |
| --- |
| <?php ## Вызов метода объекта.  // Подключение каталога библиотек в includejpath.  require\_once "lib/config.php";  // Загрузка класса.  require\_once "Math/Complex.php";  // Создаем новый объект класса Math\_Complex.  $obj = new Math\_Complex;  // Присваиваем начальное значение свойствам.  $obj->re = 16.7;  $obj->im = 101;  // Вызов метода add()c параметрами (18.09, 303) объекта $obj.  $obj->add(18.09/ 303);  // Выводим результат:  echo "({$obj->re}, {$obj->im})";  ?> |

Первым делом создается локальная переменная $this, которой присваивается то же значение, что было у $obj. To есть, в $this теперь хранится ссылка на объект, для которого вызывается метод. Далее РНР смотрит, какому классу принадлежит $obj (в нашем случае это Math\_Complex) И находит функцию-член: Math\_Complex: : add (). Функция вызывается, при этом $this, напомним, равен $obj. В итоге add о изменяет значения $obj->re И $obj->im (которые для нее выглядит Как $this->re И $this->im;). Их мы распечатываем следующей строчкой программы, уже после выхода из функции.

Вызов метода некоторого объекта автоматически предоставляет ему доступ к свойствам этого объекта посредством специальной переменной $this. При этом $this не нужно нигде объявлять явно, она появляется сама собой. Данная техника — ключевая особенность ООП.

* 1. Конструктор

|  |
| --- |
| <?php ## Пример класса с конструктором,  class Math\_Complex2 {  public $re, $im;  // Инициализация нового объекта,  function construct($re, $im) {  $this->re = $re;  $this->im ■ $im;  }  // Добавляет к текущему комплексному числу другое,  function add(Math\_Complex2 $y) {  $this->re += $y->re;  $this->im += $y->im;  }  // Преобразует число в строку (например, для вывода).  function toStringO {  return "({$this->re}, {$this->im})";  }  }  ?> |

Метод construct - это так называемый конструктор класса. Он вызывается всякий раз, когда используется оператор new для объекта.

* 1. Деструктор

Деструктор — специальный метод объекта, который вызывается при уничтожении этого объекта (например, после завершения программы). Деструкторы обычно выполняют служебную работу — закрывают файлы, записывают протоколы работы, разрывают соединения, "форматируют винчестер" — в общем, освобождают ресурсы.

|  |
| --- |
| <?php ## Деструкторы.  // Класс, упрощающий ведение разного рода журналов.  class File\_Logger {  public $f; // открытый файл  public $name; // имя журнала  public $lines = array(); // накапливаемые строки  public $t;  // Создает новый файл журнала или открывает дозапись в конец  // существующего. Параметр $name — логическое имя журнала.  public function construct($name, $fname) {  $this->name = $name;  $this->f = fopen($fname, "a+");  $this->log("### construct() called!");  }  // Гарантированно вызывается при уничтожении объекта.  // Закрывает файл журнала.  public function destructO {  $this->log("### destructO called!");  // Вначале выводим все накопленные данные.  fputs($this->f, join("", $this->lines));  // Затем закрываем файл.  fclose($this->f);  }  // Добавляет в журнал одну строку. Она не попадает в файл сразу же,  // а записывается в буфер и остается там до вызова destructO .  public function log($str) {  // Каждая строка предваряется текущей датой и именем журнала.  $prefix = "[".date("Y-m-d\_h:i:s ")."{$this->name}] ";  $str = preg\_replace('/A/m', $prefix, rtrim($str));  // Сохраняем строку.  $this->lines[] = $str."\n";  }  }  ?> |

* 1. Права доступа к членам класса
     1. Модификаторы доступа

Public – открытый доступ

Члены класса, помеченные ключевым словом public ("публичный", "открытый"), доступны для использования извне класса (например, из вызывающей программы). Пример:

|  |
| --- |
| class Hotel {  public $exit;  public function escape() { |
| echo "Let's go through the {$this->exit}!";  }  }  $theLafayette = new Hotel();  $theLafayette->exit = "main wet wall"; // допустимо  $theLafayette->escape(); // допустимо |

Private – доступ только из методов класса

С использованием ключевого слова private ("личный", "закрытый") можно сделать члены касса "невидимыми" из вызывающей программы, будто бы их и нет. В то же время, методы "своего" класса могут обращаться к ним без всякого ограничения.

Пример:

|  |
| --- |
| class Hotel {  private Sexit;  public function escape() {  $this->findWayOut(); // допустимо  echo "Let's go through the {$this->exit}!"; // допустимо  }  public function lock() { $this->exit = null; }  private function findWayOut() {  $this->exit = "main wet wall"; //допустимо  }  $theLafayette = new Hotel();  $theLafayette->findWayOut(); // Ошибка! Доступ закрыт!  $theLafayette->escape(); // допустимо  $theLafayette->exit = "hotel doors"; // Ошибка! Доступ закрыт! |

Как видно, модификатор private включает максимально возможные ограничения на доступ к членам класса. Он разрабатывался специально для того, чтобы запретить прямое изменение свойств объекта, а также доступ к различным служебным методам.

Protected – доступ из методов производного класса

Модификатор protected ("защищенный") с точки зрения вызывающей программы выглядит точно так же, как и private: он запрещает доступ к членам объекта извне.

Однако по сравнению с private он более "либерален", ибо позволяет обращаться к членам не только из "своих" методов, но также и из методов производных классов (если используется наследование). О наследовании мы подробно поговорим в следующей главе, а пока только скажем, что "защищенными" обычно делают лишь методы, но не свойства классов. Это позволяет создавать "полуслужебные" функции, которые, с одной стороны, выполняют низкоуровневые действия и не должны быть "видны" в основной программе, а с другой, могут использоваться в классах-потомках.

* + 1. Статические члены класса

При использовании классов и объектно-ориентированного подхода также можно создавать статические члены класса. Такие члены существуют в единственном экземпляре и разделяются всеми экземплярами (объектами) одного и того же класса. В этом и заключается весь смысл их существования.

Статическое свойство класса недоступно через обращение $this->property или $obj->property. Вместо этого следует использовать немного необычный синтаксис: ИмяКласса: :$property (или, при вызове из метода класса ИмяКласса, self: : $property).

Статический метод во время своего запуска не получает ссылку $this, поэтому он может работать только со статическими членами (свойствами и другими методами) своего класса.

Пример: счетчик объектов

Пример класса, который "считает", сколько его экземпляров (объектов) существует в текущий момент, и позволяет получить эту информацию из вызывающей программы

|  |
| --- |
| <?php ## Использование статических членов класса,  class Counter {  // Скрытый статический член класса — общий для всех объектов,  private static $count = 0;  // Конструктор увеличивает счетчик на 1. Обратите внимание  //на синтаксис доступа к статическим переменным класса!  public function construct() { self::$count++; }  // Деструктор же — уменьшает.  public function destructO { self::$count—; }  // Статическая функция, возвращает счетчик объектов,  public static function getCountO { return self::$count; }  // Как видите, установить счетчик в произвольное значение  // извне нельзя, можно только получить его значение. Вот он,  // модификатор private в действии .  )  // Создаем б объектов.  for ($objs=array(), $i=0; $i<6; $i++)  $objs[] = new Counter();  // Статические функции можно вызывать точно так же, как будто бы это  // обычный метод объекта. При этом $this все равно не передается,  //он просто игнорируется.  echo "Сейчас существует {$objs[0]->getCount()} объектов.<br>";  // Удаляем один объект.  $objs[5] = null;  // Счетчик объектов уменьшится!  echo "А теперь — {$objs[0]->getCount()} объектов.<br>";  // Удаляем все объекты.  $objs = array();  // Другой способ вызова статического метода — с указанием класса.  // Это очень похоже на вызов функции из библиотеки,  echo "Под конец осталось — ".Counter:igetCount()." объектов.<br>";  ?> |

* 1. Клонирование объектов

Для получения дубликата некоторого объекта применяется ключевое слово clone.

Пример:

|  |
| --- |
| <?php ## Встроенное клонирование объектов.  require\_once "lib/config.php";  require\_once "Math/Complex2.php";  $a = new Math\_Complex2(314, 101);  $x = new Math\_Complex2(0, 0);  // Создаем КОПИЮ объекта $х.  $y = clone $x;  // Теперь $х и $у полностью различны.  $y->add($a) ;  // При этом $х не изменяется!  echo их=", $х, ", у=", $у;  // Попробуйте убрать clone — вы увидите, что $х и $у имеют  // одинаковые значения, ибо ссылаются на один и тот же объект. |

* + 1. Запрет клонирования

Одна из полезных особенностей определения собственного метода clone о заключается в том, что его можно объявить закрытым (private). В этом случае в программе нельзя будет создать копию объекта никакими способами. В некоторых ситуациях это может оказаться полезным — существуют объекты, для которых операция клонирования бессмысленна, и ее нужно запретить. (К таким сущностям относятся объекты, существующие в программе в единственном экземпляре.)