Глава 2

Создание модулей

Многие приложения с открытым кодом допускают их настройку в соответствии с требованиями пользователя за счет модификации исходного кода. Хотя таким образом и можно добиться желаемого, обычно такой подход осуждается и рассматривается в сообществе Drupal как последнее средство. Настройка кода означает, что при каждом обновлении Drupal нужно будет выполнять больше работы. Ведь придется проводить тестирование, чтобы убедиться: модифицированный код функционирует должным образом. Дабы избежать этого, Drupal был с самого начала спроектирован модульным и расширяемым.

Сам по себе Drupal дает очень немного возможностей для создания приложения, и то, что устанавливается в варианте по умолчанию, часто называют "ядро Drupal". Функциональные возможности ядра наращиваются за счет добавления модулей. Последние — это файлы, которые содержат PHP-код; они размещаются в подкаталоге `sites/all/modules`, создаваемом при установке Drupal. Взгляните теперь на этот каталог и сравните его содержимое со списком модулей, который вы видите, когда перемещаетесь по своему сайту Drupal по пути Administer⇨Site building⇨Modules.

В этой главе мы собираемся создать модуль с нуля. Вы узнаете, как нужно писать код модулей, придерживаясь принятых стандартов. Нам нужна реальная цель, поэтому давайте сосредоточимся на проблеме, часто возникающей в реальном мире, — добавлении аннотаций. Просматривая страницы веб-сайта Drupal, пользователи могут комментировать его содержимое (если администратор установил модуль комментариев). Но как насчет того, чтобы они могли делать аннотации к веб-странице (т.е. примечания, которые может видеть только один пользователь)? Это могло бы быть полезным для конфиденциального обмена мнениями относительно содержимого (мы знаем, что это кажется вам уже знакомым, но потерпите немного).

Создание файлов

Первое, что мы должны сделать, — это выбрать название для модуля. Имя annotate ("аннотировать") кажется подходящим — оно и короткое, и описательное. Затем нам нужно куда-то поместить наш новый модуль. Давайте разместим его в подкаталоге `sites/all/modules`, но так, чтобы он хранился отдельно от модулей ядра. Создадим в каталоге `sites/all/modules` подкаталог и назовем его `annotate`. Мы создаем подкаталог, а не только файл с именем `annotate.module`, потому что мы собираемся включать в него, помимо файла модуля, другие файлы, содержащие описание модуля. Например, нам понадобится файл `README.txt`, дабы объяснить другим пользователям, что именно делает наш модуль и как его можно использовать, и файл `annotate.info`, чтобы предоставить информацию о нашем модуле системе Drupal. Итак, вы готовы начать работу?

Наш файл `annotate.info` таков:

```
; $Id$
name = Annotate
description = Allows users to annotate nodes.
package = Example
version = "$Name$"
```

Формат данного файла `.ini`, это — простой стандарт для файлов конфигурации PHP (см. `http://php. net/parse_ini_file`). Мы начнем с дескриптора идентификации системы управления версиями (CVS) и затем займемся названием и описанием для Drupal, чтобы отобразить их в разделе администрирования модуля на веб-сайте. Модули отображаются группами, принцип группирования определяется пакетом; таким образом, если мы будем иметь три разных модуля, упакованных в пакет `package = Example`, то они будут отображаться в одной группе. Номер версии — это еще один дескриптор идентификации CVS. Если мы хотим использовать наш модуль совместно с другими, регистрируя его в репозитории добавленных модулей Drupal, это значение будет заполнено автоматически.

На заметку Может показаться непонятным, для чего нужен отдельный файл `.info`. Почему бы не ввести в наш основной модуль функцию, которая возвращала бы эти метаданные? Потому что, когда загружается страница управления модулем, пришлось бы загружать каждый отдельный модуль и анализировать, подходит он или нет. Это привело бы к большим потребностям в памяти, возможно, превышающим ее объемы, выделенные на работу PHP в целом. С помощью файлов `.info` информацию можно загрузить быстро и с минимальным использованием памяти.

Теперь мы готовы создать сам модуль. Создайте файл с именем `annotate.module` в подкаталоге `annotate`. Начните файл с открытия дескриптора PHP и дескриптора идентификации CVS, сопровождая свои действия комментариями.

```
<?php
// $Id$
/**
 * @file
 * Предоставляем пользователям возможность приватно добавлять аннотации к узлу.
*
 * Добавляем текстовое поле при отображении узла
 * так, чтобы аутентифицированные пользователи могли делать аннотации.
 */
```

Прежде всего обратите внимание на стиль комментирования. Мы начинаем комментарий с символов `/**`, а в каждой последующей строке используем одну звездочку, смещенную на один пробел вправо (`*`), и символы `*/` в отдельной строке, означающие конец комментария. Лексема `@file` означает, что все, представленное в следующей строке, является описанием того, что делает этот файл. Это однострочное описание используется для того, чтобы `api.module`, экстрактор и форматтер автоматизированной документации Drupal, мог узнать, что делает этот файл. После пустой строки мы добавляем более длинное описание, предназначенное для программистов, которые будут исследовать (и, в этом нет никакого сомнения, совершенствовать) наш код. Обратите внимание на то, что мы преднамеренно не используем заключительный дескриптор (`?>`); эти дескрипторы являются опциональными в PHP и, если их использовать, могут возникнуть проблемы с замыкающим пробельным символом в файлах (см. `http://drupal.org/node/545`).

На заметку Почему мы настолько подробно рассматриваем эту структуру? Потому что, когда сотни людей со всего мира вместе работают над одним проектом, то можно сэкономить много времени за счет того, что все будут делать одни и те же вещи одинаково. Подробности относительно стиля кодирования, требуемого для Drupal, можно найти в разделе "Coding standards" ("Стандарты кодирования") руководства по системе Drupal (`http://drupal.org/node/318`).

Сохраните ваш файл и откройте подкаталог `Administer⇨Site building⇨Modules`. Ваш модуль должен появиться в списке.

Следующий пункт нашей повестки дня — определить некоторые параметры настройки так, чтобы мы могли использовать веб-форму, позволяющую выбирать, какой узел будет аннотироваться. Для этого нужно сделать две вещи. Во-первых, мы должны определить путь, по которому можно обратиться к нашим параметрам настройки. Во-вторых, мы должны создать форму параметров настройки.

Применение обработчиков прерываний

Напоминаем, что Drupal основан на системе обработчиков прерываний, иногда называемых "обратные вызовы". В процессе их выполнения Drupal опрашивает модули на предмет того, не хотели бы они что-нибудь выполнить. Например, при выяснении, какой модуль отвечает за текущий запрос, Drupal просит, чтобы все модули подтвердили пути, которые они поддерживают. Система делает это, создавая список всех модулей и вызывая функцию, которая для каждого модуля имеет имя, состоящее из имени этого модуля, дополненного символами `_menu`. Когда речь идет о модуле `annotate`, он вызывает нашу функцию `annotate_menu()` и передает ей один параметр. Этот параметр указывает, можно или нельзя кэшировать ответ от модуля; обычно элементы меню могут кэшироваться; об исключениях из этого правила мы расскажем в главе 4, в которой описывается система меню или обратного вызова технологии Drupal. Каждый элемент меню представляет собой ассоциативный массив.

Вот что мы теперь добавим к нашему модулю.

```
/**
 * Реализация hook_menu().
 */
function annotate_menu($may_cache) {
  $items = array();
  if ($may_cache) {
    $items[] = array(
      'path' => 'admin/settings/annotate',
      'title' => t('Annotation settings'),
      'description' => t('Change how annotations behave.'),
      'callback' => 'drupal_get_form',
      'callback arguments' => array('annotate_admin_settings'),
      'access' => user_access('administer site configuration')
    );
  }
  return $items;
}
```

Подробности в данном случае нам не очень важны. Этот код просто говорит: "Если пользователь идет по адресу `http://example.com/?q=admin/settings/annotate`, вызовите функцию `drupal_get_form()` и передайте ей идентификатор формы `annotate_admin_settings`". Когда придет время отобразить форму, Drupal попросит, чтобы мы дали ему определение формы (подробнее об этом чуть позже). Когда Drupal закончит опрос всех модулей относительно элементов меню, он будет

иметь меню, в котором можно выбрать функцию, пригодную для запроса пути, который был затребован.

Обратите внимание на то, что любой возвращенный текст, который будет отображен на дисплее пользователя, находится в функции `t()`, названной так потому, что она выполняет перевод строки. Прогоняя весь текст через функцию перевода строки, можно легко выполнить локализацию вашего модуля для различных языков.

На заметку Если вы хотите поближе познакомиться с функцией, которая управляет механизмом обработчика прерываний, рассмотрите функцию `module_invoke_all()` в каталоге `includes/module.inc`.

Теперь вы понимаете, почему мы называем эту функцию `hook_menu()`, или обработчиком событий меню. Обработчики прерываний Drupal всегда создаются с помощью добавления к названию вашего модуля имени обработчика прерываний.

На заметку Технология Drupal развивается быстро. Полный список поддерживаемых обработчиков прерываний и примеров их использования можно найти на сайте Drupal API documentation (документация интерфейсов прикладного программирования Drupal, `http://api.drupal.org`).

Добавление специфичных для модуля параметров настройки

Drupal имеет узлы различных типов, такие как текстовые блоки и страницы. Предположим, мы хотим ограничить использование аннотаций применительно только к узлам некоторых типов. Чтобы добиться этого, мы должны создать страницу, на которой можем сообщить нашему модулю, узлы каких типов мы хотим аннотировать. Добавьте следующий код в модуль `annotate`:

```
/**
 * Определяем форму параметров настройки.
 */
function annotate_admin_settings() {
  $form['annotate_nodetypes'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Users may annotate these node types'),
    '#options' => node_get_types('names'),
    '#default_value' => variable_get('annotate_nodetypes',
      array('story')),
    '#description' => t('A text field will be available on these
      node types to make user-specific notes.'),
  );
  $form['array_filter'] = array('#type' => 'hidden');
  return system_settings_form($form);
}
```

Формы в Drupal представлены в виде древовидной структуры с вложениями, т.е. в виде массива массивов. Это структура предписывает механизму генерации изображений форм, как он должен отображать формы. Для удобочитаемости кода мы помещаем каждый элемент массива в отдельную строку. Каждая директива обозначается символом “решетка” (#) и действует как ключ массива. Вначале объявляется тип элемента формы, в данном случае `heckboxes`; это означает, что с помощью ключевого массива могут быть созданы переключатели на несколько положений. Мы присваиваем элементу формы заголовок, поскольку обычно прогон нашего текста осуществляется с помощью функции `t()`. Затем мы устанавливаем опции для вывода функции `node_get_types('names')`, с помощью которой удобно

возвращать ключевой массив типов узлов, которые доступны в установленной версии Drupal. Вывод функции `node_get_types('names')` мог бы выглядеть так:

```
'page' => 'Page', 'story' => 'Story'
```

Ключи массива — это внутренние имена Drupal для типов узлов с дружественными названиями (теми, которые будут показываться пользователю) в правой части. Если у вас был модуль с рецептом чабера (savory recipe), массив выглядел бы следующим образом:

```
'page' => 'Page', 'savory_recipe' => 'Savory recipe', 'story' => 'Story'
```

Затем в нашей веб-форме Drupal создаст переключатели для узлов типов "текстовый блок" (story) и "страница" (page). Следующая директива, `#default_value`, будет значением по умолчанию для этого элемента формы. Поскольку переключатели — многократно встречающийся элемент формы (т.е. в ней более чем один переключатель), значением для `#default_value` будет некий массив.

Значение для `#default_value` достойно обсуждения:

```
variable_get('annotate_nodetypes', array('story'))
```

Drupal позволяет программистам хранить и извлекать любое значение, используя специальную пару функций — `variable_get()` и `variable_set()`. Эти значения хранятся в таблице базы данных переменных и доступны в любое время при обработке запроса. Поскольку эти переменные извлекаются из базы данных при выполнении каждого запроса, хранить таким образом данные огромных объемов — плохая идея.

Но это — очень удобная система для того, чтобы хранить такие значения, как параметры настройки конфигурации модуля. Обратите внимание: то, что мы передаем переменной `variable_get()`, — это ключ к описанию нашего значения (поэтому мы можем возвратить его) и значение по умолчанию. В данном случае значение по умолчанию — это массив типов узлов, для которых должно быть разрешено аннотирование. Мы собираемся позволить аннотировать по умолчанию узлы типа `story` (блок текста).

Наконец, мы даем описание, чтобы сообщить администратору сайта немного сведений, которые должны войти в это поле. Теперь после перехода по ссылкам Administer⇨Settings⇨Annotate, должна быть показана форма для модуля `annotate.module` (рис. 2.1).

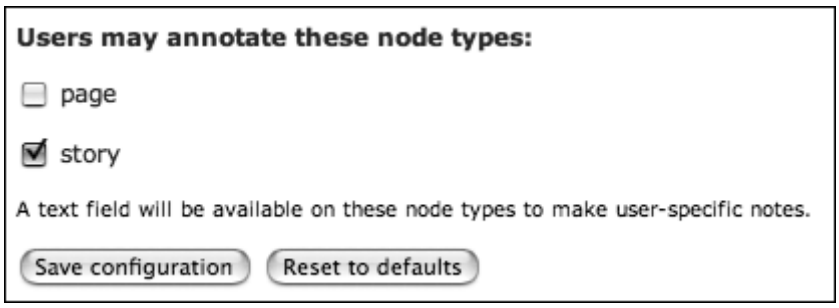

Рис. 2.1. Форма для конфигурирования модуля annotate.module готова

Строка, в которой определяется выражение `$form['array_filter']`, кажется немного загадочной; пока достаточно будет сказать, что она необходима, когда

приходится хранить значения переключателя на несколько положений, используя обработчик прерываний параметров настройки.

Написав лишь несколько строк кода, мы получили функциональную форму конфигурации для нашего модуля, которая автоматически сохранит и будет помнить наши параметры настройки! Хотя одна из строк была довольно длинной, тем не менее вы должны почувствовать, какие возможности открываются перед вами благодаря использованию Drupal.

Добавление формы ввода данных

Для того чтобы пользователь мог ввести примечания, касающиеся веб-страницы, мы должны предусмотреть для них поле ввода. Давайте добавим в форму поле для примечаний:

```
/**
 * Реализация hook_nodeapi().
 */
function annotate_nodeapi(&$node, $op, $teaser, $page) {
  switch ($op) {
    case 'view':
      global $user;
      // Если отображается только резюме узла или если пользователь
      // анонимный (не зарегистрированный), отбросить.
      if ($teaser || $user->uid == 0) {
      break;
      }

      $types_to_annotate = variable_get('annotate_nodetypes', ar-
ray('story'));
      if (!in_array($node->type, $types_to_annotate)) {
        break;
      }

      // Добавляем нашу форму как элемент содержимого.
      $node->content['annotation_form'] = array(
        '#value' => drupal_get_form('annotate_entry_form', $node),
        '#weight' => 10
      );
  }
}
```

Этот код кажется довольно сложным, поэтому давайте рассмотрим его подробнее. Прежде всего обратите внимание на то, что мы ввели еще один обработчик прерываний Drupal. На сей раз это обработчик прерываний `_nodeapi()`, и он вызывается, когда Drupal проводит с узлом различные действия таким образом, чтобы другие модули (как наш) могли изменить узел, прежде чем продолжится обработка. Мы получаем доступ к узлу через переменную `$node`. Амперсанд в первом параметре показывает, что это — фактически *ссылка* на объект `$node`, и это означает: любая модификация, которую мы выполним здесь по отношению к объекту `$node`, сохранится в нашем модуле. Поскольку наша цель состоит в добавлении формы, мы рады, что имеем возможность изменить этот узел.

Нам также дают немного информации о том, что происходит в Drupal в момент вызова нашего кода. Эта информация постоянно находится в переменной `$op`, и ее можно бы вставить (узел создается), удалить (узел удаляется) или принять одно из многих других значений. Сейчас нас интересует только модификация узла в момент, когда он подготавливается к отображению; переменная `$op` в этом случае бу-

дет видима. Мы структурируем наш код, используя оператор выбора, так, чтобы мы могли легко увидеть то, что наш модуль делает в том или ином случае.

Затем мы быстро проверяем случаи, когда мы не хотим отображать поле аннотации. Один случай — это когда параметр `$teaser` принимает значение `TRUE`. Если это так, то узел не отображается отдельно, но может быть отображен в списке, например в результатах поиска. Мы не заинтересованы в добавлении чего-либо в таком случае. Второй вариант — это когда идентификатор пользователя в объекте `$user` равен `0`, это означает, что пользователь не зарегистрировался.

(Обратите внимание на то, что мы использовали глобальное ключевое слово, чтобы переместить объект `$user` в зону видимости.) Мы используем оператор завершения, чтобы выйти из оператора выбора и избегнуть модификации страницы.

Прежде чем мы добавим на веб-страницу форму аннотации, мы должны убедиться в том, что узел, который обрабатывается для визуализации, принадлежит к одному из типов, для которых мы разрешили аннотирование на нашей странице с параметрами настройки. Поэтому мы извлекаем массив типов узлов, который мы сохранили раньше, при реализации обработчика прерываний параметров настройки, и сохраняем его в переменной с легкоузнаваемым именем `$types_to_annotate`. При вызове в качестве второго параметра функции `variable_get()` мы определяем массив по умолчанию, чтобы использовать его в случае, если администратор сайта еще не посетил страницу параметров настройки для нашего модуля, дабы сделать необходимые установки.

На следующем шаге нужно проверить, действительно ли относится узел, с которым мы работаем, к типу, содержавшемуся в переменной `$types_to_annotate`; здесь нас снова выручает оператор завершения, если речь идет о типе узла, который мы не хотим аннотировать.

Наша конечная задача состоит в том, чтобы создать форму и добавить ее к атрибуту содержимого объекта `$node`. Сначала мы должны будем определить форму так, чтобы мы имели возможность что-нибудь добавлять в нее. Мы сделаем это с помощью отдельной функции, единственная задача которой будет состоять в определении формы:

```
/**
 * Определим форму для ввода аннотации.
 */
function annotate_entry_form($node) {
  $form['annotate'] = array(
    '#type' => 'fieldset',
    '#title' => t('Annotations')
  );
  $form['annotate']['nid'] = array(
    '#type' => 'value',
    '#value' => $node->nid
  );
  $form['annotate']['note'] = array(
    '#type' => 'textarea',
    '#title' => t('Notes'),
    '#default_value' => $node->annotation,
    '#description' => t('Make your personal annotations about this content
      here. Only you (and the site administrator) will be able to see
      them.')
  );

  $form['annotate']['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Update')
  );
```

```
  return $form;
}
```

Мы создаем форму тем же самым методом, который использовали в нашей функции `annotate_admin_settings()`, создавая ключевой массив — только на этот раз мы хотим вставить текстовое поле и кнопку Ввод (Submit) в набор полей так, чтобы они группировались на веб-странице. Сначала мы создаем массив типа `'fieldset'` ("набор полей") и даем ему заголовок. Затем мы создаем текстовый массив (`textarea array`). Обратите внимание на то, что ключ массива для текстового массива является членом массива набора полей. Иными словами, мы используем `$form['annotate']['note']` вместо `$form['note']`. Таким образом, Drupal может вывести, что элемент текстового массива (`textarea`) является членом элемента набора полей (`fieldset`). Наконец, мы создаем кнопку Submit и возвращаем массив, который и определяет нашу форму.

Возвращаясь к функции `annotate_nodeapi()`, отметим, что мы добавили форму к содержимому страницы с помощью добавления значения и веса к содержимому узла. Значение определяет, что нужно отображать, а вес подсказывает Drupal, где это нужно отобразить. Мы хотим, чтобы наша форма для аннотации находилась внизу страницы, поэтому мы назначаем для нее относительно тяжелый вес 10. То, что мы хотим отобразить, это наша форма, таким образом мы вызываем функцию `drupal_get_form()`, чтобы изменить нашу форму, из массива, описывающего, как она должна быть построена для завершения HTML-формы. Обратите внимание на то, как мы передаем объект `$node` нашей функции формы; нам нужно это, чтобы получить предыдущую аннотацию и предварительно заполнить ею форму.

Просмотрите страницу в вашем веб-браузере, и вы увидите, что эта форма дополнена формой для аннотаций (рис. 2.2).

Рис. 2.2. Форма для аннотаций, появившаяся на веб-странице Drupal

Что произойдет, если мы щелкнем на кнопке Update (Обновить)? Ничего, потому что мы еще не написали код, позволяющий что-то делать с содержимым формы. Давайте добавим такой код. Но, прежде чем мы это сделаем, нам нужно подумать, где мы собираемся хранить данные, которые вводит пользователь.

Хранение данных в таблице базы данных

Самый общий подход к хранению данных, используемых модулем, состоит в том, чтобы создать отдельную таблицу в базе данных для такого модуля. Благодаря этому данные хранятся отдельно от таблиц ядра Drupal. При решении вопроса, какие поля нужно создавать для вашего модуля, вы должны спросить себя: "Какие данные нужно сохранить? Если я делаю запрос к этой таблице, то что мне понадобится? И наконец, чего я захочу от моего модуля в будущем?"

Данные, которые мы должны хранить, — это просто текст аннотации, численный идентификатор узла, который к ней обращается, и пользовательский идентификатор посетителя, который написал аннотацию. Может также оказаться полезным сохранять временную метку, тогда мы могли бы отображать список недавно обновленных аннотаций, упорядоченных в соответствии с временными метками. Наконец, основной вопрос, который мы зададим этой таблице, таков: "Что будет аннотацией для этого пользователя и для этого узла?" Мы создадим составной индекс на основе полей `uid` и `nid`, чтобы чаще всего получаемые запросы выполнялись с наиболее высокой скоростью. SQL-запрос к нашей таблице будет выглядеть примерно как следующая инструкция:

```
CREATE TABLE annotate (
  uid int NOT NULL default '0',
  nid int NOT NULL default '0',
  note longtext NOT NULL,
  timestamp int NOT NULL default '0',
  PRIMARY KEY (uid, nid),
);
```

Мы могли бы поместить эту инструкцию SQL в файл `README.txt` нашего модуля, и другие пользователи, решившие устанавливать этот модуль, могли бы вручную добавить таблицы базы данных к своим базам данных. Вместо этого мы собираемся воспользоваться преимуществами Drupal для того, чтобы создавать таблицы базы данных в то же самое время, когда активизируется ваш модуль. Мы создадим специальный файл; его имя должно начинаться с имени вашего модуля и заканчиваться суффиксом `.install`, таким образом, для `annotate.module` имя файла было бы `annotate.install`:

```
<?php
  // $Id$

  function annotate_install() {
    drupal_set_message(t('Beginning installation of annotate module.'));
    switch ($GLOBALS['db_type']) {
      case 'mysql':
      case 'mysqli':
        db_query("CREATE TABLE annotations (
          uid int NOT NULL default 0,
          nid int NOT NULL default 0,
          note longtext NOT NULL,
          timestamp int NOT NULL default 0,
          PRIMARY KEY (uid, nid)
          ) /*!40100 DEFAULT CHARACTER SET utf8 */;"
        );
        $success = TRUE;
        break;
      case 'pgsql':
        db_query("CREATE TABLE annotations (
          uid int NOT NULL DEFAULT 0,
          nid int NOT NULL DEFAULT 0,
          note text NOT NULL,
          timestamp int NOT NULL DEFAULT 0,
          PRIMARY KEY (uid, nid)
          );"
        );
        $success = TRUE;
        break;
      default:
        drupal_set_message(t('Unsupported database.'));
    }
    if ($success) {
```

```
        drupal_set_message(t('The module installed tables successfully.'));
    }
    else {
      drupal_set_message(t('The installation of the annotate module was
      unsuccessful.'),'error');
  }
}
```

Этот файл не относится к числу труднопонимаемых. Когда вначале активизируется модуль аннотации, Drupal ищет файл `annotate.install` и выполняет функцию `annotate_install()`. Если все идет хорошо, будут созданы таблицы базы данных. Сделайте это сейчас, запрещая и затем разрешая использование модуля.

На заметку Если вы сделали опечатку в вашем файле `.install` или его выполнение закончилось неудачей по другой причине, вы можете заставить Drupal забыть о вашем модуле и его таблицах, отключив модуль (Administer⇨Site building⇨Modules) и удалив строку модуля из таблицы `system` базы данных.

После создания таблицы для хранения данных необходимо будет несколько модифицировать наш код. Например, мы должны будем добавить некоторый код, чтобы обеспечить обработку данных, когда пользователь вводит аннотацию и щелкает на кнопке Update. Наша функция для обработки данных формы, отправленных серверу, следующая:

```
/*
 * Сохраняем аннотацию в базе данных.
 */
function annotate_entry_form_submit($form_id, $form_values) {
  global $user;
  $nid = $form_values['nid'];
  $note = $form_values['note'];
  db_query("DELETE FROM {annotations} WHERE uid = %d and nid = %d", $user->uid,
    $nid);
  db_query("INSERT INTO {annotations} (uid, nid, note, timestamp) VALUES
(%d, %d, '%s', %d)", $user->uid, $nid, $note, time());
  drupal_set_message(t('Your annotation was saved.'));
}
```

Поскольку мы разрешаем одному пользователю делать только одну аннотацию для одного узла, мы можем безопасно удалить предыдущую аннотацию (если она есть) и вставить в базу данных нашу собственную.

Есть несколько моментов, касающихся наших взаимодействий с базой данных, на которые следует обратить внимание. Во-первых, мы не должны волноваться о соединении с базой данных, потому что Drupal уже сделал это за нас при выполнении начальной загрузки. Во-вторых, при всяком обращении к таблице базы данных мы помещаем соответствующий код в фигурные скобки. Это делается для того, чтобы задание префиксов к таблице могло быть сделано незаметно (см. `http://drupal.org/node/2622`). И в-третьих, мы используем в наших запросах заполнители и затем предоставляем переменные, которыми они будут замещены, так, чтобы встроенный в Drupal механизм санации запроса мог внести свой вклад, дабы предотвратить SQL-атаки на основе ввода данных. Мы используем заполнитель `%d` для чисел и `'%s'` для строк. Затем мы используем функцию `drupal_set_message()`, чтобы скрыть сообщение в ходе сеанса пользователя, которое Drupal отобразит как примечание на следующей странице, которую увидит пользователь. Таким образом пользователь получает какую-то обратную связь.

Наконец, мы должны изменить наш код обработчика прерываний объекта `$node` так, чтобы, если уже есть какая-то аннотация, ее можно извлечь из базы

данных. Непосредственно перед тем, как мы присваиваем нашу форму свойству `content` объекта `$node`, мы добавляем следующие строки:

```
// Get previously saved note, if any.
$result = db_query("SELECT note FROM {annotations} WHERE uid = %d AND nid
= %d",
  $user->uid, $node->nid);
$node->annotation = db_result($result);
```

Сначала мы делаем запрос к таблице нашей базы данных, чтобы выбрать аннотацию для данного пользователя и данного узла. Затем мы используем функцию `db_result()`, чтобы получить первую строку из результирующего множества. Так как мы разрешаем отображать только одну аннотацию на пользователя для одного узла, должна быть только одна строка. Протестируйте ваш модуль. Он должен быть в состоянии хранить и находить аннотации. Поздравьте себя — вы создали модуль Drupal с нуля. Скоро вы присоединитесь к разработчикам ядра Drupal!

Дальнейшие шаги

Мы, конечно, будем использовать этот модуль совместно с сообществом приверженцев открытого исходного кода, поэтому нужно создать и поместить файл `README.txt` в каталог аннотаций — туда же, где хранятся файлы `annotate.info`, `annotate.module` и `annotate.install`. Затем вы могли бы выгрузить все это в репозиторий программ на сайте `drupal.org` и создать страницу проекта, чтобы иметь обратную связь от других членов сообщества.

Резюме

После прочтения этой главы вы должны уметь следующее.

- Создавать с нуля модули Drupal
- Понимать, как можно вмешиваться в выполнение кода Drupal
- Сохранять и извлекать определенные для модуля параметры настройки
- Создавать и обрабатывать простые формы, используя API форм Drupal
- Сохранять данные в базе данных Drupal и извлекать их оттуда