- Построение системы автоматической проверки решений задач по информатике частично реализовано с использованием открытых кодов ejudge, специально разработанных для этих целей совместно со специалистами ВМиК МГУ, ИСП РАН. CMS MOODLE была адаптирована для использования с системой Ejudge программистами проекта с использованием фреймворка symfony на основе Model-View-Controller паттерна. Для работы проверяющей системы установлены эмулятор DOS dosemu, компиляторы поддерживаемых языков программирования (Borland Pascal, Borland C/C++, Free Pascal, GNU C/C++/Pascal/Java, Borland Kylix). Отчеты, генерируемые системой, представляются в виде XML-документа и могут быть экспортированы в формат CSV.
- Создание утилит, облегчающих ввод данных в систему. В частности, утилиты конвертирования LaTeX файлов к виду, удобному для представления в системе MOODLE.

Дальнейшее развитие проекта направлено как на повышение объема и качества представленных учебных материалов, так и на решение обнаруженных технических проблем и ограничений:

- Интенсификация ввода материалов, которая предполагает как самостоятельное создание утилит, облегчающих и автоматизирующих ввод данных, так и тесное взаимодействие с сообществом MOODLE для совместного создания и обмена полезными решениями.
- Более тесная интеграция модулей проекта между собой.
- Интеграция проекта со сторонними образовательными ресурсами.
- Повышение популярности и, как следствие, масштабируемости проекта.
- Тесная обратная связь с целевой аудиторией (школьники старших классов, преподаватели), способствующая повышению качества ресурса.

УДК 004.422.8
© Маматов А.В., Немцев А.Н., Штифанов А.И., Загороднюк Р.А., Беленко В.А., Немцев С.Н., 2008
(Белгородский государственный университет)

РАЗРАБОТКА НОВЫХ БЛОКОВ ДЛЯ СИСТЕМЫ MOODLE

Одним из значительных преимуществ системы Moodle по сравнению с другими системами электронного обучения является открытость кода и возможность расширения функциональности системы путем создания новых блоков, модулей, элементов и т.п. Фактически, используя функции ядра системы и её многочисленные библиотеки, можно создавать блоки для решения любых задач, связанных с учебным процессом и с деятельностью образовательных учреждений.

Авторы имеют опыт разработки дополнительных блоков для системы Moodle и в данной статье излагают основные моменты, связанные с программированием новых блоков.

Прежде чем рассмотреть структуру блока, рассмотрим структуру типового скрипта системы на примере admin/admin.php. Данный скрипт предназначен только для главного администратора системы и используется для назначения других администраторов. Содержание данного скрипта условно разобьем на разделы.

Первый раздел скрипта содержит список подключаемых файлов. У библиотечных файлов и файлов описания классов данный раздел отсутствует. Все остальные скрипты системы должны начинаться со строки подключения файла config.php, расположенного в корневом каталоге системы:

require_once(«../config.php»).

Понятно, что количество указаний на папки верхнего уровня – ../ – зависит от местоположения скрипта.

Назначение файла config.php состоит не только в определении основных конфигурационных параметров системы, связанных с доступом к базе данных и наименованием папок, но и в подключении всего ядра системы. То есть если отследить процесс включения файлов с использованием функций require_once и include, то вырисовывается такая последовательность (для версии 1.5):

config.php->setup.php->(adodb.inc.php; weblib.php; datalib.php; moodlelib.php).

После подключения файла config.php можно подключать другие файлы, необходимые для работы скрипта. Так, например, для использования на странице так называемых «гибких» таблиц (сортируемых по любой колонке) необходимо подключить файл, содержащий описание класса flexible_table:

require_once($CFG->libdir.'/tablelib.php').

Следующий раздел скрипта содержит описание констант:

define(«MAX_USERS_PER_PAGE», 50).

Этот раздел является необязательным, и его наличие зависит от необходимости введения констант в конкретном скрипте.

Третий раздел связан с получением значений переменных, переданных методом GET или POST. Для этого необходимо использовать функции optional_param() и required_param(). Формат описания этих функций следующий:

optional_param ($varname, $default=NULL, $options=PARAM_CLEAN);

required_param($varname, $options=PARAM_CLEAN).

Первым параметром является имя переменной, передаваемой скрипту методом GET или POST. Последний параметр определяет тип переменной. По умолчанию, последний параметр равен значению PARAM_CLEAN. Это означает, что значение переменной будет очищено с помощью функции clean_text, выполняющей операцию удаления опасных тегов, могущих повредить работу скрипта. Если необходимо конвертировать переменную в определенный формат данных, то можно использовать следующие константы: PARAM_INT, PARAM_ALPHA, PARAM_BOOL. Для очистки от HTML-тегов используется константа PARAM_NOTAGS. Для проверки правильности записи URL – PARAM_URL. Для очистки от символов, недопустимых в названиях папок и файлов, – PARAM_SAFEDIR, PARAM_FILE, PARAM_PATH.

Отличие двух вышеприведенных функций состоит в том, что функция required_param ищет в списке переданных переменных ту, которая указана в её первом параметре $varname. Если переменная не найдена, т.е. она не была передана ни методом GET, ни методом POST, тогда работа скрипта прекращается и выводится сообщение о том, что требуемый параметр отсутствует. Функция optional_param также ищет в списке переданных переменных ту, которая указана в её первом параметре $varname, но в случае неудачного поиска не прекращает работу скрипта, а инициализирует переменную значением, указанным во втором параметре $default.

Следующий раздел (по введенной нами нумерации – четвертый) связан с проверкой различных параметров и переменных. Прежде всего, мы проверяем существование переменной $site, хранящей основные настройки стартовой страницы сайта:

if (!$site = get_site()) { redirect(«$CFG->wwwroot/$CFG->admin/index.php»);}

Если переменная $site не может быть проинициализирована с помощью функции get_site(), то происходит выход из скрипта и переход на стартовую страницу управления системой.

Следующая проверка выполняет контроль корректного входа пользователя в систему:

require_login().

Если пользователь не «залогинился», то функция переадресует пользователя на страницу входа на сайт для ввода логина и пароля.

Далее мы должны проверить роль пользователя. Если наш скрипт предназначен только для администраторов, то следующая проверка отсечет всех пользователей, не имеющих прав администратора.

if (!isadmin()) { error(«You must be an administrator.»); }

Для проверки роли учителя можно воспользоваться функцией isteacherinanycourse(), роль учащегося проверяется функцией isstudent ($courseid), где $courseid – идентификатор курса, в котором зарегистрирован учащийся.

В пятом разделе скрипта определяются строковые константы, которые будут востребованы неоднократно в течение работы всего скрипта. Как правило, используется следующий способ определения строковых констант:

$strgroups = get_string('groups');

$struser = get_string(«user»).

Далее используется функция print_header для вывода титульной части страницы. Формат функции следующий:

print_header ($title='', $heading='', $navigation='', $focus='', $meta='', $cache=true, $button=' ', $menu='', $usexml=false, $bodytags=''),

где $title – заголовок страницы, отображаемый в заголовке окна браузера; $heading – заголовок, отображаемый в начале страницы; $navigation – строки навигационных ссылок («ссылки-цепочки»); $focus – указывает элемент, который получит фокус ввода, например inputform.password; $meta – ме-

татеги, добавляемые в область <HEAD>; $cache – определяет «кешируемость» страницы, $button – HTML-код кнопки (обычно для режима редактирования); $menu – HTML код локального меню; $usexml – использовать ли XML для данной страницы; $bodytags – данный текст будет включен в тег <body>.

Наиболее часто в функции используются только три первых параметра, например

print_header(«$site->shortname: $struser», «$site->fullname», « $site->shortname -> $strusers -> $struser»);

После вывода заголовка заканчивается работа стандартных разделов, присутствующих практически в каждом скрипте, и начинается работа по определенному алгоритму, обусловленному назначением скрипта.

Теперь перейдем к рассмотрению определения блока в системе Moodle. В самом общем случае, для определения блока достаточно только одного файла с исходным кодом на PHP. Данный файл должен начинаться со слова block_, и в нем должен быть описан одноименный класс. Например, файл block_simplehtml.php должен содержать описание класса block_simplehtml:

```
<?php
class block_simplehtml extends block_base {
   function init() {  $this->title = get_string('simplehtml', 'block_simplehtml');
                      $this->version = 2007111100;  }
}
?>
```

В классе блока должен обязательно присутствовать метод init. Его цель состоит в том, чтобы присвоить начальные значения двум переменным класса, перечисленным в нем:

1) $this->title – хранит название, которое будет отображаться в заголовке блока.

2) $this->version – версия блока.

Чтобы блок отображал что-то на странице, необходимо добавить еще один метод в класс блока:

```
function get_content() {
   if ($this->content !== NULL) { return $this->content; }
   $this->content = new stdClass;
   $this->content->text = get_string('text', 'block_simplehtml');
   $this->content->footer = get_string('footer', 'block_simplehtml');
   return $this->content;
}
```

В начале данного метода организуется проверка на наличие контента у блока. Контент блока хранится в поле content. Если оно не является НУЛЕ-ВЫМ, то мы сразу выходим из данного метода, тем самым, экономя массу времени на повторное формирование контента. Если содержание блока не было ещё сформировано, тогда необходимо заполнить два поля: text и footer.

Для проверки работы блока необходимо сохранить файл block_simplehtml.php в папке moodle/blocks/simplehtml и создать одноименный файл в папке lang/ru, содержащий следующие строковые константы:

```
$string['simplehtml'] = 'Простой HTML-код';
$string['text'] = 'Содержание блока!';
$string['footer'] = 'Нижний колонтитул'.
```

Регистрация блока в системе осуществляется на странице администрирования.

В рамках данной статьи авторами обозначены начальные вопросы программирования дополнительных блоков и скриптов для системы Moodle. Затронутая тема очень обширна и включает в себя вопросы использования базы данных, построение интерфейса, более тонкую настройку блоков, разработку новых классов и др.

Библиографический список

1. MOODLE Developer documentation.
   URL: http://docs.moodle.org/en/Developer_documentation
2. Основы администрирования системы дистанционного обучения «Пегас» : учеб. пособие / А.И. Штифанов, В.А. Беленко, А.Н. Немцев, А.В. Маматов, А.Г. Клепикова. – Белгород : Изд-во БелГУ, 2007. – 127 с.