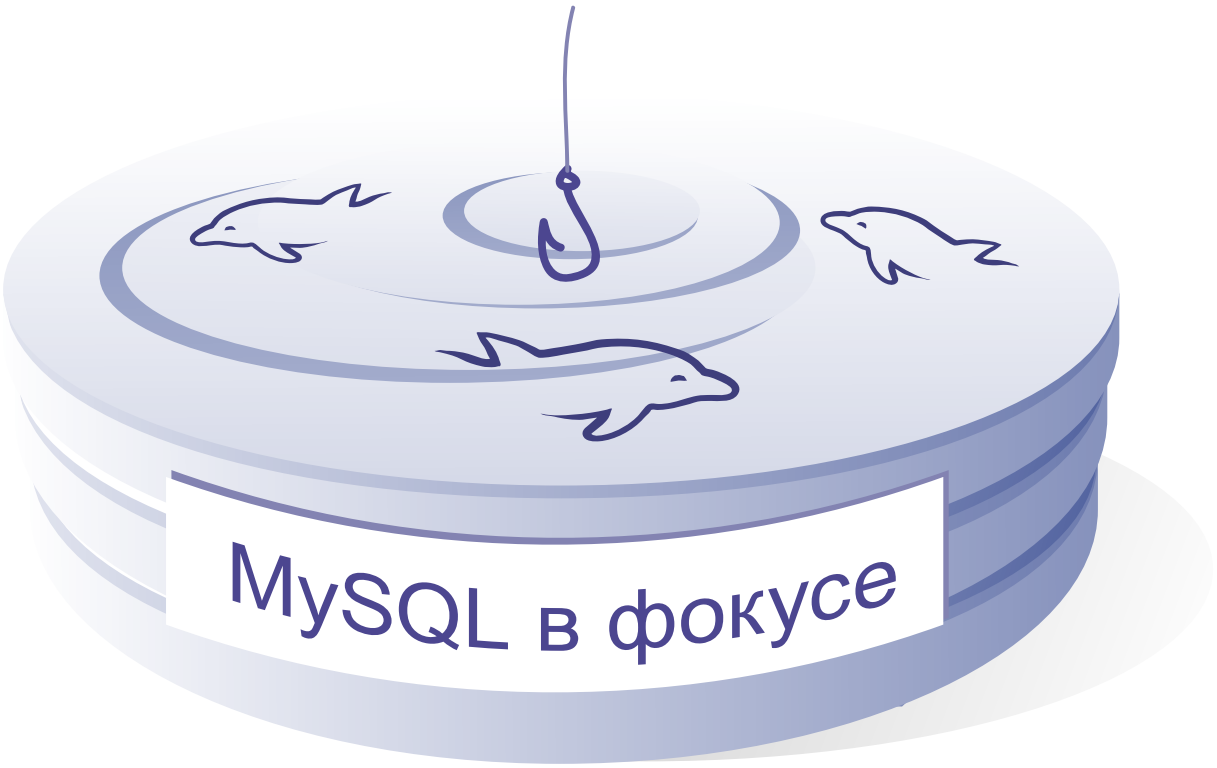
электронный журнал веб-разработчика

PHPInside

#2, Апрель'2004



Содержание

Обратная связь

Здравствуйте!

Мы рады новой встрече с вами!

Журнал постепенно начинает обретать свой стиль и формировать вокруг себя определенный образ. Однако три номера — это еще слишком мало. Мы только вышли в свет и продолжаем учиться тонкому мастерству — делать журнал о PHP и людях, живущих этой технологией. Здесь всегда могут пригодиться ваши знания и опыт.

В форуме phpclub'a или на адрес электронной почты project@yugovostok.ru приходят сообщения с вопросами, которые можно объеденить в один — "как делается PHP Inside?". На данный момент мы только отрабатываем схему взаимодействия, но, все же, кое что уже имеем. На этапе подготовки материала составляется wish list — список англоязычных статей, переводы которых редакция хотела бы видеть в новых номерах. Любой желающий участвовать в выпуске журнала своими переводами, может руководствоваться этим списком. Найти wish list всегда можно в форуме на http://phpclub.ru или запросить по адресу project@yugovostok.ru.

Еще один источник идей для перевода — http://detail.phpclub.net/pages/translate.phtml. Переведенные статьи проходят вычитку и коррекцию, затем верстаются и попадают в журнал. Пока что не очень отработана схема подготовки авторских материалов специально для журнала. Однако, мы работаем над этим и, надеюсь, ситуация скоро изменится в лучшую сторону. Параллельно с подготовкой контента ведется работа над макетом страниц и обложки. У PHP Inside появилась более качественная (и экономная для принтера) обложка, мы подсветили весь php-код в листингах и продолжаем ожидать от вас новых идей и мнений. Журнал открыт для вас, ваших идей и начинаний!

В фокусе

Шаблоны баз данных в MySQL

Автор:
Рассел Дайер

Перевод:
Александр Ширяев

С приобретением опыта создания баз данных (БД) MySQL появляются определенные навыки и приемы их разработки. Большинство БД имеют много общего. При создании новых БД разработчики зачастую строят одинаковые базовые таблицы с небольшими различиями в их именах и некоторыми корректировками в свойствах их столбцов. При построении новой базы разработчик часто берет для неё таблицы из существующих баз, дает им другие имена и потом уже модифицирует. Такой подход может сохранить уйму времени.

Но почему бы не пойти дальше? Ведь можно создать характерную БД общего назначения из набора пустых стандартных таблиц. Из хорошо спроектированного шаблона можно быстро построить основу для требуемой БД. Шаблон также позволяет вам сосредоточиться на более интересных аспектах проекта.

В этой статье я разберу процесс создания шаблона в MySQL и затем применю его на практическом примере. Потребности вашего клиента будут определять образ применения шаблона. Тем не менее, условимся, что главным образом мы будем разрабатывать БД для бизнеса по продаже осязаемого товара.

Ключевые таблицы

БД для бизнеса по продаже товара обычно содержит набор ключевых таблиц с описанием товара и информацией по нему. Большую часть занимает, как правило, каталог продукции. Для нашего шаблона мы создадим БД с названием product_template с ключевой таблицей products. Создадим ее из минимального количества основных столбцов:

```
CREATE DATABASE product_template;

USE product_template;

CREATE TABLE products (
    rec_id       INT             AUTO_INCREMENT PRIMARY KEY,
    rec_date     DATETIME,
    name         varchar(50),
    description  BLOB,
    category     VARCHAR(10),
    sub_category VARCHAR(10),
    status       ENUM('ac','in')
);
```

Проверим:

```
DESCRIBE products;
+--------------+-----------------+------+-----+---------+----------------+
| Field        | Type            | Null | Key | Default | Extra          |
+--------------+-----------------+------+-----+---------+----------------+
| rec_id       | int(11)         |      | PRI | NULL    | auto_increment |
| rec_date     | datetime        | YES  |     | NULL    |                |
| name         | varchar(50)     | YES  |     | NULL    |                |
| description  | blob            | YES  |     | NULL    |                |
| category     | varchar(10)     | YES  |     | NULL    |                |
| sub_category | varchar(10)     | YES  |     | NULL    |                |
| status       | enum('ac','in') | YES  |     | NULL    |                |
+--------------+-----------------+------+-----+---------+----------------+
```

Независимо от того, какие типы продуктов имеются в продаже у наших клиентов, мы, вероятно, можем использовать все эти столбцы. Любому продукту надо присвоить уникальный номер; создадим для этого первичный ключ rec_id. Не обязательно всегда заносить в БД дату создания записи, но это может оказаться полезным во многих случаях, поэтому мы зарезервируем отдельный столбец. Так как любой продукт нуждается в имени, у нас есть столбец name. Столбцы category и sub_category могут быть полезны при распределении товара по группам и подгруппам. Однако, их необходимость спорна.

Я создал столбец status, который позволит мне временно скрывать записи от пользователя не удаляя их. Почти все мои запросы SELECT содержат оператор WHERE, который исключает неактивные записи. Следовательно, достаточно изменить статус записи на in (от англ. inactive), чтобы исключить ее из результата запроса.

Возможно, вам покажется более целесообразным сделать тип этого столбца булевым, что позволит выбирать записи, статус которых равен true.

Мы могли бы внести в таблицу products и другие базовые столбцы, в которых могут нуждаться некоторые клиенты. Тем не менее, в качестве шаблона, набора этих столбцов вполне достаточно. Мы модифицируем таблицу под нужды воображаемого клиента, после того, как закончим разработку по шаблону.

Итоговые таблицы

Несмотря на то, что таблица products может также содержать и инвентарную информацию, под эти цели мы создадим отдельную таблицу inventory, тем самым получив возможность разделять записи по количеству для каждого элемента и месту его размещения. Тогда две таблицы будут связаны ключевым полем.

```
CREATE TABLE inventory (
    rec_id      INT       PRIMARY KEY,
    rec_date    DATETIME,
    description BLOB,
    product_id  INT,
    quantity    INT,
    store_id    INT
);
```

Проверим:

```
DESCRIBE inventory;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| rec_id      | int(11)  |      | PRI | 0       | auto_increment |
| rec_date    | datetime | YES  |     | NULL    |                |
| description | blob     | YES  |     | NULL    |                |
| product_id  | int(11)  | YES  |     | NULL    |                |
| quantity    | int(11)  | YES  |     | NULL    |                |
| store_id    | int(11)  | YES  |     | NULL    |                |
+-------------+----------+------+-----+---------+----------------+
```

Названия некоторых столбцов в этой таблице совпадают с именами столбцов из таблицы products. Кого-то это может смутить. Однако, это удобно тем, что не надо смотреть в описание таблиц, чтобы знать имена общих полей. При таком их использовании можно во всех таблицах рассчитывать на ключевое поле rec_id, поле описания description и статуса status.

Кроме стандартных столбцов, эта итоговая таблица включает в себя product_id, который связан с rec_id в таблице products. Это первый столбец, имеющий особое имя. Оно необходимо для того, чтобы избежать путаницы —мы знаем, с какой таблицей будет связан этот столбец. Конечно, можно использовать подобные имена и для ключевых полей в главной таблице (к примеру, product_id в products вместо rec_id), но это не обязательно. Например, мы знаем, что в таблице products ключевое поле — это идентификационный номер продукта. Назначение повторяющихся имен – частично стиль, частично индивидуальное предпочтение.

Следующий дополнительный столбец quantity отображает количество товара. И последний столбец, store_id, связывает с другой таблицей, содержащей информацию о местонахождении магазина заказчика.

Информационные таблицы

В целях экономии времени и ускорения работы БД многие разработчики используют информационную таблицу для хранения излишних данных. Такие таблицы могут хранить имена рабочих, информацию для покупателей, почтовые коды или местонахождения магазинов. Для нашего шаблона введем две минимальные характерные информационные таблицы. Первую создадим для основных данных, без адресов:

```
CREATE TABLE reference (
    rec_id      INT       AUTO_INCREMENT PRIMARY KEY,
    rec_date    DATETIME,
    name        VARCHAR(25),
    description BLOB,
    status      ENUM('ac','in')
);
```

Проверим:

```
DESCRIBE reference;
+-------------+-----------------+------+-----+---------+-------+
| Field       | Type            | Null | Key | Default | Extra |
+-------------+-----------------+------+-----+---------+-------+
| rec_id      | int(11)         | YES  |     | NULL    |       |
| rec_date    | datetime        | YES  |     | NULL    |       |
| name        | varchar(25)     | YES  |     | NULL    |       |
| description | blob            | YES  |     | NULL    |       |
| status      | enum('ac','in') | YES  |     | NULL    |       |
+-------------+-----------------+------+-----+---------+-------+
```

На данный момент, вероятно, наши характерные таблицы выглядят немного нудными и идентичными. Однако, именно эта серость позволяет создать легко модифицируемый и удобный для адаптации к конкретному клиенту шаблон. Именно это позволяет разработчику писать SQL-запросы быстро и интуитивно.

Таблицу, хранящую адреса, назовем addresses, в ней будет много общих столбцов:

```
CREATE TABLE addresses (
    rec_id       INT          AUTO_INCREMENT PRIMARY KEY,
    rec_date     DATETIME,
    name         VARCHAR(50),
    name2        VARCHAR(50),
    description  BLOB,
    category     VARCHAR(10),
    sub_category VARCHAR(10),
    address1     VARCHAR(50),
    address2     VARCHAR(50),
    city         VARCHAR(50),
    state        VARCHAR(8),
    postal_code  VARCHAR(10),
    country      VARCHAR(25),
    contact      VARCHAR(50),
    telephone    VARCHAR(15)
);
```

Проверим:

```
DESCRIBE addresses;
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| rec_id       | int(11)     |      | PRI | NULL    | auto_increment |
| rec_date     | datetime    | YES  |     | NULL    |                |
| name         | varchar(50) | YES  |     | NULL    |                |
| name2        | varchar(50) | YES  |     | NULL    |                |
| description  | blob        | YES  |     | NULL    |                |
| category     | varchar(10) | YES  |     | NULL    |                |
| sub_category | varchar(10) | YES  |     | NULL    |                |
| address1     | varchar(50) | YES  |     | NULL    |                |
| address2     | varchar(50) | YES  |     | NULL    |                |
| city         | varchar(50) | YES  |     | NULL    |                |
| state        | varchar(8)  | YES  |     | NULL    |                |
| postal_code  | varchar(10) | YES  |     | NULL    |                |
| country      | varchar(25) | YES  |     | NULL    |                |
| contact      | varchar(50) | YES  |     | NULL    |                |
| telephone    | varchar(15) | YES  |     | NULL    |                |
+--------------+-------------+------+-----+---------+----------------+
```

Эта таблица содержит два столбца имен: один для имени, второй для фамилии (опять же, вы можете переименовать эти столбцы на свое усмотрение). Эта таблица с адресами, вероятно, лучше всего подходит для людей, потому вы можете создать еще одну - для учреждений.

Отталкиваясь от нужд клиента, мы используем приведенные выше характерные таблицы для создания нескольких таблиц свойств с разными именами и назначениями. При копировании мы могли бы переименовывать некоторые из них, но я предпочитаю этого не делать. В каждом отдельном случае нам надо будет дополнять таблицы специфичными столбцами. Переходим к следующей части, в которой рассматриваются несколько конкретных примеров.

Использование шаблона

Как только мы создали шаблон БД, можем приступить к использованию его для разработки БД для клиента. В этой части мы представим, что заключили контракт на разработку БД для книжного магазина.

Первым делом копируем шаблон в новую БД. Для этого можно использовать различные утилиты, но лучше мы создадим новую БД `bookstore`, а затем скопируем таблицы из `product_template` в таблицу `products` в новой БД, используя оператор `CREATE TABLE...SELECT`.

```
CREATE DATABASE bookstore;

USE bookstore;

CREATE TABLE books
SELECT * FROM products_template.products;
```

Хоть это и вполне работающий метод, но у него есть два недостатка. Во-первых, вместе со структурой таблицы будет скопирована информация. Это не принципиально, если мы копируем таблицу из шаблона — в ней не будет никакой информации. Но если он и есть, тогда с использованием простого `DELETE` можно очистить новую таблицу.

Во-вторых, проблема с ключевым полем rec_id — вышеприведенный метод не продублирует индексы, и столбец не будет автоинкрементен. Сам столбец rec_id создан, но не проиндексирован, и у него нет свойства auto_increment. В таком случае мы можем просто изменить новую таблицу:

```
ALTER TABLE books
CHANGE COLUMN rec_id
rec_id INT AUTO_INCREMENT PRIMARY KEY;
```

Альтернативой `CREATE TABLE...SELECT` является выполнение команд `SHOW CREATE TABLE`, исправление имен новых таблиц и затем запуск модифицированного кода.

```
SHOW CREATE TABLE products_template.products;

CREATE TABLE `products` (
  `rec_id`       int(11)         NOT     NULL auto_increment,
  `rec_date`     datetime        default NULL,
  `name`         varchar(50)     default NULL,
  `description`  blob,
  `category`     varchar(10)     default NULL,
  `sub_category` varchar(10)     default NULL,
  `status`       enum('ac','in') default NULL,
  PRIMARY KEY  (`rec_id`)
) TYPE=MyISAM
```

Здесь приведена только суть — без псевдографики форматирования таблицы. Теперь можно скопировать этот текст (часть `CREATE TABLE...`) в текстовый редактор и поменять имя таблицы с products на books. Потом вставим исправленный текст в mysql-клиент для создания пустой новой таблицы с требуемыми настройками. Конечно, если мы вынуждены делать это больше, чем один раз, то стоит задуматься над написанием скрипта, автоматизирующего этот процесс.

Теперь когда у нас есть наброски books, мы можем настроить таблицу под нужды клиента. Таблица продукции для книжного магазина требует некоторых дополнительных столбцов. Также надо связать books с inventory и парочкой таблиц свойств:

```
ALTER TABLE books
ADD COLUMN author_id INT,
ADD COLUMN genre INT,
ADD COLUMN publisher INT,
ADD COLUMN pub_year CHAR(4),
ADD COLUMN isbn VARCHAR(20);
```

```
DESCRIBE books;
+--------------+-----------------+------+-----+---------+----------------+
| Field        | Type            | Null | Key | Default | Extra          |
+--------------+-----------------+------+-----+---------+----------------+
| rec_id       | int(11)         |      | PRI | NULL    | auto_increment |
| rec_date     | datetime        | YES  |     | NULL    |                |
| name         | varchar(50)     | YES  |     | NULL    |                |
| description  | blob            | YES  |     | NULL    |                |
| category     | varchar(10)     | YES  |     | NULL    |                |
| sub_category | varchar(10)     | YES  |     | NULL    |                |
| status       | enum('ac','in') | YES  |     | NULL    |                |
| author_id    | int(11)         | YES  |     | NULL    |                |
| genre        | int(11)         | YES  |     | NULL    |                |
| publisher    | int(11)         | YES  |     | NULL    |                |
| pub_year     | varchar(4)      | YES  |     | NULL    |                |
| isbn         | varchar(20)     | YES  |     | NULL    |                |
+--------------+-----------------+------+-----+---------+----------------+
```

Основываясь на столбцах, которые мы добавили в products, нам надо сделать копии таблиц свойств: две копии reference для авторов и жанров и копию addresses для информации об издательстве.

Целью данного упражнения было показать, что всего за несколько минут можно подготовить хорошую базу для создания БД новому клиенту. У вас есть таблица продуктов, инвентарная таблица и различные таблицы свойств, которые связаны с главной таблицей. Все, что вам необходимо сделать, так это разработать парочку дополнительных таблиц под нужды клиента.

Коллекция шаблонов

Примеры БД, приведенные в этой статье, рассчитаны на клиентов, продвигающих бизнес по продаже товара. Все они также с легкостью могут быть модифицированы и под других клиентов, если такая необходимость возникнет.

Выполняя заказ для клиента, различайте, какие столбцы типичны для всех БД (и давайте им соответствующие имена), а какие характерны именно этому клиенту. Когда с основными таблицами будет покончено, сделайте копию БД и удалите из нее характерные этому клиенту столбцы. Теперь добавьте новый шаблон в свой инструментарий на будущее. Со временем вы будете готовы к созданию БД любого типа и сможете быстро создавать даже большие и сложные БД.

Оригинал статьи находится по адресу:
http://www.onlamp.com/pub/a/onlamp/2004/03/18/mysql_templates.html

Оптимизация запросов в MySQL

Автор:
Сергей Соколов

Оптимизация - это изменение системы с целью повышения ее быстродействия.

Оптимизацию работы с БД можно разделить на 3 типа:

- оптимизация запросов
- оптимизация структуры
- оптимизация сервера.

Рассмотрим подробнее оптимизацию запросов. Оптимизация запросов — наиболее простой и приводящий к наиболее высоким результатам тип оптимизации.

SELECT

Запросами, которые чаще всего поддаются оптимизации, являются запросы на выборку. Для того чтобы посмотреть как будет выполняться запрос на выборку используется оператор EXPLAIN: http://www.mysql.com/doc/ru/EXPLAIN.html

С его помощью мы можем посмотреть, в каком порядке будут связываться таблицы и какие индексы при этом будут использоваться. Основная ошибка начинающих — это отсутствие индексов на нужных полях или создание оных на ненужных полях. Если вы делаете простую выборку наподобие:

```
SELECT * FROM table WHERE field1 = 123
```

То вам нужно проставить индекс на поле field1, если вы используете в выборке условие по двум полям:

```
SELECT * FROM table WHERE field1 = 123 AND field2 = 234
```

То вам нужно создать составной индекс на поля field1, field2. Если вы используете соединение 2 или более таблиц:

```
SELECT *
FROM a, b
WHERE a.b_id = b.id
```

Или в более общем виде:

```
SELECT * FROM a
[LEFT] JOIN b ON b.id = a.b_id
[LEFT] JOIN c ON c.id = b.c_id
```

То вам следует создать индексы по полям, по которым будут присоединятся таблицы. В данном случае это поля b.id и c.id. Однако это утверждение верно только в том случае, если выборка будет происходить в том порядке, в котором они перечислены в запросе. Если, к примеру, оптимизатор MySQL будет выбирать записи из таблиц в следующем порядке: c,b,a, то нужно будет проставить индексы по полям: b.c_id и a.b_id.

При связывании с помощью LEFT JOIN таблица, которая идет в запросе слева, всегда будет просматриваться первой.

Про синтаксис создания индексов можно прочитать в документации: http://www.mysql.com/doc/ru/CREATE_INDEX.html.

Более подробно про использовании индексов можно прочитать здесь: http://www.mysql.com/doc/ru/MySQL_indexes.html.

Иногда бывает такая ситуация, что нам постоянно приходится делать выборки из одной и той же части некоторой очень большой таблицы, например, во многих запросах происходит соединение с частью таблицы:

```
[LEFT] JOIN b ON b.id = a.b_id AND b.field1 = 123 AND b.field2 = 234
```

В таких случаях может быть разумным вынести эту часть в отдельную временную таблицу:

```
CREATE TEMPORARY TABLE tmp_b TYPE=HEAP SELECT * FROM b WHERE b.field1 =
123 AND b.field2 = 234
```

И работать уже с ней (про временные таблицы читайте в документации http://www.mysql.com/doc/ru/CREATE_TABLE.html).

Также если мы несколько раз рассчитываем агрегатную функцию для одних и тех же данных, то для ускорения следует сделать такой расчет отдельно и положить его результат во временную таблицу.

Бывают проблемы с быстродействием, когда люди пытаются в одном запросе «поймать сразу 2-х зайцев», например, на форуме phpclub'a автор следующего запроса спрашивал, почему он работает медленно:

```
SELECT f_m. *, MAX( f_m_v_w.date ) AS last_visited, COUNT( DISTINCT f_b.id
) AS books_num,
IF ( f_m.region != 999, f_r.name, f_m.region_other ) AS region_name
FROM fair_members f_m
LEFT JOIN fair_members_visits_week f_m_v_w ON f_m_v_w.member_id = f_m.id
LEFT JOIN fair_regions AS f_r ON f_m.region = f_r.id
LEFT JOIN fair_books AS f_b ON f_b.pub_id = f_m.id
GROUP BY f_m.id
```

Автор запроса пытается в одном запросе посчитать максимальное значение атрибута из одной таблицы и кол-во записей в другой таблице. В результате к запросу приходится присоединять 2 разные таблицы, которые сильно замедляют выборку. Для увеличения быстродействия такой выборки необходимо вынести подсчет MAX'а или COUNT'а в отдельный запрос.

Для подсчета кол-ва строк используйте функцию COUNT(*), с указанием "звездочки" в качестве аргумента. Почему COUNT(*) обычно быстрее COUNT(id), поясню на примере. Есть таблица message: id | user_id | text с индексом PRIMARY(id), INDEX (user_id). Нам надо подсчитать сообщения пользователя с заданым $user_id.

Сравним 2 запроса:

```
SELECT COUNT(*) FROM message WHERE user_id = $user_id
```

а также

```
SELECT COUNT(id) FROM message WHERE user_id = $user_id
```

Для выполнения первого запроса нам достаточно просто пробежаться по индексу user_id и подсчитать кол-во записей, удовлетворяющих условию — такая операция достаточно быстрая, т.к., во-первых, индексы у нас упорядочены и ,во-вторых, часто находятся в буфере.

Для выполнения второго запроса мы сначала проходим по индексу, для отбора записей удовлетворяющих условию, после чего если запись попадает под условие, то вытаскиваем ее (запись скорее всего будет на диске) чтобы получить значение id и только потом инкриментим счетчик.

В итоге получаем, что при большом количестве записей скорость первого запроса будет выше в несколько раз.

UPDATE, INSERT

Скорость вставок и обновлений в базе зависит от размера вставляемой (обновляемой) записи и от времени вставки индексов.

Время вставки индексов в свою очередь зависит от количества вставляемых индексов и размера таблицы. Эту зависимость можно выразить формулой: [Время вставки индексов] = [кол-во индексов] * LOG2([Размер таблицы]).

При операциях обновления под [кол-во индексов] понимаются только те индексы, в которых присутствуют обновляемые поля.

Условия в запросах на обновления оптимизируются так же, как и в случае с выборками. При частом изменении некоторой большой таблицы с большим количеством индексов имеет смысл производить вставки в другую небольшую вспомогательную таблицу с тем же набором полей (но с отсутствием индексов) и периодически перекидывать данные из нее в основную таблицу, очищая вспомогательную. При этом следует учесть, что данные будут выводиться с запозданием, что не всегда может быть приемлемым.

В официальной документации MySQL говорится, чтобы удалить все строки в таблице, нужно использовать команду `TRUNCATE TABLE table_name`.

Ответы на многие вопросы по оптимизации запросов можно найти в мануале: http://www.mysql.com/doc/ru/Query_Speed.html

Статья предоставлена сайтом: http://detail.phpclub.net

Идеи

Постигните Дао PEAR::DB и Smarty!

Автор:
Джо Прадо Майя

Перевод:
Игорь Луканин

Полное разделение бизнес-логики и логики представления является одной из целей, которую часто преследуют разработчики веб-приложений. Однако сам PHP компромиссен в этом смысле — вы можете как добавлять элементы разметки в бизнес-логику, так и реализовывать бизнес-логику в шаблонах.

Наши цели

Некоторое время я пользовался этим в собственных разработках, пока не пришел к заключению, после проб и ошибок на своих собственных программах, что существует гораздо более удобный способ интеграции приложения, использующего базу данных, с мощным шаблонным движком.

Наверно, вы уже догадались, что я говорю о PEAR::DB и Smarty, и я хочу рассказать вам в этой статье о своем опыте работы с ними. Прочитав статью, вы получите представление о том, как спроектировать приложение, при этом разделив бизнес-логику и логику представления.

Библиотеки для абстрактной работы с базой данных являются одними из наиболее распространенных для большинства языков программирования, например JDBC для Java или DBI для Perl. PHP не является исключением — существует бесчисленное множество таких библиотек, например, PEAR::DB. Библиотеки для работы с шаблонами (шаблонные движки) тоже довольно часты среди компонентов веб-приложений, написанных на многих языках, например на Perl или Cold Fusion. Smarty — шаблонный движок, поддерживаемый Монтом Ортом (Monte Ohrt) и Андреем Змиевски (Andrei Zmievski), разработчиком ядра PHP. Smarty отличается от существующих решений своей простотой и, одновременно, большой мощностью: например, существует возможность кешировать шаблоны в PHP-скрипты, что дает большой прирост производительности.

Установка

Когда я работаю с PEAR::DB и Smarty, я всегда стараюсь сделать код программы как можно более расширяемым, независимо от того, какой класс для абстрактной работы с базой данных или шаблонный движок я буду использовать в будущем.

Это означает, что необходимо создать класс-интерфейс к шаблонному движку, который скроет специфические особенности Smarty.

Пример такого класса приведен в листинге 1.

```
<?php

require_once('config.inc.php') ;
require_once(SMARTY_PATH . 'Smarty.class.php') ;

class template
{       var $smarty ;
        var $template ;

        function template()
        {
      $this -> smarty = new Smarty ;
      $this -> smarty -> template_dir = APP_PATH . 'templates/' ;
      $this -> smarty -> compile_dir = APP_PATH . 'templates_c' ;
      $this -> smarty -> config_dir = APP_PATH . 'configs/' ;
        }

        function set_template($name)
        { $this -> template = $name ; }

        function assign($name,$value = false)
        {       if ($value) { $this -> smarty -> assign($name,$value) ; }
                else { $this -> smarty -> assign($name) ; }
        }

        function display_template()
        { $this -> smarty -> display($this -> template) ; }

        function get_template_contents()
        { return($this -> smarty -> fetch($this -> template)) ; }
}
?>
```

Листинг 1

APP_PATH и SMARTY_PATH — это константы, определяемые в файле конфигурации. Они указывают путь к скриптам всего приложения и путь к файлам Smarty соответственно. Обычно я включаю Smarty и PEAR::DB в состав своих приложений, чтобы избавиться от возможных проблем с версиями библиотек, например, если администратор сервера решит обновить PEAR.

В любом случае вышеприведенный класс позволяет создавать скрипты, абстрагированные от особенностей реализации конкретного шаблонного движка. Посмотрим на листинг 2.

```
<?php

require_once('config.inc.php') ;
require_once(INCLUDE_PATH . 'class.template.php') ;

$template = new template ;
$template -> set_template('example.tpl.html') ;
$template -> assign('links','list of links goes here') ;
$template -> display_template() ;
?>
```

Листинг 2

INCLUDE_PATH, как вы уже, наверно, догадались, — путь к классам моей программы.

Скорее всего, вы заметили, что вышеприведенный скрипт не содержит ничего, так или иначе связанного с представлением. Это очень простой скрипт, но при использовании такого подхода вы всегда будете свободны от логики представления в скриптах, реализующих бизнес-логику.

Функции шаблонного языка Smarty

Одним из наиболее полезных особенностей Smarty является то, что доступно большое количество встроенных функций, которые вы можете использовать непосредственно в шаблонах. Может показаться, что нет никакой необходимости в использовании функций в шаблонах, хотя бы потому, что изначально требовалось отделить бизнес-логику от представления. Однако, это будет именно логика представления, а не бизнес-логика!

Я очень часто пользуюсь функцией html_options. Простой пример ее использования приведен в листинге 3.

```php
<?php

include_once('config.inc.php') ;
include_once(INCLUDE_PATH . 'class.template.php') ;

$template = new template ;
$template -> set_template('example.tpl.html') ;

$states = array(
        'CA' => 'California',
        'TX' => 'Texas',
        'NY' => 'New York'
) ;

// первый список выбора
$template -> assign('states',$states) ;

// второй список выбора
$template -> assign(array(
        'abbreviations' => array_keys($states),
        'titles' => array_values($states)
)) ;

$template -> display_template() ;

?>
```

Листинг 3

Вы можете создать список выбора прямо из ассоциативного массива:

```
<select name="states">
{html_options options=$states}
</select>
```

Для этой же цели вы можете использовать два массива значений:

```
<select name="other_states">
{html_options values=$abbreviations output=$titles}
</select>
```

Такая техника легко интегрируется с возможностями PEAR::DB. В данном случае весьма удобно использование функций get*(), пример в листинге 4.

```
<?php

require_once('config.inc.php') ;
require_once(INCLUDE_PATH . 'class.template.php') ;

$template = new template ;
$template -> set_template('example.tpl.html') ;

require_once('DB.php') ;
$db = DB::connect('mysql://user:password@server/database') ;

$states = $db -> getassoc('SELECT abbrev, title FROM states') ;
$template -> assign('states',$states) ;
$template -> display_template() ;

?>
```

Листинг 4

В этом и состоит интеграция класса абстрактной работы с базой данных и шаблонного движка.

Работа со сложными выборками из базы данных

`Html_options` является очень удобным средством составления списков выбора непосредственно из данных, полученных из БД. Однако, Smarty также обладает другими возможностями, позволяющими обрабатывать в шаблоне данные сложных выборок, применяя, к примеру, циклы и условные операторы (см. листинг 5).

```
<?php

require_once('config.inc.php') ;
require_once(INCLUDE_PATH . 'class.template.php') ;

$template = new template ;
$template -> set_template('example.tpl.html') ;
require_once('DB.php') ;

$db = DB::connect('mysql://user:password@server/database') ;
$news=$db->getall('SELECT * FROM news ORDER BY news_id DESC',DB_FETCHMODE_ASSOC);
$template -> assign('news',$news) ;
$template -> display_template() ;

?>
```

Листинг 5

И далее:

```
<table>
    <tr>
        <td>Номер</td>
        <td>Дата</td>
        <td>Заголовок</td>
        <td>Автор</td>
    </tr>
{section name="i" loop=$news}
    <tr>
        <td>{$news[i].news_id}</td>
        <td>{$news[i].news_date}</td>
        <td>{$news[i].news_title}</td>
        <td>{$news[i].news_author}</td>
    </tr>
{/section}
</table>
```

Функция `section` работает, как цикл `for` в PHP. Если же вам требуется показывать часть шаблона только при наступлении определенного условия, используйте такой пример:

```
<table>
    <tr>
        <td>Номер</td>
        <td>Дата</td>
        <td>Title</td>
        <td>Author</td>
</tr>
{section name="i" loop=$news}
{cycle values="#CCCCCC,#999999" assign="row_color"}
    <tr bgcolor="{$row_color}">
        <td>{$news[i].news_id}</td>
        <td>{$news[i].news_date}</td>
        <td>{$news[i].news_title}</td>
        <td>{$news[i].news_author}</td>
    </tr>
{if $smarty.section.i.last}
    <tr>
        <td colspan="4">Total of {$smarty.section.i.total} news entries
found.</td>
    </tr>
{/if}
{sectionelse}
    <tr>
        <td colspan="4">No news could be found.</td>
    </tr>
{/section}
</table>
```

В этом примере использованы новые функции Smarty:

Функция `cycle` принимает строку, содержащую список значений, разделенных запятой, и последовательно присваивает их переменной, указанной в атрибуте `assign`. Таким образом, вы можете "раскрасить" строки таблицы в разные цвета.

Переменная `$smarty.section.i` содержит информацию о цикле `i` (это имя вы должны указать в атрибуте `name`).

Вы можете узнать, является ли текущая итерация последней из переменной `$smarty.section.i.last`, а общее количество итераций содержится в переменной `$smarty.section.i.total`.

Часть `sectionelse` отображается, если массив, указанный в атрибуте `loop`, был пуст. Вы можете использовать это, чтобы написать, например: "Ничего не найдено".

Разделение файлов шаблонов

Как и PHP-скрипты, шаблоны могут становиться настолько большими, что становится целесообразным разделить большой шаблон на несколько малых. Часто легче и лучше включить в шаблон кусок HTML-кода из другого файла, чем повторять один и тот же код в разных файлах, например:

```
{include file="header.tpl.html"}
{include file="adverts.tpl.html"}
<!-- основное содержание страницы -->
{include file="copyright.tpl.html"}
{include file="footer.tpl.html"}
```

Очень полезно делать ваши приложения модульными, и, как следствие, более гибкими при изменениях. Если вы обычно помещаете копирайт в конце страницы, за исключением всплывающих окон, то удобным будет перемещение HTML-кода копирайта в отдельный шаблон.

Функция `include` имеет другую интересную функциональность — вы можете передавать переменные-флаги во включаемый шаблон, например, чтобы контролировать его отображение.

```
{include file="header.tpl.html" title="Особый заголовок"}
```

Представим, что обычно у вас такой шаблон:

```
<title>Имя сайта — слоган</title>
```

Однако, он может содержать следующий код:

```
<title>Имя сайта — {$title|default:"слоган"}</title>
```

Это очень простой пример, но он показывает, какую логику вы можете помещать в свои шаблоны.

Заключение

Я надеюсь, что эта статья была полезна тем, что продемонстрировала лучшие возможности интеграции PEAR::DB и Smarty. Нелишним, однако, будет заглядывать иногда в документацию.

PEAR Manual (http://pear.php.net/manual/ru/)

Smarty Manual (http://smarty.php.net/manual/ru/)

Оригинал статьи находится по адресу: http://www.onlamp.com/pub/a/php/2003/04/17/pear_smarty.html

Тестирование веб-приложений на безопасность

Автор:
Нитеш Дханжани

Перевод:
Эмиль Хасанов

В этой статье рассматриваются некоторые общие недостатки в сфере безопасности веб-приложений и демонстрируются способы их выявления,. Также будет продемонстрирован PHP-скрипт, работающий как прокси-сервер, и позволяющий прерывать и изменять запросы HTTP между браузером и атакуемым веб-сервером. Этот скрипт может хорошо помочь в проверке недостатков безопасности.

Для начала посмотрим на некоторые бреши в безопасности веб-приложений, которые могут быть протестированы с использованием обычного прокси-сервера. Самые распространенные проблемы связаны с проверкой входных данных на достоверность.

Проверка данных на достоверность

Рассмотрим скрипт `login.php`, который проверяет права доступа к веб-сайту. Предположим, он принимает два параметра: username и password — методом POST протокола HTTP. Вставка в исходный код этого сценария проверки аутентификации может выглядеть следующим образом:

```
$sqlquery="SELECT * FROM USERS WHERE username='$username'
    AND password='$password'";

if(function_to_perform_query($sqlquery)) {
    //Аутентифицирован!!!
}
else {
    //Неправильный username или password. Ошибка!
}
```

На первый взгляд, все просто, но, предположим, что пользователь введет значение «admin» в поле username формы аутентификации, и пароль следующего содержания:

```
a' or 'a'='a
```

Тогда переменная `$sqlquery` будет содержать следующее значение:

```
SELECT * FROM USERS WHERE username='admin' AND password='a' or 'a'='a'
```

Если выполнить этот код, то пользователь получит права пользователя "admin". Причина очевидна и проста. Условие запроса SQL всегда возвратит истину, поскольку существует дополнительное условие "or 'a'='a'", которое, конечно же, будет истиной (поскольку "a" всегда равно "a").

Такие вставки в SQL-запросы в непроверяющих и неочищающих входные данные веб-приложениях, часто называют SQL injection”.

Большинство языков программирования имеют функцию, которая позволяет запускать внешние программы. Многие слабо проработанные веб-приложения не производят проверку входных данных, что приводит к возможности запуска этих программ с потенциально опасными параметрами.

В качестве примера, можно привести следующий код:

```
system("/bin/echo $i");
```

Если параметр `$i` был принят от пользователя, то он может присвоить ему такое значение:

```
`cat /etc/passwd`
```

Это заставит `system()` выполнить следующее:

```
/bin/echo `cat /etc/passwd`
```

Таким образом, на экран вместо `$i` выводится содержимое файла `/etc/passwd`.

Защитить системный вызов от потенциально опасных данных может очистка введенных данных от специальных символов. Рекомендуется обратить внимание, в первую очередь, на следующие:

```
` . ; \ / @ & | % ~ < > " $ ( ) { } [ ] * ! '
```

В дополнение, для того, чтобы PHP экранировал данные, полученные методами GET и POST, автоматически, можно отредактировать файл `php.ini` и установить значение параметра `magic_quotes_gpc` в On. Более подробную информацию можно найти в соответствующем разделе документации по PHP (прим. перев.:http://www.php.net/manual/ru/ref.info.php#ini.magic-quotes-gpc).

Сессии

Поскольку HTTP не модифицируемый (прим. перев.: динамичный, меняющий параметры своего состояния. ориг — stateful) протокол, то веб-приложения должны сами обеспечивать управление сессиями, если требуется их использование. Многие механизмы сессий используют сохранение состояния значений на стороне клиента в cookies. Cookies представляют собой небольшие порции информации, которые сохраняются на жестком диске пользователя по запросу веб-приложений.

Пользователь или программы на его компьютере могут модифицировать их, поэтому такие данные также необходимо проверять. Слабо проработанные веб-приложения часто доверяют данным, полученным из cookies, сохраненных на стороне клиента, это делается, например, для упрощения механизма аутентификации для пользователя.

Рассмотрим cookie, содержащий следующие данные:

```
lang=en-us; user=joe; time=10:10 EST;
```

Имеющий злые намерения пользователь может попытаться изменить эту cookie, вставив значение admin или root вместо joe, и получить права высокого уровня.

Веб-приложение также может хранить данные, помещая их в URL:

```
http://www.example.com/authenticated.cgi?user=john&sessionid=12345678
```

Часто веб-приложения сохраняют идентификаторы сессии последовательно. Это позволяет пользователю john подменить значение sessionid, меньше чем 12345678, для доступа к сессиям пользователей, которые вошли в систему перед ним.

Спрятанные элементы HTML

В веб-формах часто содержатся скрытые элементы. Их значения не отображаются посетителю, но используются, чтобы передавать информацию на сервер и между разными формами. Вот пример HTML-кода скрытого элемента:

```
<INPUT NAME="shippingcharges" TYPE=HIDDEN VALUE="5.25">
```

Проблема в данном случае в том, что злонамеренный пользователь может изменить HTML и послать любое значение, какое захочет. Часто веб-приложения полностью доверяют вводу с клиентской стороны. Так, к примеру, пользователь может использовать прокси-сервер (как мы покажем далее) для изменения значения shippingcharges, которое вводится в форме покупки. Представим, что случится, если, к примеру, пользователь введет значение -100. Если веб-приложение слепо примет это значение, то клиент совершит покупку, да еще получит деньги на свою кредитную карточку!

ПРЕДУПРЕЖДЕНИЕ: Все вышеописанные проблемы с безопасностью веб-приложений приведены для использования в нашем примере с собственным прокси-сервером. Они могут считаться полным набором всех проблем с безопасностью в веб-приложениях.

Наш собственный прокси-сервер

Веб-приложения должны корректно проверять данные, вводимые пользователем. Для того, чтобы проверить веб-приложение на доверчивость данным, полученным методом GET, все, что необходимо сделать — это изменить значения параметров в адресной строке. Дело в том, что при передаче методом GET, параметры просто добавляются в URL:

```
http://www.blahwebserver.com/script.cgi?login=admin&display=true
```

Но, в случае использования метода POST, все становится гораздо сложнее. В отличие от метода GET, данные, переданные методом POST, не показываются в URL.

Вот пример запросов POST, посылаемых браузером веб-серверу:

```
POST /script.cgi HTTP/1.0\r\n
Host: www.blahwebserver.com\r\n
Content-Length: 24\r\n\r\n
login=admin&display=true
```

Конечно, можно послать запрос методом POST так, как это указано выше, используя программу telnet для соединения с веб-сервером и вводя команды вручную, но это не удобно. Для того, чтобы упростить себе жизнь, напишем прокси, который поможет изменять запросы, передаваемые методом POST на лету, так, как будто эти значения вводятся в браузере. Просмотрите исходный текст в файле `proxy.php` (прим. ред.: в приложении к журналу). В дополнение к перехвату данных, передаваемых методом POST, вы можете также захотеть перехватывать и изменять значения cookie, которые присутствуют в HTTP-заголовках. Следовательно, если нужно, просто удалите это условие `if` в `proxy.php` и вы сможете изменять любой запрос от браузера:

```
if($ispost==1)
```

Вам понадобится консольная программа PHP для запуска скрипта `proxy.php`. Запустите его следующим образом:

```
[bash]$ php -q proxy.php
```

ПРЕДУПРЕЖДЕНИЕ: Если вы получаете сообщение об ошибке о неопределенных функциях с сокетами, то это, вероятнее всего, означает, что ваша версия PHP не поддерживает свежие расширения функций работы с сокетами. Вам потребуется перекомпилировать PHP запуском ./configure с флагом "--enable-sockets".

Теперь необходимо изменить настройки прокси своего браузера, вписав адрес IP 127.0.0.1 и порт 8000. Как только это сделано, необходимо удостовериться, что просмотр веб-страниц работает как обычно. Всякий раз, когда браузер будет отправлять запрос методом POST, скрипт `proxy.php` засечет это и позволит изменять запрос в любом текстовом редакторе.

Для того, чтобы работать как сервер, скрипт `proxy.php` связывается с портом, используя функции `socket_bind` и `socket_listen`. Как только это сделано, скрипт начинает крутиться в цикле, ожидая входящих соединений, отвечая каждому новому соединению с использованием функции `socket_accept`, которая блокирует скрипт на время данного соединения. Предупреждаем, что данная реализация не обеспечивает мультипоточность. Впрочем, для исследовательских целей вполне достаточно и такого варианта.

Когда браузеру требуется доступ к ресурсу методом POST, он соединяется с прокси-сервером и посылает запрос примерно следующего содержания:

```
POST http://www.blahwebserver.com/login.php HTTP/1.1\r\n
Host: www.blahwebserver.com\r\n
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;
en-US; rv:1.5) Gecko/20031007\r\n
Accept: text/xml,application/xml,application/xhtml+xml,text
/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif
;q=0.2,*/*;q=0.1\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Proxy-Connection: keep-alive\r\n
Referer: http://www.blahwebserver.com/login.php\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 25\r\n
\r\n
login=admin&pass=p455w0rd
```

Получив такой запрос, `proxy.php` проверит метод запроса: GET или POST. Запрос GET заканчивается комбинацией символов \r\n\r\n. Тем не менее, если это запрос по методу POST, как в примере выше, поле Content-Length определяет длину строки с указываемыми параметрами. Скрипт `proxy.php` получает эти данные и ожидает указанные 25 символов (login=admin&pass=p455w0rd), которые должны находиться после HTTP-заголовка, заканчивающегося комбинацией символов \r\n\r\n.

Запрос сохраняется во временном файле в каталоге /tmp. Если запрос происходит методом POST, запускается редактор, определенный в переменной $editor. Это позволяет пользователю редактировать POST-запрос на лету. Как только пользователь сохранит измененный файл и выйдет из редактора — и, предположим, пользователь изменил запрос для внедрения и выполнения SQL-кода путем отправки `login=admin&pass=a' or 'a'='a` как данных в POST-запросе — запрос изменится следующим образом:

```
POST /login.php HTTP/1.0\r\n
Host: www.blahwebserver.com\r\n
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;
en-US; rv:1.5) Gecko/20031007\r\n
Accept: text/xml,application/xml,application/xhtml+xml,text
/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif
;q=0.2,*/*;q=0.1\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Referer: http://www.blahwebserver.com/login.php\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 29\r\n
\r\n
login=admin&pass=a' or 'a'='a
```

Обратите внимание, что прокси изменил HTTP/1.1 на HTTP/1.0, что бы не думать о параметре keep-alive, который является частью спецификации HTTP/1.1.

Также скрипт удаляет строку Proxy-Connection, и, соответственно, пересчитывает значение Content-Length перед отсылкой на веб-сервер.

После отсылки запроса на веб-сервер `proxy.php` ожидает его ответа и при получении тут же переписывает ответ в соединении между браузером и прокси. После этого соединение по сокету закрывается, и скрипт возвращается в режим ожидания новых запросов от браузера.

Вот и все. Теперь собственный прокси сервер помогает нам модифицировать POST-запросы на лету!

Заключение

Скрипт `proxy.php` не обрабатывает запросы, по защищенному протоколу HTTPS, но это легко исправить, добавив поддержку HTTP-заголовка CONNECT. Возможно, я доработаю `proxy.php` и напишу об этом в ближайшее время. Однако, если вы не хотите писать собственный прокси-сервер, рекомендую взглянуть на готовые решения SPIKE Proxy и Paros Proxy:

http://www.immunitysec.com/spikeproxy_downloads.html

http://www.proofsecure.com/download.shtml

Nitesh Dhanjani является главным консультантом партнерства с ограниченной ответственностью Ernst & Young Advanced Security Center, где готовит обзоры по вопросам атак и проникновений. Он представляет книгу «Использование и расширение средств атак и проникновений с открытым кодом» на OSCON 2004.

Оригинал статьи и исходные коды находятся по адресу:
http://www.onlamp.com/pub/a/php/2004/01/22/php_proxy.html

Также исходные коды для статьи можно скачать с:
http://mag.phpclub.net/apr2004/proxy.zip

Написание языкового фильтра

Автор:
Джефри Джонс

Перевод:
Александр Ширяев

Сегодня мы рассмотрим написание собственного языкового фильтра. В общем-то, в этом нет ничего сложного, и вам не потребуется много времени, чтобы разобраться, что к чему. Итак, приступим к делу.

Зачем вообще нужен языковой фильтр? Слишком много раз мне встречались сайты, на которых пользователь мог совершенно свободно и бесконтрольно оставлять сообщения. Вам бы понравилось, если бы поисковики отображали ваш сайт лишь по брани и непристойностям? Мне — нет. Каждому, кто хочет фильтровать информацию начиная с источника, нужен языковой фильтр.

Я познакомлю вас с азами написания функции по фильтрации непристойностей. Существует много различных способов, и я расскажу вам про два из них. Первый связан с обычными массивами, а второй будет использовать внешний текстовый фильтр либо базу данных, наполненную бранью для сверки. Мне оба метода по душе.

Языковой фильтр, основанный на массивах

```
<?php
function language_filter($string) {
        $obscenities = array("curse","word"," foul ","language");
        foreach ($obscenities as $curse_word) {
        if (stristr(trim($string),$curse_word)) {
                $length = strlen($curse_word);
                for ($i = 1; $i <= $length; $i++) {
                $stars .= "*";
                }
                $string = eregi_replace($curse_word,$stars,trim($string));
                $stars = "";
        }
        }
return $string;
}
?>
```

Листинг 1

Вышеприведенный код полнофункционален. Достаточно компактный, но очень мощный. Давайте разберемся, что тут к чему. Сперва даем нашей функции произвольное имя — я предпочитаю `language_filter()`. Затем передаем ей все аргументы, которые нам понадобятся. В данном случае нам нужна строка, которую требуется очистить, а в итоге — переменная `$string`.

Ради приличия я не стал заполнять массив пошлостями. Тем не менее, смысл в том, чтобы создать типичный массив, содержащий слова, которые мы не хотим отображать на своем сайте. Итак, создаете массив, помещаете в него слова (заключаете их в кавычки и разделяете запятыми). Вроде, все просто. После настройки массива запускается цикл, пропускающий строку через каждое из ругательств, находящееся в массиве.

Таким образом мы проверяем, содержится ли ругательство в строке, переданной функции. Если в строке обнаруживается плохое слово, то мы переходим к рутинной цензуре. Узнаем длину этого слова и применяем цикл `for()` для замены каждой буквы на звездочку.

После того, как заменили каждую букву, надо заменить слово в строке. Для этого мы используем встроенную функцию `eregi_replace()`. Она идентична `ereg_replace()`, но с небольшим отличием — она регистронезависима. Одним словом, она заменяет ругательство цензурным вариантом, состоящим из звездочек.

После того, как наша цензура сработала по отношению к какому-то слову, мы очищаем переменную `$stars` и начинаем все сначала. Цикл затронет каждое слово из массива, пока не пройдет весь список. После завершения работы цикла с каждым словом будет возвращена новая форматированная строка.

ПРИМЕЧАНИЕ: Заметьте, что некоторые слова в массиве ограничены пробелами. Это не ошибка. Некоторые ругательные слова иногда входят в состав приличных слов. На этом примере мы хотим запретить слово foul целиком, но не слова типа foulplay (не святое, но используемое не в качестве ругательства). Окаймляющие пробелы нужны для того, чтобы заблокировать слово целиком. С этим придется поэкспериментировать.

Языковой фильтр на текстовых файлах

```
<?php
function language_filter($string) {
        $obscenities = @file("path/to/your/file/foul_language.txt");
        foreach ($obscenities as $curse_word) {
        if (stristr(trim($string),$curse_word)) {
                $length = strlen($curse_word);
                for ($i = 1; $i <= $length; $i++) {
                $stars .= "*";
                }
                $string = eregi_replace($curse_word,$stars,trim($string));
                $stars = "";
        }
        }
return $string;
}
?>
```

Листинг 2

Этот код почти такой же, как на предыдущей странице, за исключением того, что мы объявляем наши ругательства в массиве, взяв их из текстового файла.

В чем же разница? Да ни в чем, кроме опрятности кода. Если бы массив был большой, код с ним выглядел бы громоздким и неуклюжим. Если вы используете этот метод, то необходимо убедиться, что слова в файле разделены переносом строки. И не урезайте слова — они могут лишиться необходимых пробелов.

Вызов функции

```php
<?php
$string = "Curse words are not always foul in their language.";
print language_filter($string);
//Вернет: ***** ****s are not always **** in their ********.
?>
```

Листинг 3

По возврату переменной из функции мы можем вывести ее на экран либо сохранить в другую переменную для последующего использования. Оба варианта приемлемы.

Итак, работа сделана. Языковой фильтр — это несложно, но ему уделяют очень много внимания. Есть множество способов его реализации. Метод, описанный мною, наиболее прост, его я использую в каждой своей работе. Надеюсь он пригодится и вам!

Оригинал статьи находится по адресу: http://www.phpfreaks.com/tutorials/122/0.php

Абстрактный доступ к БД с помощью ADODB

Автор:
Максим Матюхин

Для начала скажу что статья рассчитана на программистов, имеющих опыт работы с СУБД, а не на начинающих пхпеишников. Я предполагаю, что вы знакомы с PHP, ОПП, SQL и имеете опыт разработки web-приложений. ADODB — это абстрактный класс доступа к базам данных, написанный на PHP.

Предположим, вы написали скрипт под mysql. И тут заказчик говорит вам, что хостинг меняется, и там есть только PostgreSQL. Если вы не использовали класс абстрактного доступа к базам данных, то вам пришлось бы:

- заменить весь код работы с mysql на postgresql;
- переписать SQL-запросы (так как есть отличия).

Если бы Вы использовали абстрактный класс доступа к БД — Вам, скорее всего, не пришлось бы менять php-код (только в одном месте указать что используется PostgreSQL). Менять SQL-запросы все равно придется, но и это не всегда необходимо.

Я намеренно в этом описании использовал фразу «абстрактный класс доступа к БД», поскольку ADODB — не единственный подобный класс. Наиболее известные конкуренты:

- Pear::DB;
- Pear::MDB.

Насколько я знаю, другие классы имеют слабую функциональность, хотя должен признать, работают быстрее. Многие пишут такие классы сами, но я не сторонник изобретения велосипедов.

Противники таких массивных классов, как ADODB или Pear::DB утверждают, что их использование плохо сказывается на производительности. Да, производительность падает, и это вполне логично. Но:

- Скорость не часто является самым важным фактором (мало кто из вас пишет сайты с очень большой нагрузкой, на которых бы снижение производительности стало критичным);
- Использование таких классов повышает производительность программиста;
- При использовании программного обеспечения (ПО) типа phpAccelerator падение производительности будет не таким заметным (и я не поверю, что популярные сайты не имеют возможности использовать такое ПО);
- Разработчики ADODB написали php-extension, который ускоряет работу класса (но работать можно и без него).

От себя могу добавить, что многие из написанных мною сайтов используют ADODB и проблем с производительностью не имеют.

Установка

Здесь все просто. Скачайте с http://php.weblogs.com/adodb архив и распакуйте его (например в папку ./adodb). Все, класс готов к использованию.

Можете еще скачать и php-extension, но я его использовать не пробовал. Чтобы использовать этот класс, вам необходимо включить (include) файл `./adodb/adodb.inc.php`.

Использование

Первый пример приведен в листинге 1.

```php
<?php
// подключаем класс
include_once("adodb/adodb.inc.php");

// указываем тип БД
$conn = &ADONewConnection('mysql');
// соединяемся с БД
$conn->Connect('localhost', 'root', 'password', 'scripts');
// режим отладки &#151; включен (true)
$conn->debug = true;
$conn->setFetchMode(ADODB_FETCH_ASSOC);
?>
```

Листинг 1

Данный пример демонстрирует подключение к БД. В строке листинга 1

```php
<?php $conn = &ADONewConnection('mysql'); ?>
```

создается объект соединения с базой данных. Именно через поля и методы данного объекта и будет в дальнейшем вестись работа с базой данных.

Что касается режима отладки, то по умолчанию он выключен. При включенном режиме отладки на экран браузера будут выводиться SQL-запросы и тексты ошибок (если такие были). Очень упрощает процесс написания и отладки скриптов.

Метод `$conn->setFetchMode()` — указывает, каким образом данные о записях будут записаны в массив — будет ли это ассоциативный массив, или простой нумерованный или и тот и другой. Необходимо установить одно из значений (0, 1, 2, 3). Для пояснений приведу код из исходников adodb:

```php
<?php define('ADODB_FETCH_DEFAULT',0);
        define('ADODB_FETCH_NUM',1);
        define('ADODB_FETCH_ASSOC',2);
        define('ADODB_FETCH_BOTH',3);
?>
```

Судя по исходникам ADODB_FETCH_DEFAULT == ADODB_FETCH_BOTH

Теперь сделаем запрос к БД:

```
<?php
// делаем запрос к БД
$res = $conn->Execute("SELECT id, title, description FROM tab");
// если по запросу найдены записи в таблице
if ($res && $res->RecordCount() > 0) {
        // выводим эти записи в цикле
        while (!$res->EOF) {
        echo "ID = ".$res->fields['id']."\n";
        echo "title = ".$res->fields['title']."\n";
        echo "description".$res->fields['description'];
        // переходим к следующей записи
        $res->MoveNext();
        }
}
?>
```

Листинг 2

Это простейший пример запроса к БД.

Метод `$conn->Execute()` выполняет запрос к базе данных и возвращает множество записей (recordset). Множество записей (recordset) в ADODB является отдельным объектом, который имеет свои поля и методы для работы с полученными записями. Некоторые из них использованы в данном примере.

- `$res->EOF` — равен true, если обработаны все записи множества
- `$res->fields` — хранит ассоциативный массив значений текущей записи
- `$res->RecordCount()` — возвращает количество строк, полученных в ходе выполнения запроса
- `$res->MoveNext()` — переходит к следующей записи (в массив `$res->fields` будет занесена следующая запись множества)

Метод `$conn->Execute()` — может быть использован для любых запросов:

```
<?php
// делаем вставку строки
$conn->Execute("INSERT INTO tab(name, value) VALUES ('name', 'ha ha ha')");
// получаем идентификатор вставки
// аналог mysql_insert_id();
$id = $conn->Insert_ID();

$conn->Execute("DELETE FROM tab WHERE id = ".$id);
?>
```

Листинг 3

Опишу еще некоторые полезные методы класса AdoConnection:

- `$conn->getRow($sql)` — возвратит массив со значениями первой записи из всего множества найденных записей;

- `$conn->getAll($sql)` — возвратит 2-мерный массив со всеми найденными записями.

Практические примеры

Рассмотрим **постраничный вывод и ограничение** SELECT-запросов. Вообще-то не все базы данных умеют делать запросы типа: `SELECT * FROM tab LIMIT 0, 10`, а все те, которые умеют, делают это по-разному:

MySQL:

```
SELECT * FROM tab LIMIT 0, 10
```

PostgreSQL:

```
SELECT * FROM tab OFFSET 0, LIMIT 10
```

FireBird:

```
SELECT FIRST 10 SKIP 0 * FROM tab
```

Класс adodb сам может делать ограниченные выборки, составляя правильные SQL-запросы под указанную БД, поддерживающую лимитированные SELECT-запросы

```
<? $res = $conn->SelectLimit("SELECT * FROM tab", 10, 0); ?>
```

Метод `$conn->SelectLimit()` сам построит правильный SQL-запрос. На основе этого метода в ADODB работают функции для постраничной выборки:

```
<?php
// определяем текущую страницу
$start = max(1, intval($_GET['start']));
// количество записей на странице
$rows_per_page = 10;
$res = $conn->PageExecute("SELECT * FROM tab", $rows_per_page, $start);

// получаем найденное количество записей
$records_amount = $res->MaxRecordCount();
?>
```

Метод `$conn->PageExecute()`, кроме простого LIMIT-запроса, делает автоматически еще и запрос типа: `SELECT COUNT(*) FROM tab`

Таким образом он сам узнает, сколько всего по данному запросу найдено строк. Это количество можно узнать с помощью метода: `$res->MaxRecordCount();`

Также для управления постраничным выводом есть следующие методы:

- `$res->AbsolutePage()` — возвращает текущую страницу
- `$res->AtFirstPage()` — возвращает true, если текущая страница — первая
- `$res->AtLastPage()` — возвращает true, если текущая страница — последняя

- `$res->LastPageNo()` — возвращает номер последней страницы

Теперь перейдем к **генерации INSERT/UPDATE запросов**. Для начала пример:

```
<?php
// пример генерации INSERT-запроса

// массив, который нужно вставить в таблицу
$frm = array("field1"=>"value1", "field2"=>"value2");
// делаем пустой запрос
$res = $conn->Execute("SELECT * FROM tab WHERE id = -1");
// формируем SQL-запрос
$sql = $conn->GetInsertSQL($res, $frm);
// выполняем запрос
$conn->Execute($sql)

// пример генерации UPDATE-запроса

// получаем данные о строке, которую нужно обновить
$res = $conn->Execute("SELECT * FROM tab WHERE id = 17");
$sql = $conn->GetUpdateSQL($res, $frm);
// выполняем запрос
$conn->Execute($sql)
?>
```

Листинг 4

Так вот, идея в том, чтобы все данные, которые нужно вставить, записать в ассоциативный массив. Сделать запрос к БД, чтобы получить имена полей таблицы и сконструировать SQL-запрос по этим данным.

Уверен, что будет много противников такого метода (мол, лишний SQL-запрос к БД делать), но мне эти функции кажутся очень удобными.

А теперь перейдем к **работе с транзакциями**. Вот пример из мануала:

```
<?php
        $conn->StartTrans();
        $conn->Execute("update table1 set val=$val1 where id=$id");
        $conn->Execute("update table2 set val=$val2 where id=$id");
        $conn->CompleteTrans();
?>
```

Метод `$conn->CompleteTrans();` сам проверит, были ли ошибки, и если так , сделает откат.

ADODB имеет еще и другие функции для работы с транзакциями, но они устарели, и разработчики ADODB рекомендуют использовать этот вариант.

Далее мы рассмотрим **последовательности**. Часто при работе с таблицами каждой записи нужно присвоить уникальный идентификатор, который потом используется в качестве первичного ключа. Но не все СУБД поддерживают такую возможность. ADODB эмулирует эту возможность почти для всех СУБД.

На практике это выглядит примерно так:

```
<?php
    $uid = $conn->GenID('site_users');
    $conn->Execute("INSERT INTO site_users(uid, login, password) VALUES
    (".$uid.", '$login', '$password')");
?>
```

Метод `$conn->GenID()` создает последовательность site_users (если она до этого не была создана) и возвращает значение на единицу больше чем текущее значение последовательности.

ADODB поддерживает **серверное кеширование запросов**. Суть в том, что при первом выполнении запроса его результаты заносятся в кеш-файл. При последующем таком же запросе (если кеш-файл не устарел) данные будут браться из файла.

Честно говоря, мне не нравится метод, которым они производят кеширование (по-моему они сделали его слишком уж универсальным) и предпочитаю делать кеширование своими руками.

Если вас все-таки интересует кеширование, то работает оно так:

```
<?php
        $ADODB_CACHE_DIR = '/tmp/ADODB_cache';
        $rs = $conn->CacheExecute('SELECT * FROM tab');
?>
```

По умолчанию время жизни кеш-файлов — 1 час. Это время можно изменить 2-мя путями:

```
<?
$conn->cacheSecs = 24*3600 // 24 часа
$rs = $conn->CacheExecute('SELECT * FROM tab');

        // или так:
        // время жизни кеша может задаваться первым параметром
        // метода CacheExecute
        $rs = $conn->CacheExecute(24*3500, 'SELECT * FROM tab');
?>
```

Наверное, вы видели, что на некоторых сайтах выводится статистика запросов: *«Страница сгенерирована за 0.0016 секунд. Запросов к базе данных - 12»*

Как вычисляется время генерирования страницы - к данной статье не относится, а вот посчитать количество запросов к БД (а также посчитать количество запросов, взятых из кеша) ADODB позволяет (см. листинг 5).

Функции, указанные в листинге 5, вызываются до запроса, поэтому вы можете с их помощью переписать SQL-запрос.

```php
<?php
// пример взят из мануала
function CountExecs($conn, $sql, $inputarray) {
      global $EXECS;
      $EXECS++;
}

function CountCachedExecs($conn, $secs2cache, $sql, $inputarray) {
        global $CACHED;
        $CACHED++;
}

$conn = NewADOConnection('mysql');
$conn->Connect(...);
$conn->fnExecute = 'CountExecs';
$conn->fnCacheExecute = 'CountCachedExecs';

...
// выводим статистику
echo "Всего запросов к базе данных: <b>".$EXECS+$CACHED."</b><br />";
echo "Из них взято из кеша : <b>".$CACHED."</b>";
?>
```

Листинг 5

Я являюсь фанатом как **adodb,** так **и** репозитария **PEAR**.

К сожалению, основным классом работы с базами данных в PEAR является PEAR::DB, и многие PEAR-классы используют его. Что же делать любителям adodb?

Во-первых, если хорошо присмотреться, то классов, использующих PEAR::DB не так уж и много. У меня почти весь pear-репозитарий на компьютере, и там pear::DB используют лишь

- `DB::NestedSet`
- `DB::DataObject`
- `DB::Pager`
- `DB::QueryTool`
- `HTML::Select`
- `XML::sql2xml`
- `Auth`
- `Cache`
- Log
- `LiveUser`
- `Mail::Queue`
- `Translation`

Во-вторых, многие классы использую «контейнеры», и для этих классов можно написать контейнер, использующий ADODB (как писать контейнеры, смотрите на примере контейнеров pear::DB указанных классов).

В-третьих, ADODB имеет файл `adodb-pear.inc.php`, который является эмуляцией класса PEAR::DB, и остальные классы можно подогнать под работу с adodb с минимальными телодвижениями (часто достаточно в тексте класса строку `require_once ('DB.php');` заменить на `require_once('adodb-pear.inc.php');` но так бывает не всегда).

Так что ADODB можно успешно применять с pear-классами. Приведу пример использования adodb с классом pear::XML::sql2xml. Для тех, кто не в курсе: этот класс трансформирует результат запроса (SELECT) к БД в XML-строку:

```
<?php
require_once("adodb/adodb-pear.inc.php");
require_once("XML/sql2xml.php");

$db = DB::connect("mysql://root@localhost/lot");
$sql2xml = new xml_sql2xml();
$result = $db->getAll("select * from lot_sessions");
$xmlstring = $sql2xml->getXML($result);
?>
```

Те, кто уже имеют опыт работы с XML_sql2xml наверное чаще применяют код, который предлагает автор класса XML_sql2xml:

```
<?php
require_once("XML/sql2xml.php");
$sql2xml = new xml_sql2xml("mysql://root@localhost/lot");
$sql2xml->Add("select * from lot_sessions");
$xmlstring = $sql2xml->getXML();
?>
```

и

```
<?php
$db = DB::connect("mysql://root@localhost/lot");
$sql2xml = new xml_sql2xml();
$result = $db->query("select * from lot_sessions");
$xmlstring = $sql2xml->getXML($result);
?>
```

Оба эти примера не сработают и нужно будет править класс XML_sql2xml.

Заключение

Поскольку статья носит ознакомительный характер, многое осталось «за кадром». Я не пытался описать все классы, поля и методы — для этого есть официальная документация. Также я не описывал функциональные возможности, которые не использовал на практике:

- хранение сессий в БД (в том числе и зашифрованных сессий)
- работа с хранимыми процедурами
- работа с БД, находящейся на удаленном сервере

- словари (позволяют программно создавать базы данных и таблицы)

Ссылки по теме

- Adodb (http://php.weblogs.com/adodb) — официальный сайт adodb
- Pear::DB (http://pear.php.net/DB) — отсюда можете скачать pear::DB
- Pear::MDB (http://pear.php.net/MDB) — а отсюда можете скачать pear::MDB
- Использование PEAR для доступа к базе данных (http://detail.phpclub.net/article/2002-11-01) — статья по pear::DB
- Объектно-ориентированное программирование. Часть 3. Абстрактные классы БД (http://detail.phpclub.net/article/2001-03-13) — обзор классов для доступа к БД

Статья предоставлена сайтом: http://detail.phpclub.net

Люди

История успеха phpBB[1]

На сегодняшний день phpBB является одним из самых популярных скриптов веб-конференций. Тысячи сайтов создают сообщества, используя именно phpBB. В отличие от многих других подобных скриптов, phpBB является бесплатным и распространяется по лицензии GPL. Это один из примеров успешного проекта Open Source, который функционирует не хуже, а может даже и лучше своих платных аналогов.

Интервью брал:
Эмир Мусабейсик

Перевод:
nw

Помимо впечатляющих возможностей и хорошей производительности, phpBB имеет еще одно огромное преимущество — отзывчивое сообщество. Продукт приобрел многих сторонников, которые гарантируют квалифицированную помощь каждому нуждающемуся в ней.

Скоро должен состояться релиз новой версии скрипта — 2.2. В phpBB 2.2 будет улучшена функциональность администраторской части и добавлены некоторые дополнительные возможности, что только укрепит его позиции. Одновременно с релизом новой версии, команда разработчиков начнет работу над переводом ядра под портальные функции. Этот портальный проект поможет phpBB стать еще популярнее среди сообщества вебмастеров.

Я начал работу над проектом летом 2000 года с возвышенной целью — снабдить форумом сайт моей жены

Недавно мне выпала честь взять интервью у Джеймса Эткинсона (theFinn), основателя и менеджера проекта phpBB.

Эмир Мусабейсик (EM): Как и почему была начата работа над проектом phpBB? Каким образом подобралась команда разработчиков? Думали ли вы, что ваш продукт станет таким популярным?

Джеймс Эткинсон (JA): Я начал работу над проектом летом 2000 года с возвышенной целью — снабдить форумом сайт моей жены. На тот момент реальным, известным мне, программным обеспечением для веб-конференций были UBB и Phorum. UBB был слишком дорогим для бедного студента колледжа, а стиль Phorum мне совершенно не нравился. Мне был нужен хороший, открытый проект, поэтому было решено начать свою разработку. Я скопировал стиль UBB и стал распространять скрипт под GPL лицензией.

Команда стала собираться после того, как я оставил сообщение на DevShed с просьбой потестировать мой форум. Вскоре после этого я открыл проект на sourceforge.net, и люди стали присылать мне свои варианты кода и запросы по поводу присоединения к проекту. С тех пор процесс начал набирать обороты как снежный ком.

Что касается популярности продукта, то она стала для меня настоящим шоком. Я начинал работу над проектом без особых претензий и стремлений.

1 Оригинальное интервью состоялось 13 февраля 2004 года.

Open Source

EM: phpBB является одним из нескольких Open Source проектов веб-конференций. Почему вы выбрали именно Open Source?

JA: Я даже не рассматривал других вариантов. Я и все члены команды phpBB верят в идеалы свободного программного обеспечения.

EM: Как вы считаете, какие преимущества перед коммерческими решениями приобрел phpBB, используя модель Open Source?

JA: Самое большое наше приобретение от Open Source — это то, что мы имеем большее сообщество пользователей, чем другие платные веб-конференции. Для вхождения в phpBB сообщество не нужно платить никаких вступительных взносов, любой вебмастер может при желании поставить phpBB на свой сайт. Постепенно, накапливая опыт, многие люди начинают ощущать, что могут дать что-то взамен, и присоединяются к сообществу, чтобы помогать другим пользователям. И это одна из самых замечательных вещей в Open Source — люди помогают людям.

Постепенно, накапливая опыт, многие люди начинают ощущать, что могут дать что-то взамен, и присоединяются к сообществу, чтобы помогать другим пользователям. И это одна из самых замечательных вещей в Open Source — люди помогают людям

EM: Можете ли вы дать совет разработчикам, которые только начинают работу над своими проектами?

JA: Самое главное – это прислушиваться к своим пользователям. Именно они ведут ваш продукт вперед и помогают в его поддержке. Во-вторых, вам следует придерживаться своих идеалов и не изменять им. Одна из причин, по которым phpBB стал таким популярным, заключается в том, что он оставался стабильным. Мы никогда не меняли свое имя, не меняли своих приоритетов и никогда не меняли лицензию на продукт. Мы придерживались тех идеалов, которые были заложены в продукт с момента его основания. Такой подход позволил нашему сообществу считать phpBB крепким и стабильным.

EM: Во многом, своим успехом вы обязаны сообществу пользователей. Создается впечатление, что вы умеете создавать не только хорошие скрипты, но и можете дать совет вебмастерам по созданию сообществ.

JA: Одной из самых больших ошибок начинающих становится их нежелание начинать с малого. Я видел много веб-конференций с 10-12 форумами при двадцати сообщениях на всю конференцию. Это плохой путь для старта. Начните с двух-трех форумов и позвольте дискуссии завязаться, а уже потом расширяйтесь по мере необходимости.

Также вам нужна приманка — то, что соберет людей вокруг вашей веб-конференции и станет отправной точкой для дискуссий. SitePoint великолепный тому пример. Это большой сайт с массой материалов, которые приводят посетителей в форумы.

Разработка в деталях

EM: Не могли бы вы описать, как члены команды взаимодействуют между собой? Как вы решаете, что пришло время добавить новую возможность? Кто отвечает за какие аспекты проекта?

JA: Мы общаемся посредством форумов на phpBB.com. Там существует служебный форум «За кулисами», где мы планируем работу по сайту и форуму.

Идеи по новым возможностям поступают напрямую от пользователей. На sourceforge.net у нас есть список пожеланий, и команда разработчиков (под руководством psoTFX) решает, как и в какие сроки реализовывать те или иные пожелания. Наш цикл разработки достаточно закрыт, и мы стараемся не вносить код, полученный не от членов команды, однако с большой охотой принимаем пожелания и идеи от пользователей.

EM: Производительность является большой головной болью для администраторов форумов. Как phpBB справляется с большой конкуренцией на этом направлении? Как вам удается балансировать между внедрением новых возможностей и всеобщей производительностью phpBB?

JA: Команда разработчиков всегда принимает во внимание общий эффект от внедрения каждой новой возможности. Если какая-либо из возможностей требует много системных ресурсов, то она либо переписывается (как это не раз случалось, например, с нашим поиском и системой разграничения прав), либо удаляется совсем.

Мы считаем, что производительность форума не должна приноситься в жертву некоторым «красивым» функциям. phpBB был изначально ориентирован на стабильность, безопасность и производительность. Такая ориентация будет продолжена в последующих разработках и релиз phpBB 2.2 демонстрирует наращивание производительности при наличии новой функциональности. Эта производительность будет особенно заметна в работе больших форумов. Мы многому научились у пользователей, которые администрируют большие форумы.

EM: Безопасность тоже является важным фактором для phpBB, который и так кажется достаточно безопасным. Что в этой сфере вы планируете улучшить в будущем?

JA: Над безопасностью мы серьезно работали в первых версиях (v1.x). Тогда мы старались убедиться в том, что все дыры залатаны. Теперь каждая новая возможность тщательно проверяется с точки зрения безопасности на всех этапах разработки. Наша команда разработчиков следит за всеми данными, входящими в скрипт извне, с целью избежать нарушения целостности SQL-запросов (так называемых SQL-инъекций). В целом, в последних версиях мы уделяем много внимания проверке данных, вводимых в формы.

Безопасность является одним из главных стремлений phpBB.

EM: В основном phpBB работает с MySQL, но разработан также и абстрактный класс для других СУБД. Можете ли вы привести примеры успешной работы с другими базами данных?

JA: Я люблю говорить: «О да, x.com использует phpBB с Postgres, и он работает великолепно», — но чаще всего мы слышим именно о работе с MySQL. Это самая популярная СУБД для использования с нашим форумом, и большинство пользователей работают именно с MySQL.

Наверное нам следует уделять больше внимания именно этой базе данных.

Внутри phpBB v2.2

EM: В версии 2.0 phpBB был тотально переписан, похоже, это произошло и с версией 2.2. Каковы основные изменения?

JA: Полностью мы ничего не переписывали. Мы начали работу над 2.0 с «причесывания». В 2.2 была переписана система разграничения прав, функционал модерирования, некоторые дополнения к пользовательской контрольной панели и много дополнений к административной части.

Основные цели остаются теми же что и всегда — делать самый классный форум, какой только можем!

EM: Назовите основные новые функции в phpBB 2.2.

JA: Основные новые функции коснулись панели управления пользователя. Мы пошли по пути большей модульности в управлении данными пользователя, добавили новшества в систему подписки на форумы и топики, а также списки «друзей и недоброжелателей».

Мы также включили обновленную контрольную панель модератора, которая более удобна для модерирования разветвленных форумов и топиков. Это, например, такие функции, как объединение топиков и сообщение о топике модератору. Также была добавлена возможность прилагать к сообщениям файлы. Эта функция была адаптирована из мода Эйсида Берна, который теперь присоединился к нашей команде разработчиков.

Тонны новых вещей были добавлены в администраторскую секцию: редактирование шаблона онлайн, новая система бана, а также возможность, позволяющая поисковым роботам Google более эффективно индексировать сообщения в форумах, и еще много-много новых функций.

EM: Вскоре после разработки phpBB 2.2 вы приступаете к разработке скрипта портала. Почему вы решили взяться за это?

JA: Это то, что решила сделать наша команда. BartVB, разработчик, давно входящий в нашу команду, подал эту идею и возглавил работу в этом направлении. Честно говоря, я принимал небольшое участие в данном проекте и поэтому не могу подробно ответить на ваш вопрос.

EM: Что станет основным преимуществом вашего портала перед другими аналогами, существующими на рынке?

JA: Главное преимущество — в знакомом интерфейсе phpBB. Веб-разработчики, которые хотят быстро организовать сайт, основанный на веб-конференциях, смогут это сделать достаточно уверенно. Главная идея — сделать продукт полностью модульным и отлично расширяемым.

EM: Каковы основные цели phpBB?

JA: Основные цели остаются теми же что и всегда — делать самый классный форум, какой только можем! Мы идем по тому направлению, по которому нас ведут пользователи. Именно они двигают проект вперед!

EM: Если оглянуться назад, есть ли что-нибудь такое, о чем вы сожалеете?

JA: Орфографические ошибки в версии 1.0, на которые обратили внимание будущие соратники. Но вообще-то я не могу назвать ничего такого. Это был великолепный путь, который уже пройден, и я смотрю вперед, на будущее нашего проекта!

Ссылки к статье: http://www.phpbb.com

Оригинал интервью находится по адресу:
http://www.sitepoint.com/article/james-atkinson-founder-phpbb

Команда этого выпуска

Авторы и переводчики

- Александр Ширяев [dallas];
- Игорь Луканин [lucas];
- Эмиль Хасанов [Hem], г. Набережные Челны;
- Андрей Олищук [nw], г. Коломна;

Авторы, чьи статьи были предоставлены сайтами

- Сергей Соколов;
- Макс Матюхин;

Редакционная коллегия, корректоры

- Александр Смирнов [PHPclub];
- Елена Тесля [Lenka];
- Александр Войцеховский [young];
- Антон Чаплыгин;
- Андрей Олищук [nw];
- Александр Ширяев [dallas];
- Дмитрий Попов;
- http://phpclub.ru

Подготовка обложки, верстка

- Антон Чаплыгин;
- Зенькович Денис;
- Максим Пилипенко [krisha];
- Олищук Андрей [nw].

Спасибо всем участникам проекта!

http://mag.phpclub.net

project@yugovostok.ru