Обзор Pascal Server Pages из первых рук

Владимир Сибиров a.k.a. Trustmaster
http://freepascal.ru/

25 января 2006 г.

Содержание

Введение

Что такое и с чем это едят

Pascal Server Pages (сокращенно PSP) — кроссплатформенная библиотека, облегчающая CGI веб-программирование на Free Pascal. Ее цель — внести некоторые стандарты в столь пестрый мир CGI программ и обеспечить программиста всем необходимым ему для создания веб-приложений, а также дополнительными средствами, которые могут в этом пригодиться.

Проект стартовал в 2003 году, как это обычно бывает, «just for fun» (забавы ради, шутки для), и через некоторое время был опубликован в Сети. С тех пор он вырос из простого модуля в полноценную библиотеку, а контингент пользователей включает не только тех, кто учил Паскаль в школе/колледже и теперь хочет применять его для веб, но и опытных программистов, не желающих расставаться с любимым языком при решении любых задач, для которых производительность имеет значение.

Что есть **PSP глазами PHP/Perl/ASP/JSP/и т.п.-программиста**? Это не скриптовый интепретируемый язык. С одной стороны, здесь существует жесткая типизация данных, традиционная структура программ и необходимость в компиляции под целевую платформу (под «целевой платформой» подразумевается архитектура и операционная система того сервера, на котором приложения будут работать). С другой стороны, здесь отсутствует необходимость в интепретаторе, и ничто не мешает нам пользоваться наработанным кодом или теми функциями, которые в обычном веб-программировании недоступны. Как вам, к примеру, управление питанием компьютера и окнами Windows удаленно через веб-интерфейс? А между тем, такое приложение на PSP написано. Или веб-интерфейс к аудиоплееру или вещательной станции? И такой проект с использованием PSP существует. Что еще мы получаем «в упаковке»? Производительность исполнимого кода. Она заведомо выше, чем у интепретируемого

скрипта или транслируемого байт-кода. Насколько выше? Как показывает тест, проведенный WrenSoft (CGI vs. PHP vs. ASP), исполнимые CGI-программы работают почти в 10 раз быстрее, чем ASP и PHP4. Даже если учитывать то, что с переходом на многопоточную структуру в PHP 5.1, производительность PHP5 возросла в 2,5 раза, то перевес на стороне binCGI остается величиной в 4 раза. Но за производительность, конечно, приходится платить. Мы не гарантируем вам, что использование PSP будет легкой прогулкой. Будьте готовы созерцать время от времени различные «Runtime Error» и «505 Internal Server Error» во время отладки (что, впрочем, превращается из кошмара в легкую игру средствами даже стандартного FPC). Но если вы не ищите сверхлегких путей, и язык Pascal вам близок — Pascal Server Pages для вас.

Что есть **PSP глазами FPC-программиста**? Не секрет, что 90% всех бинарных CGI программ пишутся с чистого листа без использования сторонних библиотек. Каждый готов изобрести свой велосипед. На это уходит, порой, очень много времени. PSP позволяет сконцентрироваться на решении поставленной задачи, не вдаваясь во внутренние тонкости HTTP/CGI и не тратя время на реализацию и отладку часто используемых алгоритмов. Еще это возможность дать своей программе лицо в веб, не изменяя ее основного кода. Наконец, это то, что позволяет вам сделать интерактивный сайт во Free Pascal, без необходимости изучать PHP/Perl/JSP/и т.п.

Данный проект — не единственный PSP в Сети. И дело не в Sony Play Station Pro, не в замечательных аудиоплагинах фирмы PSP AudioWare, и даже не в Python Server Pages. Существуют и другие реализации Серверных Страниц Паскаль. Одна из них, имеющая в точности такое же имя (Nemesis Project: Pascal Server Pages) есть не что иное, как небольшое веб-расширение для интепретатора языка Паскаль Nemesis, написанного на Delphi. Там код встраивается непосредственно в HTML (стиль PHP и ASP), но количество веб-ориентированных возможностей очень мало. Другой интересный проект — Borland WebSnap, портированием которого в Free Pascal занимается один из ключевых разработчиков FPC — Michael Van Canneyt. Он основан на полной визуализации процесса с помощью компонентов. Мы в Pascal Server Pages Development Team считаем, что нет смысла использовать «жирные» компоненты там, где в них нет особой необходимости и там, где важна компактность и производительность. И, разумеется, существует множество различный юнитов для работы с CGI в FPC — еще одна иллюстрация к изобретению велосипедов.

Перед стартом

Что нужно знать, прежде чем вы начнете использовать PSP

Для начала все-таки стоит узнать, как работают CGI-программы:

1. Пользователь в браузере вводит адрес страницы, нажимает на ссылку или же отправляет данные формы.

2. Браузер формирует HTTP-запрос и отправляет его на вебсервер.
3. Вебсервер производит анализ запроса и передает необходимые переменные, в том числе и данные формы или аргументы ссылки, CGI-программе при вызове ее на исполнение.
4. CGI-программа получает входные данные, каким-то образом их использует, выполняет необходимые действия и формирует вывод, состоящий из заголовков HTTP-ответа и его тела (чаще всего, статичного HTML-документа).
5. Вебсервер отсылает HTTP-ответ браузеру, и тот отображает результат.

Из этого следует, что каждая CGI-программа вызывается множество раз на очень короткий период времени (обычно десятитысячные доли секунды), и одновременно в памяти может находиться несколько ее экземпляров. Это немного напоминает многопроцессность/ветвление (multiprocessing, forking), но без какой-либо совместной связи между процессами (если, конечно, вы ее сами не учтете).

Как видно, сервер — важная деталь в жизни PSP-программиста. Что нам нужно и какова структура рабочих PSP-проектов? PSP поддерживается большинством вебсерверов, исключая, пожалуй IIS (*интересно, но у IIS какая-то «аллергическая» реакция на FPC во время обработки метода POST, потому что простейшие CGI-программы на Си, имевшие в точности такие же алгоритмы, работали на нем на ура*). Но в первую очередь мы ориентируемся на вебсервер Apache, как на самый распространенный в мире хостинга. В принципе, кроме вебсвервера с полноценным cgi-bin нам понадобится разве что FTP-доступ, чтобы закачать наш проект: исполнимые приложения и используемые файлы должны находиться в папке cgi-bin, а статичные страницы в htdocs/html. Конечно, неплохо иметь возможность закачивать проекты в исходниках и компилировать их уже на сервере через шелл-доступ, скажем, если написали вы его под IA-32 Windows, а на сервере используется Solaris, да и еще и с архитектурой SPARC. Этот случай, пожалуй, не имеет другого выхода. Но для компиляции и тестирования с одинаковой архитектурой есть масса возможностей: установка нескольких ОС, использование виртуальных машин и кросскомпиляция. О последней, а также уникальной возможности в PSP будет рассказано позже. Сейчас отметим, что для создания веб-приложений с PSP достаточно компилятора FPC и базовой установки вебсервера Apache на вашем компьютере.

Итак, FPC и Apache установлены. Самое время взяться за PSP. Найти последний официальный релиз, а также сопутствующие файлы можно в файловом респозитории SourceForge. Новейшие devel-версии, а также все «вкусности» доступны с SVN-респозитория OpenSVN. Полученный архив распакуйте в любую директорию. Структура каталога обычно описана в сопровождающих файлах, я не буду затрагивать это в данной статье.

Несколько слов по поводу версий. Сейчас в основном используются две ветки: 1.4 и 1.5 (к последней буквально на днях примкнет 1.6, которая будет полностью совместима с ней на уровне исходного кода). 1.4 — стабильная, но уже

закрытая на данный момент, ветка, которая привнесла в свое время огромные изменения (между версиями 1.3.3 и 1.4.0 пропасть в 8 месяцев и 90% кода). Ее отличает техническая строгость и несколько Си-подобный синтаксис. Версия 1.5 более дружественна к пользователю, она прячет некоторые детали, давая расширенную конфигурацию взамен. К тому же в ней используется популярный стиль MixedCaseNaming. Однако на данный момент у версии 1.5.0 есть 2 существенных недостатка по сравнению с 1.4.2, что объясняется ее фактическим альфа-статусом: отсутствие документации и пакетизации (packaging). Как бы то ни было, мы настоятельно рекомендуем пользователей версии 1.5.0 использовать документацию из версии 1.4, потому что она содержит массу необходимой информации, а различия между 1.4.2 и 1.5.0 столь незначительны, что перевод серьезного проекта с одной версии на другую занимает всего пару часов. Планируется, что оба недостатка будут решены уже в версии 1.5.1. Поэтому в данной статье мы будем рассматривать именно ветку 1.5.

Ну и конечно же, какое начало без «hello, world»? Давайте тогда поприветствуем того, чье имя передают в параметре ссылки "`name`" (например, "`http://domain.com/cgi-bin/hello.psp?name=Вася`"):

```
{$H+}{$MODE OBJFPC}

program hello;

// Подключаем веб-юнит
uses pwu;

begin
    // сразу выводим содержимое переменной в строке,
    // дополнительно фильруя ее от нежелательного содержимого
    WebWriteFF('Здравствуй, {$name}!');
end.
```

Вот и все! Дальше практика, практика, практика... И мы были бы особенно признательны тому, кто в процессе собственного обучения напишет Tutorial для новичков.

Основные возможности

Обзор основных возможностей библиотеки

Думаю, я уже достаточно утомил опытных программистов нудными теоретическими выкладками. Поэтому перейдем к самому вкусному — к возможностям.

Структура

Сразу стоит сказать, что Pascal Server Pages не библиотека классов. Весь код имеет функционально ориентированную основу, и только для некоторых задач, где это наиболее удобно, используются классы. И дело не в том, что нам лень было создавать сложные иерархии, и даже не в том, что мы консерваторы. На это есть 2 причины. Первая — мы не используем «жирное» ООП там, где оно не требуется и не является оптимальным; мы считаем, что на низком уровне должен быть функциональный код. Впрочем, это вытекает из второй причины: с самого начала предполагалось, что библиотека со временем будет разделена на 2 уровня: «низкий» будет помещен в динамическую библиотеку, а «высокий» останется в юнитах и будет объектно ориентирован. Поэтому в ветке 1.4 вообще не было объектно-ориентированного кода, и стиль был очень похож на Си. На данный момент PSP — все еще набор юнитов, но динамическая версия тоже существует и находится на стадии тестирования.

Конфигурация

Файл `PWU.conf` используется для конфигурации всех программ в одной директории. Я не буду описывать его опции здесь, т.к. он снабжен подробными комментариями. Во время выполнения программы также можно получать и задавать значения конфигурационных переменных, например:

```
s := GetWebConfigVar('error_reporting');
SetWebConfigVar('error_halt', 'on');
```

Кроме конфигурации есть в PSP такая вещь как RTI (RunTime Information). Эти переменные содержат внутренние флаги PSP, например:

```
s := GetRTI('SESSION_REGISTERED');
```

Собственно, CGI

Все входные данные PSP получает и обрабатывает автоматически. Вам остается только использовать их с помощью функций: `CountCgiVars` (число переменных), `IsCgiVar` (наличие такой переменной), `GetCgiVar` (доступ по имени), `FetchCgiVarName` и `FetchCgiVarValue` (доступ по индексу). В PSP под Cgi подразумеваются GET/POST переменные. Но существует также общее понятие «переменных Web», которые включают и Cgi, и Cookie, и Sess, и шаблонные макросы, которые получают с помощью функций `CountWebVars`, `IsWebVar`, `GetWebVar`, `FetchWebVarName`, `FetchWebVarValue`. Имейте ввиду, это далеко не полный список функций. А вот небольшой пример использования:

```
range := GetCgiVarAsInt('range');
if range <= CountWebVars then
```

```
for i := 0 to range - 1 do
begin
    WebWriteLn(FetchWebVarName(i) + ' = ' +
               FetchWebVarValue(i) + '<br>');
end;
```

Заголовки в PSP не следует задавать напрямую как в обычных CGI программах. Для этого есть специальные функции (`GetWebHeader`, `SetWebHeader` и т.д.), которые, к примеру, будут работать даже в конце вашей программы, если включена буферизация вывода.

```
SetWebHeader('Content-Type', 'application/xml');
```

Пользовательские сеансы

Есть 3 способа различать пользователей на сайте и хранить для каждого из них специфические данные. Первый — постоянно передавать GET/POST параметры — никогда практически не используется, ибо утомителен и неоправдан. Второй — использовние Cookies (куки, печенья, плюшки) — хранение этих данных в браузере пользователя. Третий — использование механизма сессий.

Плюшки в PSP читаются и задаются подобно другим веб-переменным: с помощью функций `GetCookie` (получить значение пременной), `SetCookie` (задать переменную), `UnsetCookie` (удалить переменную) и множества их вариаций и родственников. Главное, чтобы если вы не используете буферизацию вывода, плюшки задавались/удалялись до какого-либо вывода:

```
if lowercase(GetConfigVar('output_buffering')) = 'off' then
begin
    SetCookie('foo', 'значение');
    UnsetCookie('bar');
    WebWrite('только теперь можно что-то вывести');
end;
```

Сессии в свое время наделали много шума в PHP4. Механизм сессий хранит в cookies только идентификатор, а сами данные — на сервере. При этом сессия означает именно один пользовательский сеанс — промежуток времени, который конкретный пользователь находится на данном сайте с интервалом между перемещением по страницам не больше заданного. В PSP сессионные переменные читаются/записываются так же, как и остальные:

```
s := GetSess('foo');
SetSess('bar', 'значение');
UnsetSess('foo');
```

Регистрация новой сессии и уничтожение сессии используют cookie и потому должны проходить до какого-либо вывода, если не включена буферизация.

Загрузка файлов

Загрузка файлов на сервер — еще одна возможность, которая, разумеется, в PSP присутствует. Делается это посредством форм типа `multipart/form-data` и нескольких встроенных функций. Пример формы:

```
<form action="/cgi-bin/upload.psp" method="post"
      type="multipart/form-data">
<input type="text" name="text" value="Текстовое поле" />
<input type="file" name="file" />
<input type="submit" value="Отправить" />
</form>
```

Соответственно, `upload.pas`:

```
program upload;

uses pwu;

var ftext, fname: string;

begin
    // Текстовые и прочие поля доступны как обычно
    ftext := GetCgiVar('text');
    // Сохраняем файл
    if IsUpFile('file') then
    begin
        fname := GetUpFileName('file');
        if SaveUpFile('file', '/tmp/' + fname) then
            WebWriteLn('Файл сохранен');
    end
    else WebWriteLn('Ошибка загрузки файла');
end.
```

Отличительная особенность загрузки файлов в PSP в том, что файл загружается сначала целиком в память, а не во временный файл. Отчасти это устраняет проблемы с правами доступа, но с другой стороны — это не экономично. В будущем механизм загрузки будет переписан, и уже вы будете выбирать, каким образом загружать файл.

Окружение

Вебсервер общается с CGI-программой в основном посредством переменных окружения. В PSP с ними работать так же легко, как и со всеми остальными:

```
userAgent := GetEnvVar('USER_AGENT');
for i := 0 to CountEnvVars - 1 do
```

```
    WebWriteLn(FetchEnvVarName(i) + ' = ' +
               FetchEnvVarValue(i) + '<br>');
```

Кроме того, существует юнит `PwuEnvVar`, который дает доступ к переменным окружения в стиле ASP:

```
referer := CGIEnvVar.Referer;
```

Шаблонизация и защита

Генерировать вывод стринички простым выводом (например, `WebWriteLn`), конечно, не самый удобный способ, особенно для крупных проектов. Поэтому в PSP присутствует такая возможность, как шаблонизация. Действует она таким образом: переменные Web можно задавать динамически с помощью функции `SetWebVar`. Механизм шаблонизации знает об этих переменных и может подставлять их в вывод. Тут макросы форматирования есть нечто среднее между стилем Perl/PHP и .NET:

```
s := 'строка для |%$| подстановки';
SetWebVar('MyVar', s);
WebWriteF('Нам срочно нужна {$MyVar}!');
// Теперь посмотрите разницу
WebWriteFF('Нам срочно нужна {$MyVar}!');
```

Какая разница между этими двумя функциями? Такая, что первая отфильтрует переменную `MyVar` только если включена соотвествующая опция автофильтрации в конфигурационном файле. А вторая сделает это в любом случае. Зачем это нужно? Таким образом PSP автоматически защищает вас от Cross Site Scripting (XSS) атак при использовании шаблонов.

На самом деле, в серьезных проектах обычно существует полное разделение кода и содержимого, и в них вы вообще не встретите `WebWrite/write/print` и т.п. Для этого используются шаблоны. Работают они схожим образом: переменные задаются динамически, а парсер подставляет их значения вместо последовательностей `{$ИмяПеременной}`. Но сами шаблоны хранятся в отдельных файлах и выводятся с помощью функции `WebTemplateOut`:

```
<!-- пример template.tpl -->
<dl>
<dt>{$title}</dt>
<dd>{$description}</dd>
</dl>

// пример template.pas
SetWebVar('title', 'Заголовок');
SetWebVar('description', 'Описание');
WebTemplateOut('tpl/template.tpl', true); // включена фильтрация HTML
```

Кодирование URL, Base64 и MD5

Конечно, эти функции можно встретить и в пакетах FPC, и в FCL, однако в PSP тоже присутствует их компактная реализация, чтобы вы могли кодировать/раскодировать HTTP-URL, использовать кодирование/декодирование Base64 и хэши MD5. Простой пример:

```
s := 'Просто тестовая строка типа AnsiString';
WebWriteLn(url_encode(s) + '<br>' + base64_encode(s) + '<br>' +
           md5_string(s) + '<hr>');
WebWriteLn(url_decode(url_encode(s)) + '<br>' +
           base64_decode(base64_encode(s)));
```

Безопасный доступ к файлам

Как уже упоминалось, есть у CGI-программ одна проблема с файлами. Что произойдет, если 2 экземпляра одной программы будут одновременно записывать в один файл? Или одна будет читать, а другая записывать? Правильно — crash, boom, bang! Юнит `FileShare` позволяет решить эту проблему простым кроссплатформенным способом: при помощи умной системы файлов-флагов и сборщика мусора. Для этого достаточно вызывать его функции перед и после кода работы с файлом:

```
if file_exists('filename.ext') then
begin
    // Запись в файл
    file_mark_write('filename.ext');
    assign(fh, 'filename.ext');
    rewrite(fh);
    // Какие-то действия
    close(fh);
    file_unmark_write('filename.ext');
    // Чтение файла
    file_mark_read('filename.ext', wk);
    assign(fh, 'filename.ext');
    reset(fh);
    // Some actions
    close(fh);
    file_unmark_read('filename.ext', wk);
end;
```

GZIP внутри

В PSP встроена Pascal-версия библиотеки ZLib. Поэтому он легко использует GZIP-сжатие вкупе с буферизацией вывода, что позволяет существенно экономить трафик. Для этого нужно убедиться, что в `pwu.pp` определен макрос `GZIP_ENABLED` и в конфигурационном файле `PWU.conf` включены опции

output_buffering и output_compression. Никакие изменения в коде не нужны. Кроме того, имеется юнит GZIP, который предоставляет всего по паре функций для сжатия/разжатия строк и файлов без вникания в тонкости библиотеки ZLib.

Сетевые имена

Как известно, сам по себе юнит Sockets в FPC не может распознавать имена машин в сети. Юнит HostName в PSP позволяет легко переводить IP-адреса в доменные имена и обратно, а также разрешать их в тип TInetSockAddr.

HTTP-клиент

Довольно часто приходится получать внутри веб-приложений содержимое других страниц по HTTP. Конечно, для этого можно использовать Ararat Synapse, но в PSP имеется юнит http, который решает эту проблему стандартными средствами RTL. Он позволяет как поэтапно контролировать HTTP-диалог, так и выполнить какое-то действие «легким движением руки». К примеру, скопировать удаленный документ в локальный файл можно так:

```
if http_copy('www.server.com/path/script.php?cid=256&name=example',
             '/tmp/example.html') then
    WebWriteLn('Успешно скопировали!');
```

SMTP-клиент

Корреспонденцию отправлять приходится ничуть не реже, чем скачивать странички. И протокол SMTP здесь тоже реализован. Пример программы:

```
uses smtp;

var cp: SMTPConnection;
    mp: SMTPMessage;

begin
    // Создаем новое сообщение (от, кому, тема, текст)
    mp := smtp_message('Tester <test@localhost>',
                       'Admin <admin@localhost>',
                       'SMTP тест',
                       'Здорово если вы можете это читать!');
    // Так задаются заголовки
    smtp_set_header(mp, 'X-Priority', '2 (High)');
    // То же самое
    // smtp_put_header(mp, 'X-Priority: 2 (High)');
    // Присоединяем вложение (количество не ограничено)
```

```
    if smtp_attach(mp, 'smtp.pp', 'text/plain')
       then writeln('Вложили файл')
       else writeln('Ошибка вложения');
    // Задаем content-type тела письма
    smtp_set_text_type(mp, 'text/plain; charset=windows-1251');
    // Соединяемся с SMTP-сервером
    //   (server/server:port, username, password)
    cp := smtp_connect('freepascal.ru', 'username', 'password');
    if cp <> nil then writeln('Соединились')
                 else writeln('Ошибка соединения');
    // Отправляем (можно отправить несколько писем одно за другим)
    if smtp_send(cp, mp) then writeln('Письмо отправлено')
                         else writeln('Невозможно отправить');
    // Close connection
    smtp_close(cp);
    writeln('Отправка завершена');
end.
```

Sendmail под UNIX

На многих UNIX-серверах почта отправляется через локальный процесс SendMail. В PSP 1.5 есть и такая возможность. К примеру:

```
WebSendMail_P([High], 'mailform@youraddress.com',
                      'youraddress@yahoo.com', 'Тестируем Sendmail',
                      StrLoadFile('messageTEXT.txt'));
SetLength(AttachedFile, 1); // число присоединяемых файлов
AttachedFile[0].FileName:= 'MyAttachment.txt'; // путь к вложению
PWUEmailType:= [HTML];
WebSendMail_PA([Low], AttachedFile, 'mailform@youraddress.com',
                      'youraddress@yahoo.com', 'Тестируем вложения',
                      StrLoadFile('messageHTML.txt'));
```

Строки и регулярные выражения

Иногда стандартного набора строковых функций из RTL недостаточно. В PSP используется еще один модуль, который содержит два с половиной десятка дополнительных строковых функций (поиск, замена, удаление подстрок, разбивка, сравнение по натуральному алгоритму и т.д.) — `SubStrings`. Кроме того, некоторые часто применяемые операции со строками и текстовыми файлами (например, чтение текстового файла в массив строк) можно выполнить с помощью дополнительного модуля `StrWrap1`.

Регулярные выражения используются в веб-программировании и тут и там. PSP предоставляет вам встроенную библиотеку Perl-совместимых регулярных выражений `TRegExpr` Андрея Сорокина, доступ к которым осуществляется через юнит `regexp` в несколько PHP-подобном стиле:

```
uses regexp;

const SOURCE = 'The quick brown fox jumped over the lazy dog.';

var res: RegexpResult;
    ent: RegexpEntry;
    i: longword;

begin
    writeln(SOURCE);
    // Проверка на соответсвие шаблону
    writeln('Есть здесь лисы и собаки?');
    writeln(regexp_check('/^[\w\s]+fox[\w\s]+dog\.$/', SOURCE));
    // Считаем полные вхождения:
    writeln('Number of "the" in source: ',
            regexp_count('/the/i', SOURCE));
    // Полноценное регулярное выражение с выделением подмасок
    writeln('Как лисы и собаки описаны?');
    if regexp_match('/^the ([\s\w]+) fox.*?(\w+) dog\.$/i',
                    SOURCE, ent) then
        begin
            // Извлекаем каждую подмаску
            for i := 0 to regexp_entry_count(ent) do
                begin
                    writeln('Подмаска ', i, ' value: ',
                            regexp_entry_item(ent, i));
                    writeln('Подмаска ', i, ' length: ',
                            regexp_entry_length(ent, i));
                    writeln('Подмаска ', i, ' position: ',
                            regexp_entry_pos(ent, i));
                end;
        end
    else writeln('Исходная срока шаблону не соответствует: ',
                 regexp_error(ent));
    // Не забываем освобождать память
    regexp_free(ent);
    { ... }
```

Simple Data Storage

Simple Data Storage (SDS) смело можно было бы выделять в отдельный проект. Что это такое? Это редактор электронных таблиц посредством SDS2/SQL-запросов (начиная с SDS 2.0.0, раньше это был чистый API) с человекочитаемым форматом файла. Он позволяет создавать и удалять таблицы, вносить в них данные, делать выборки, редактировать и удалять записи. Зачем это нужно? Во многих проектах, особенно в сфере веб, зачастую использование баз данных типа MySQL/Postgres/FireBird — словно пальба из пушки по во-

робьям. Для решения относительно простых задач хранения данных и создано SDS. Кто-то может назвать его FPC-клоном/аналогом SQLite. Но мы не ставили такой задачи. Дело в том, что зная формат файлов SDS (а он, конечно же, документирован) можно править SDS-таблицы хоть в блокноте. Тут кто-то может упомянуть XML. SDS не соприкасается с XML, т.к. имеет более простую структуру, заточенную именно под таблицы, и потому все операции здесь проходят гораздо быстрее. Запросы SDS2/SQL частично реализуют стандарты SQL'92, но имеют упрощенную структуру. Здесь есть `CREATE`, `DROP`, `INSERT`, `UPDATE`, `DELETE`, `SELECT`, условия, сортировка, работа с датами и т.д. Но всего 6 типов данных: `VARCHAR`, `INT`, `FLOAT`, `DATE`, `TIME` и `DATETIME`. В API присутствует возможность импорта и экспорта таблиц (в том числе в формате SQL, CSV и XML). Конечно, с SDS стоит познакомиться лично, а здесь я лишь приведу для вас небольшой кусочек кода:

```
res := sds_query(
  'SELECT * FROM `test.sds` WHERE posts > 20 ORDER BY posts DESC, name ASC'
  );
if sds_result_rows(res) < 1
   then writeln(sds_result_error(res))
   else writeln('Query OK: ', sds_result_time(res):16:12, ' sec.');
writeln('------------------------------------');
while not sds_result_eof(res) do
    begin
        row := sds_fetch_row(res);
        writeln('id: ', sds_fetch_field(row, 'id'));
        writeln('name: ', sds_fetch_field(row, 'name'));
        writeln('posts: ', sds_fetch_field(row, 'posts'));
        writeln('cash: ', sds_fetch_field(row, 'cash'));
        writeln('registered: ', sds_fetch_field(row, 'regdate'));
        writeln('------------------------------------');
        sds_free_row(row);
    end;
sds_free_result(res);
```

> *А между тем, в SDS3 планируется ввести жесткую типизацию, более полно реализовать стандарты SQL'92, повысить производительность за счет новой системы ввода/вывода (с разделением таблицы на сегменты и блокировкой и изменением сегментов, а не таблиц целиком) и в то же время сделать формат файла не просто человекочитаемым, а визуальным.*

А как же stack overflow и отладка?

К сожалению, есть у исполняемых CGI-программ еще один недостаток, которого скрипты лишены — уязвимость к переполнению буфера. Конечно, тут многое зависит от программиста, но абсолютно неуязвимого кода в этом мире очень мало. Так что стоит быть бдительным, заботясь о памяти и проверках в своих программах. А поможет вам в этом Free Pascal Compiler. Есть у него несколько замечательных директив: `{$R+}`,`{$Q+}`, `{$CHECKPOINTER ON}`. С

ними программа проверяет диапазоны и указатели, завершая выполнение в случае ошибки. Что касается отладки ваших PSP-программ, то от ошибок времени выполнения можно избавиться при помощи двух способов. Первый — встроенные сообщения об ошибках в PSP. PSP проверяет аргументы на наиболее распространенные ошибки и выдает информацию о них прямо в выводе ваших страниц (точно так же делает PHP). Но более серьезные случаи — тоже не беда. Потому что никто не отбирал у вас пошаговой отладки (для этого можно отправлять заголовки заранее, создавать лог-файлы и, наконец, проверять код по частям). К тому же, FPC умеет генерировать backtrace-информацию об ошибках, сообщая вам всю ошибочную цепь вызовов, включая не только имена и адреса, но и номера строк исходных файлов, в которых ошибка возникла.

Чего-то не хватает?

Как видите, мы старались максимально облегчить жизнь веб-программиста, выбравшего для своей деятельности Free Pascal. Но все равно, нельзя объять необъятное, и для полного счастья всегда чего-то не хватает. Конечно, команда разработчиков выслушает все пожелания и запросы новых возможностей, но не всегда в вашем распоряжении нет того, что еще не вошло в состав PSP, а иногда в этом и нет необходимости. С компилятором Free Pascal поставляется прекрасный набор дополнительных пакетов с модулями, которые тоже могут пригодиться веб-программисту. Например, это модули для работы с БД MySQL или динамической генерации изображений GD. Кроме того, существуют прекрасные сторонние библиотеки, такие как библиотека LibSQL, клиентская библиотека для FirebirdSQL, коллекция TCP/IP протоколов Ararat Synapse. Но даже если этого окажется недостаточно — почему бы вам не реализовать нужную возможность для сообщества открытых исходников?

Взгляд в будущее уже сегодня

Ближайшее будущее проекта

Вероятно вы уже заметили, что в этой статье я много внимания уделяю различиям между ветками проекта, а также то, что примеры, приведенные для PSP 1.5.0 отличаются чередованием стиля underscored_naming PSP 1.4 и MixedCaseNaming PSP 1.5. Дело тут в том, что проект развивается довольно динамично, и далеко не всегда успевает перейти «на новые рельсы». О том, чего следует ожидать в ближайшем будущем я расскажу кратко в этой главе.

PSP 1.5

В первую очередь, я должен сказать о двух принципах, которыми кратко можно охарактеризовать эту ветвь: «Think Pascal» («думай в стиле Паскаль») и «Release

often, release early» («публикуй часто, публикуй рано»). Именно поэтому первостепенной задачей в этой ветви на ближайшее время является довершение полноценного статуса: полное унаследование документации и кода из 1.4 с адаптацией их под новый стиль. И во время этого процесса, и после него ветвь будет развиваться экстенсивно-эволюционным путем — устранение ошибок, реализация новых возможностей, усовершенствование организации. О принципиальных новшествах сейчас сказать трудно.

PSP-IDE и FakeLinux

В сфере внимания команды разработчиков Pascal Server Pages находится не только сама библиотека, но и сопутствующие вопросы. Об одном интересном направлении, которое уже реализовано, но еще не выпущено, я бы хотел рассказать сейчас.

Как уже было сказано выше, одной из существенных проблем для PSP-программиста является необходимость в компиляции под целевую платформу. Кроме физического использования или эмуляции ОС есть еще один способ: кросскомпиляция. FPC позволяет вам компилировать приложения не только под ту операционную систему, под которой запущен, но и под некоторые другие. Для этого нужно поставить кросскомпиляционный пакет binutils и модули для целевой платформы. Но здесь есть несколько неприятных моментов. Во-первых, не для всех платформ существуют комбинации кросскомпиляционных утилит. Во-вторых, запустить и проверить локально полученные файлы вы все равно не сможете. В-третьих, с Pascal Server Pages этот фокус не проходит. PSP использует кроме RTL еще и специфичные функции системы, для этого ему нужно подключать системные библиотеки. К сожалению, у кросскомпиляционных утилит есть существенные проблемы с динамическим связыванием, что делает кросскомпиляцию PSP возможной лишь частично (т.е. только в тех модулях, где системные средства не используются).

PSP-IDE и FakeLinux — это то, что могло бы быть полезно многим PSP-программистам, работающим под Windows, но имеющим Linux-хостинги. Это полностью интегрированная система, которая состоит из среды-оболочки и «Линукс-подмены». Работает она следующим образом: вы ведете разработку проекта полностью в специальной среде (на данный момент это особая версия Lazarus — LazarusRB, но скоро будет доступна интеграция с Syn). В то же время на вашей машине работает виртуальный сервер coLinux под управлением Debian GNU/Linux с установленным FPC и вебсервером. Все дисковые операции происходят именно на нем, и когда вы компилируете программу, она компилируется в Linux и для Linux. После этого вы можете легко протестировать ваш проект. А после тестирования — закачать на ваш Linux-хостинг. Эта система уже работает, и вы можете посмотреть ее в действии на ролике PSP FakeLinux Video 1, а если у вас хорошее соединение и желание поэкспериментировать, то можете скачать LazarusRB + FakeLinux с SVN-респозитория.

PSP 1.6

Это еще одно направление, «которого еще нет, но оно уже есть». В первую очередь следует сказать, что PSP 1.6.x будет существовать параллельно с PSP 1.5.x. Более того, на уровне PSP-программ между ними вообще не будет никаких различий. Тогда в чем же разница? В том, что в PSP 1.6 весь рабочий код библиотеки помещен в один динамический модуль (DLL/DSO). Это позволяет не только буквально разделить интерфейс и реализацию, но и имеет свои плюсы с точки зрения производительности. С такой организацией проекта сильно уменьшается размер двоичных файлов, что экономит не только FTP-трафик разработчика (заметим, что размер CGI-программы на веб-трафике никак не отражается), но и память вебсервера, потому что меньшее количество памяти расходуется на размещение сегментов кода приложений. В масштабах одного–двух приложений разница между расходом памяти PSP 1.5 и 1.6 несущественна, но в масштабах сервера — вполне ощутима.

> *Любопытный факт: динамическая версия PSP появилась еще в ветке 1.0, но перестала поддерживаться после версии 1.1.2. А между тем, домашняя страничка PSP создана именно на той старой динамической версии, и функционирует до сих пор без каких-либо проблем и сбоев. А ведь тогда мы еще не принимали во внимание такую вещь, как конфликты менеджеров памяти. . .*

PSP 2.0?

Еще давным-давно меня спрашивали, как я узнаю, что нужно начать новую ветку, и как я узнаю, что пора начать PSP 2.0. Так что же будет в PSP 2.0? Не могу вам сказать, что будет в Pascal Server Pages 2, и будет ли такая версия вообще (это зависит скорее от сообщества, чем от меня). Но могу сказать, что то, что сейчас мы условно называем PSP2 (Perfect Server Pages 2.0), будет уже совсем другой проект, преследующий немного другие цели и имеющий более широкую аудиторию. . . Но всему свое время.

«Open Source» или «мы ищем таланты»

Как бы ни развивались события, какие бы идеи не приходили в голову разработчикам, но если вы читаете эти строки, значит проект Pascal Server Pages живет самостоятельной жизнью. Процесс запущен, сообщество существует, и код открыт для энтузиастов. А значит будущее будет таким, каким его сделаете вы. Поэтому если вас заинтересовала эта статья — смело пробуйте себя в качестве члена сообщества. А быть может, вы хотите присоединиться к команде разработчиков? Тогда смело связывайтесь с одним из них. От вас потребуются некоторые знания, владение английским языком, навыки, желание и доля свободного времени. Сфер для деятельности множество: от написания примеров и

обучающих статей до совершенствования внутренних механизмов и внедрения новых возможностей.

Ссылки и контакты

Ссылки по теме и разработчики

Текущий состав Pascal Server Pages Development Team

- Владимир Сибиров a.k.a. Trustmaster (`trustware[на]bk[точка]ru`): координатор проекта, главный разработчик версий 1.0–1.4.
- Ларс Олсон a.k.a. L505 (`fpc505[на]z505[точка]com`): главный разработчик версий 1.5, 1.6 и PSP-IDE/FakeLinux.
- Также тем или иным образом в проекте были задействованы другие люди, но на данный момент в команду разработчиков они не входят.

Ссылки по PSP

- `http://www.psp.furtopia.org/` — домашняя страничка.
- `http://www.sf.net/projects/pascal-webdev/` — файловый репозиторий на SourceForge.
- `https://opensvn.csie.org/pspcgi/` — SVN-респозиторий.
- `pascal-webdev-news[на]lists.sourceforge.net` — список рассылки.

Ссылки по полезным библиотекам

- `http://www.ararat.cz/` — Ararat Synapse — прекрасная объектно ориентированная библиотека для работы по протоколам TCP/IP.
- `http://www.boutell.com/gd/` — библиотека для работы с графическими файлами GD.

Некоторые PSP-сайты

- `http://z505.com/` — портал z505 Ларса Олсона. Содержит множество проектов, связанных с языком Паскаль, Pascal Server Pages (в том числе PasWiki и PasForum), а также множество статей на различные темы ИТ не без доли соли и иронии.
- `http://www.furtopia.org/` — дружественная нам хостинговая организация, внутри которой, конечно же, можно найти сервисы на PSP.

- `http://www.de.freepascal.org/cgi-bin/cgifpcbot?channel=fpc` — PSP можно найти также и внутри сообщества Free Pascal.