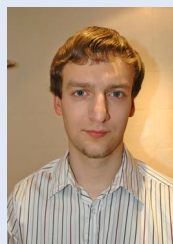


Колонка главного редактора



Последний комментарий Эрика Реймонда о лицензиях на ПО с открытым кодом напомнил всему сообществу об историческом противостоянии двух мощных сил: приверженцев термина

«Open Source» и последователей Ричарда Столлмана, ратующих за «Free Software». Первые, к которым относится и автор очередного витка бесконечных споров, получили новое объяснение, почему «вирусные» лицензии вроде GPL в некотором смысле плохи. Дело в том, что корпоративный мир, который во многом пока так и не привык к Open Source-модели, боится столь далеко идущих свобод. Свобод, которые требуют беспрекословного сохранения правил на дальнейшее распространение продуктов. Это отталкивает от использования и интеграции Open Source-компонентов в разработки того или иного бизнеса. При этом Реймонд, будучи умным и опытным аналитиком, заявляет, что компании, предпочитающие проприетарную модель, все равно будут наказаны самим рынком, отбирающим наиболее эффективные решения.

С другой стороны, нельзя недооценивать сторонников Free Software, без идей и деятельности которых мир Open Source не смог бы добиться нынешнего развития. Главное же — в том, что все это сложное «противоречие» на протяжении многих лет позволяет всему Open Source-сообществу успешно прогрессировать.

Главный редактор
Дмитрий Шурупов
(osa@samag.ru)

«Open Source»
электронное приложение к журналу
«Системный администратор»
№41, 27 марта 2009 г.

РЕДАКЦИЯ

Исполнительный директор

Владимир Положевец

Главный редактор

Дмитрий Шурупов

Верстка и оформление

Владимир Лукин

Сайт электронного приложения:

<http://osa.samag.ru>

За содержание статьи ответственность несет автор. Все права на опубликованные материалы защищены.

Новости мира Open Source

Flock отказался от Firefox в пользу Chrome

Проект социального веб-браузера Flock, основанного на Firefox, объявил о намерении отказаться от продукта Mozilla в пользу Open Source-разработки компании Google — Chrome.

Впервые Flock был представлен в октябре 2005 года, и за минувшие годы его скачали около 6 миллионов пользователей. Доля Flock на рынке веб-браузеров довольно скромна — даже меньше, чем у Netscape, который вот уже больше года официально не поддерживается. Однако, по словам разработчиков, для успеха Flock достаточно всего нескольких десятков миллионов пользователей, поскольку аудитория является довольно прибыльной из-за активности в поисковых системах.

Отказ от Firefox в качестве готовой базы, используемой Flock для создания своего социального браузера, обусловлен малым вниманием разработчиков Mozilla к проблемам Flock. Авторы проекта ожидают, что им будет проще контактировать с представителями Google. Пока переходу на Chrome мешает тот факт, что этот браузер не является межплатформенным в отличие от Firefox. Google обещает выпустить версии Chrome для GNU/Linux и Mac OS X до середины этого года. А пока следующий релиз Flock (2.1) будет по-прежнему основан на Firefox.

Французская полиция экономит миллионы евро с Open Source

Уже не один год в СМИ появляются сообщения о все новых и новых успехах французской жандармерии в переходе на программное обеспечение с открытым исходным кодом. По последним данным, полиции удалось сэкономить миллионы евро благодаря снижению затрат на информационные технологии. Например, в текущем году экономия ИТ-бюджета достигла 70 процентов.

На этот раз представитель французской полиции выступил в Утрехте (Нидерланды) на ежегодной голландской конференции, организованной местным информационным центром NOiV, специализирующимся на открытых стандартах и Open Source. С докладом о переходе с продукции Microsoft на десктопы с Ubuntu Linux выступал подполковник Хавьер Гимар (Xavier Guimard).

До 2004 года французская жандармерия закупала от 12 до 15 тысяч лицензий

на программное обеспечение, на что уходила львиная доля бюджета, выделенного на обеспечение ИТ. Но после миграции на Open Source ситуация значительно изменилась: «С июля 2007 года мы купили две сотни лицензий Microsoft. Если кому-то нужен новый компьютер, он поставляется с Ubuntu». По оценкам Гимара, экономия жандармерии на ИТ с 2004 года составила около 50 миллионов евро. Что важно, экономия была достигнута без ущерба самой ИТ-инфраструктуре.

Полиция Франции на своих десктопах использует популярные Open Source-продукты: помимо собственно дистрибутива Ubuntu Linux, это офисный пакет OpenOffice.org, веб-браузер Firefox, почтовый клиент Thunderbird.

OSI одобрила Open Source-лицензию Европейского союза

Группа Open Source Initiative (OSI), занимающаяся продвижением программного обеспечения с открытым исходным кодом, объявила об одобрении публичной лицензии на ПО, созданной комитетом Европейского союза, — EUPL (European Union Public Licence) версии 1.1.

Одобрение лицензии группой OSI означает, что программное обеспечение, которое распространяется под одобренной лицензией, соответствует определению Open Source (Open Source Definition). В последнее время OSI ужесточила требования к лицензиям, поскольку их количество начало превышать разумные рамки, что вызывает неодобрения со стороны Open Source-сообщества. Несмотря на это, группа «признанных» лицензий пополнилась разработкой от Европейского союза.

Лицензия EUPL 1.1 была опубликована Европейской комиссией 9 января этого года. Она доступна на 22 официальных языках Европейского союза (русского среди них нет). Кроме того, заявляется, что EUPL совместима с популярнейшей лицензией на свободное ПО — GNU GPLv2.

GNOME 2.26 — новая версия графической рабочей среды

Выпущена новая стабильная версия популярной графической рабочей среды с открытым исходным кодом для операционных систем UNIX и GNU/Linux — GNOME 2.26.

В очередном релизе GNOME 2.26 по традиции немало новшеств. Среди них отмечается появление новой утили-

ты для записи CD/DVD – Brasero. Вообще Brasero была и ранее доступна как самостоятельная программа, но теперь она стала приложением по умолчанию для записи дисков в GNOME, придя на смену nautilus-cd-burner. Другое новшество – удобный и простой в использовании плагин к Nautilus для быстрого расшаривания файлов (по WebDAV, HTTP, Bluetooth). В почтовом и groupware-клиенте Evolution добавлена возможность прямого импорта Microsoft Outlook Personal Folders (файлы PST) и поддержка протокола Microsoft Exchange MAPI.

Изменения затронули и мультимедийный плеер: у GNOME Media Player появился DLNA/UPnP-клиент Coherence для воспроизведения контента, распространяемого по этим протоколам (DLNA и UPnP), а также функция автоматического поиска субтитров к фильмам в сети. Другие мультимедийные изменения – это интеграция настроек громкости с PulseAudio и новая утилита Sound Preferences, улучшения в утилите Display Settings для конфигурации дополнительных подключенных к компьютеру мониторов/проекторов.

Но и этим не ограничились разработчики GNOME: множество новшеств представлено в IM-клиенте Empathy (передача файлов, звуковые темы и уведомления, улучшения в VoIP), в веб-браузере Epirhany появилась улучшенная адресная строка (подобная Awesome Bar в Firefox 3.0), была проведена интеграция демона fprintd для аутентификации пользователей по отпечатку пальца, а также интеграция Nautilus с PackageKit (для установки поддержки выбранных файлов), добавлена поддержка плагинов OpenSearch

в Deskbar, включены новые визуальные эффекты, библиотека GTK+ 2.16 пришла на смену 2.14, существенно обновилась интегрированная среда разработки Anjuta.

Создается Российская ассоциация свободного программного обеспечения

В рунете появились сведения о том, что подготовлен меморандум по созданию в России консолидированной организации, которая объединит отечественных разработчиков и интеграторов свободного программного обеспечения, – Российской ассоциации свободного программного обеспечения (РАСПО).

Сообщается, что миссией РАСПО является «содействие разработке, внедрению и популяризации свободного программного обеспечения в России, развитие отечественной индустрии программного обеспечения, основанного на открытом исходном коде и свободных лицензиях, и ее вхождение в мировой рынок разработки программного обеспечения».

Членами РАСПО «могут быть российские юридические лица, являющиеся разработчиками или внедренцами свободного ПО, либо разработчиками ПО, совместимого со свободным». Для того, чтобы присоединиться к РАСПО, потребуются рекомендации не менее чем трех действительных членов ассоциации, а также не менее 90% голосов за кандидата со стороны всех участников.

Меморандум о создании РАСПО подписали представители таких компаний, как ALT Linux, IBM, Sun Microsystems, «ГНУ/

Линуксцентр», Linux Ink, VDEL, «КОРУС Консалтинг», «Информзащита», «Этерсофт».

GNOME переходит на систему контроля версий Git

В почтовых рассылках проекта GNOME анонсирован план по переводу разработки под управление распределенной системой контроля версий Git.

До сих пор разработчики GNOME пользовались системой контроля версий Subversion (SVN), однако проведенный в конце прошлого года опрос участников проекта, результаты которого были обнародованы в январе, показал, что большинство предпочитают миграцию на Git для дальнейших разработок. Теперь о переходе GNOME с Subversion на Git объявлено официально.

Git – распределенная система контроля версий, разработанная Линусом Торвальдсом (Linus Torvalds) для Linux-ядра и адаптированная множеством других Open Source-проектов. Среди последних миграций на Git можно выделить перевод исходного кода Perl 5 на эту систему контроля версий.

Запущен новый сервер git.gnome.org, на котором сейчас представлен «предварительный функциональный вид всех git-репозиториях GNOME». Официальная миграция всех Subversion-хранилищ GNOME на Git пройдет сразу после выпуска GNOME 2.26.1, который запланирован на 16 апреля.

Дмитрий Шурупов,
по материалам www.nix.ru
(osa@samag.ru)

Обзор Open Source-игры Yo Frankie!

Так сложилось, что современная игровая индустрия далека от GNU/Linux, а может, и вовсе не знает о его существовании. Но как же быть обычному пользователю, который время от времени хочет поиграть в своем любимом дистрибутиве? Поиграть можно – хотя на современные игры и первоклассную графику рассчитывать не приходится. Конечно, не стоит сбрасывать со счетов Wine, но в статье речь пойдет о «родной» игре. Поскольку порой бывает сложно найти информацию об интересующем игровом проекте и не каждая игра находит своего игрока, этой статьей я хочу сократить расстояние между игрой и по-

тенциальным игроком, рассказав об одном свободном игровом проекте.

Big Buck Bunny

Релиз игры Yo Frankie! (<http://www.yofrankie.org>), ранее известной как Project Apricot, состоялся в ноябре 2008 года. При этом она основывается на персонажах и сюжете из короткометражного мультфильма Big Buck Bunny (<http://www.bigbuckbunny.org>), который появился еще раньше. С него и начнем.

Организация Blender Institute, входящая в состав Blender Foundation, занимается популяризацией и распростране-

нием Open Source-инструмента для работы с 3D – Blender. На ее счету – первый «открытый видеоролик», получивший название Elephants Dream (<http://www.elephantsdream.org>) и проект Big Buck Bunny. Сюжет последнего сосредоточен вокруг большого кролика по имени Buck, который проснулся однажды утром, вылез из своей норы и встретился с тремя злодеями: Frank, Rinky и Gomera. Они охотились за бабочками и кидались в героя орехами и фруктами. Большой добродушный кролик терпел недолго, придумал план возмездия и реализовал его, после чего все злодеи были наказаны. Видео можно посмотреть на YouTube (<http://www.youtube.com/watch?v=YE7VzILtp-4>).

Yo Frankie!

Мультфильм вышел весной 2008 года, после чего продолжилась работа над

свободной игрой по его мотивам. Изначально планировалось использовать движок собственной разработки – Blender Game Engine, однако позже было принято решение перейти на уже существующий – Crystal Space (подробнее об этом – см. ниже).

В любом варианте – движок кроссплатформенный, поэтому игра доступна для таких платформ, как Windows, Mac OS X, GNU/Linux. В итоге игра вышла в двух версиях – с каждым из движков.

Разработчики позаботились об открытости игры:

- ✓ при ее создании использовались уже существующие Open Source-проекты вроде Blender и Crystal Space;
- ✓ исходный код доступен под лицензиями GNU GPL и LGPL;
- ✓ содержимое (включая модели, графику и звуки) доступно под одной из лицензий Creative Commons.

При разработке игры создатели поставили для себя несколько целей:

- ✓ создать прототип полноценной игры с качеством не хуже, чем у коммерческих игр;
- ✓ разработки для Crystal Space: HDR-освещение, игровая логика, анимация персонажей;
- ✓ разработки для Blender: создание прототипа анимации, улучшения процесса разработки;
- ✓ открытая разработка;
- ✓ накопление опыта: практика, документация, DVD.

Немного о движках

Внесу некоторую ясность насчет используемых движков. Итак, изначально игра создавалась на базе Blender Game Engine (BGE), после чего разработчики сфокусировались на Crystal Space (http://en.wikipedia.org/wiki/Crystal_Space). Переход был объяснен желанием создать более зрелый проект, так как Crystal Space – более технологичный движок и обладает большими возможностями.

Но позже главным движком снова стал BGE – с целью упрощения и ускорения разработки. Дело в том, что время экспорта моделей в Blender Game Engine гораздо меньше: если для BGE – около 5 секунд, то для CS – почти минута. Соответственно, ради экономии времени (чтобы разработчики занимались разработкой, а не ожиданием завершения экспорта) было принято решение сделать Blender Game Engine основным.

На данный момент игра выпущена в двух вариантах: на базе игрового движка Blender Game Engine и Crystal Space.

Для обоих движков есть версии, доступные для трех платформ. Что касается системных требований, то нигде нет точных сведений. Лично я запускаю игру на Athlon X2 3600+ 2 МГц, 1 Гб DDR2, GeForce 7600GS. Игра идет на средних настройках с включенными shaders. На высоких начинаются отставания картинки.

Доступность

Скачать и запустить версию Yo Frankie!, основанную на движке Blender Game Engine, можно по руководству, представленному на странице <http://www.yofrankie.org/download>. В архиве, на который ведет ссылка с этой страницы, располагаются бинарные файлы для запуска под любой платформой. Доступны и 64-битные сборки. На той же странице можно получить информацию о версии на движке Crystal Space: как скачать и установить эту игру.

Помимо прочего, все материалы можно приобрести на DVD в магазине Blender (http://www.blender3d.org/e-shop/product_info.php?products_id=102). Кроме всех версий игры на диске можно найти документацию и видеоруки.

Геймплей

Как уже сообщалось, игра использует персонажей из мультфильма Big Buck Bunny, а именно – злую белку по имени Frank. От уровня к уровню ее ожидают различные препятствия и совершенно линейный сюжет. Локации достаточно красочные. Попад в воду или лаву (картина смерти в лаве достойна «Оскара»), Фрэнк возвращается на то место, с которого он прыгнул. Чтобы этого избежать, он умеет

совершать двойной прыжок, планировать в воздухе, а также умеет ударять и быстро бегать. В число способностей нашего персонажа входит возможность бросаться подобранными желудями и костями, перетаскивать предметы и овец (их тоже можно бросать). Поверженные враги в буквальном смысле распадаются на кости и в них бьет молния, взрывная волна калечит нашего персонажа – эту возможность стоит использовать для нанесения урона и вашим врагам.

Персонаж очень капризен к управлению, а это значит, что игра подойдет не каждому игроку, а только тому, кто готов с упорством начинать снова и снова, пока не преодолеет очередное препятствие.

Стоит отдельно сказать о возможности планирования нашей белки: освоив технику полета, уровни можно фактически пролетать. Учитывая, что первоначальный уровень – очень короткий, можно пройти игру за считанные минуты, облетая препятствия, лаву, воду, врагов.

Помимо обычного режима камеры есть еще два, что позволяет наблюдать за героем со всех сторон. Видимо, для демонстрации анимации и возможностей движка белка во время простоя повторяет несколько «заученных» движений – выглядит забавно. Пройдя короткий первый уровень, можно скачать дополнительные на сайте проекта: <http://www.yofrankie.org/competition-winners-announced>.

Также можно поиграть в игру вдвоем на одном компьютере. Для этого управление настраивается таким образом, чтобы игроки не мешали друг другу, а затем на одном экране, вертикально разделен-



Скриншот Yo Frankie! с yofrankie.org

ном на две части, каждый может управлять своим персонажем.

Все эти слова справедливы для версии игры на Blender Game Engine, потому что пользоваться версией Crystal Space я не смог. Этому способствовал ряд факторов: удручающая скорость работы (даже по сравнению с Blender Game Engine), низкое разрешение, отсутствие справочной системы, настроек. Что же касается различий между версиями — они есть. Так, например, в версии с Crystal Space

Фрэнк не умеет прыгать два раза, он прыгает всегда высоко, не тонет в воде и всегда быстро бегает.

Заключение

Несмотря на множество недоработок: плохое управление, угловатые модели, подтормаживания — игра оказалась достаточно интересной и увлекательной, а местами — сложной. Самая сильная сторона игры — в её открытости. Это способствовало и появлению архива с дополнительными

картами: любой может создать свой уровень, а мы с удовольствием поиграть в уже созданные кем-то.

В следующем выпуске «Open Source» я затрону уже коммерческие игры под GNU/Linux. По всем дополнениям, предложениям и поправкам обращайтесь на мой почтовый ящик.

Никита Лялин
(tinman321@gmail.com)

Веб-сервер nginx. Часть 1: Общий обзор и базовая настройка

Nginx (engine x) — это HTTP-сервер и IMAP/POP3 прокси-сервер для UNIX-подобных платформ (FreeBSD и GNU/Linux). Nginx начал разрабатываться Игорем Сысоевым, сотрудником компании «Рамблер», весной 2002 года, а осенью 2004 года появился первый публично доступный релиз. Он, как и все последующие, распространяется под лицензией BSD. На данный момент nginx работает на большом количестве высоконагруженных сайтов (среди них — «Рамблер», «Яндекс», «В Контакте», wordpress.com, Wrike и другие). Текущая версия, 0.6.x, рассматривается как стабильная с точки зрения надежности, а релизы из ветки 0.7 считаются нестабильными. При этом важно заметить, что функциональность некоторых модулей будет меняться, вследствие чего могут меняться и директивы, поэтому обратной совместимости в nginx до версии 1.0.0 не гарантируется.

Чем же nginx так хорош и почему его так любят администраторы высоконагруженных проектов? Почему бы просто не использовать Apache?

Почему Apache — плохо?

Для начала нужно объяснить, как вообще работают сетевые серверы. Те, кто знаком с сетевым программированием, знают, что по сути существуют три модели работы сервера:

- ☑ **Последовательная.** Сервер открывает слушающий socket и ждет, когда появится соединение (во время ожидания он находится в заблокированном состоянии). Когда приходит соединение, сервер обрабатывает его в том же контексте, закрывает его и снова ждет со-

единения. Очевидно, это далеко не самый лучший способ, особенно когда работа с клиентом ведется достаточно долго и подключений много. Кроме того, у последовательной модели есть еще много недостатков (например, невозможность использования нескольких процессоров), и в реальных условиях она практически не используется.

- ☑ **Многопроцессная (многопоточная).** Сервер открывает слушающий socket. Когда приходит соединение, он принимает его, после чего создает (или берет из пула заранее созданных) новый процесс или поток, который может сколь угодно долго работать с соединением, а по окончании работы завершиться или вернуться в пул. Главный поток тем временем готов принять новое соединение. Это наиболее популярная модель, потому что она относительно просто реализуется, позволяет выполнять сложные и долгие вычисления для каждого клиента и использовать все доступные процессоры. Пример ее использования — веб-сервер Apache. Однако у этого подхода есть и недостатки: при большом количестве одновременных подключений создается очень много потоков (или, что еще хуже, процессов), и операционная система тратит много ресурсов на переключения контекста. Особенно плохо, когда клиенты очень медленно принимают контент. Получаются сотни потоков или процессов, занятых только отправкой данных медленным клиентам, что создает дополнительную нагрузку на планировщик ОС,

увеличивает число прерываний и потребляет достаточно много памяти.

- ☑ **Неблокируемые сокеты/конечный автомат.** Сервер работает в рамках одного потока, но использует неблокируемые сокеты и механизм поллинга. То есть сервер на каждой итерации бесконечного цикла выбирает из всех сокетов тот, что готов для приема/отправки данных с помощью вызова `select()`. После того как socket выбран, сервер отправляет на него данные или читает их, но не ждет подтверждения, а переходит в начальное состояние и ждет события на другом socketе или же обрабатывает следующий, в котором событие произошло во время обработки предыдущего. Данная модель очень эффективно использует процессор и память, но достаточно сложна в реализации. Кроме того, в рамках этой модели обработка события на socketе должна происходить очень быстро — иначе в очереди будет скапливаться много событий, и в конце концов она переполнится. Именно по такой модели работает nginx. Кроме того, он позволяет запускать несколько рабочих процессов (так называемых *workers*), т.е. может использовать несколько процессоров.

Итак, представим следующую ситуацию: на HTTP-сервер с каналом в 1 Гбит/с подключается 200 клиентов с каналом по 256 Кбит/с:

- ☑ Что происходит в случае Apache? Создается 200 потоков/процессов, которые относительно быстро генерируют контент (это могут быть как динамические страницы, так и статические файлы, читаемые с диска), но медленно отдают его клиентам. Операционная система вынуждена справляться с множеством потоков и блокировок ввода/вывода.
- ☑ Nginx в такой ситуации затрачивает на каждый коннект на порядок меньше ресурсов ОС и памяти. Однако тут

выявляется ограничение сетевой модели nginx: он не может генерировать динамический контент внутри себя, т.к. это приведет к блокировкам внутри nginx. Естественно, решение есть: nginx умеет проксировать такие запросы (на генерирование контента) на любой другой веб-сервер (например, все тот же Apache) или на FastCGI-сервер.

Рассмотрим механизм работы связки nginx в качестве «главного» сервера и Apache в качестве сервера для генерации динамического контента:

- ☑ Nginx принимает соединение от клиента и читает от него весь запрос. Тут следует отметить, что пока nginx не прочитал весь запрос, он не отдает его на «обработку». Из-за этого обычно «ломаются» практически все индикаторы прогресса загрузки файлов – впрочем, существует возможность починить их с помощью стороннего модуля upload_progress (это потребует модификации приложения).
- ☑ После того как nginx прочитал весь ответ, он открывает соединение к Apache. Последний выполняет свою работу (генерирует динамический контент), после чего отдает свой ответ nginx, который его буферизует в памяти или временном файле. Тем временем Apache освобождает ресурсы.
- ☑ Далее nginx медленно отдает контент клиенту, тратя при этом на порядки меньше ресурсов, чем Apache.

Такая схема называется фронтэнд + бэкэнд (frontend + backend) и применяется очень часто.

Установка

Поскольку nginx только начинает завоевывать популярность, имеются некоторые проблемы с бинарными пакетами, так что будьте готовы к тому, что его придется компилировать самостоятельно. С этим обычно не возникает проблем – надо лишь внимательно прочитать вывод команды «./configure --help» и выбрать необходимые вам опции компиляции. Например, такие:

```
./configure \
# префикс установки
--prefix=/opt/nginx-0.6.x \
# расположение конфигурационного файла
--conf-path=/etc/nginx/nginx.conf \
# ... и pid-файла
--pid-path=/var/run/nginx.pid \
# имя пользователя, под которым будет запускаться nginx
--user=nginx \
# список нужных модулей
--with-http_ssl_module --with-http_gzip_static_module \
--with-http_stub_status_module \
# ... и не нужных модулей
--without-http_ssi_module --without-http_userid_module \
--without-http_autoindex_module \
--without-http_geo_module \
--without-http_referer_module \
--without-http_memcached_module \
--without-http_limit_zone_module
```

После конфигурации стоит запустить стандартную процедуру «make && make install», после чего можно пользоваться nginx. Кроме того, в Gentoo вы можете воспользоваться ebuild из стандартного дерева портов; в RHEL/CentOS – репозиторием epel (в нем расположен nginx 0.6.x) или srpm для версии 0.7, который можно скачать с <http://blogs.mail.ru/community/nginx/>; в Debian можно воспользоваться пакетом nginx из ветки unstable.

Конфигурационный файл

Конфигурационный файл nginx очень удобен и интуитивно понятен. Называется он обычно nginx.conf и располагается в \$prefix/conf, если расположение не было переопределено при компиляции. Я предпочитаю размещать его в /etc/nginx/, как делают

и разработчики всех упомянутых выше пакетов. Структура конфигурационного файла такова:

```
# Имя пользователя, с правами которого будет запускаться nginx
user nginx;
worker_processes 1; # Количество рабочих процессов
events {
    <...>
# В этом блоке указывается механизм поллинга, который будет
# использоваться (см. ниже), и максимальное количество
# возможных подключений
}

http {
    <Глобальные директивы HTTP-сервера (например, настройки
    таймаутов)>;
    <Почти все из них можно переопределить для отдельного
    виртуального хоста или пути>;

    # Описание серверов (аналог VirtualHost из Apache):
    server {
        listen *:80;
        server_name aaa.bbb;

        <Директивы сервера. Здесь обычно указывают расположение
        документов (root), перенаправления и переопределяют
        глобальные настройки>;

        # Так можно определить путь (location), для которого также
        # переопределяются практически все директивы, указанные
        # на более глобальных уровнях:
        location /abcd/ {
            <директивы>;
        }
        # Кроме того, можно сделать location по регулярному
        # выражению. Например, так:
        location ~ /\.php$ {
            <директивы>;
        }
    }

    # Другой сервер:
    server {
        listen *:80;
        server_name ccc.bbb;

        <директивы>
    }
}
```

Обратите внимание на то, что каждая директива должна оканчиваться точкой с запятой.

Обратное проксирование и FastCGI

Итак, мы рассмотрели преимущества схемы frontend + backend, разобрались с установкой nginx, структурой и синтаксисом его конфигурационного файла. Пришло время узнать, как организовать в нем обратное проксирование. Оказывается, это очень просто! Например:

```
location / {
    proxy_pass http://1.2.3.4:8080;
}
```

В этом примере все запросы, попадающие в «location /», будут проксироваться на порт 8080 сервера 1.2.3 – им может оказаться как Apache, так и любой другой HTTP-сервер. Однако тут есть несколько тонкостей, связанных с тем, что приложение будет считать, что, во-первых, все запросы приходят к нему с одного IP-адреса (что может быть расценено, например, как попытка DDoS-атаки или подбора пароля), а во-вторых, что оно запущено на хосте 1.2.3.4 и порту 8080 (соответственно генерировать не правильные редиректы и абсолютные ссылки). Чтобы избежать этих проблем без необходимости переписывания приложения, мне кажется удобной следующая конфигурация:

- ☑ Nginx слушает внешний интерфейс на порту 80.
- ☑ Если бэкэнд (допустим, Apache) расположен на том же хосте, что и nginx, то он «слушает» порт 80 на 127.0.0.1 или другом внутреннем IP-адресе.

Конфигурация nginx в таком случае выглядит следующим образом:

```
server {
    listen 4.3.2.1:80;
    # Устанавливаем заголовок Host и X-Real-IP к каждому запросу,
    # отправляемому на backend
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header Host $host:$proxy_port;
    # Или «proxy_set_header Host $host;», если приложение
    # будет дописывать «:80» ко всем ссылкам
}
```

Для того чтобы приложение различало IP-адреса посетителей, нужно либо поставить модуль `mod_extract_forwarded` (если оно исполняется сервером Apache), либо модифицировать приложение так, чтобы оно брало информацию об IP-адресе пользователя из HTTP-заголовка X-Real-IP.

Другой вариант бэкенда – это использование FastCGI (см. <http://ru.wikipedia.org/wiki/FastCGI>). В этом случае конфигурация nginx будет выглядеть примерно так:

```
server {
    <...>

    # location, в который будут попадать запросы на PHP-скрипты
    location ~ .php$ {
        # Определяем адрес и порт fastcgi-сервера,
        fastcgi_pass 127.0.0.1:8888;
        # ...индексный файл
        fastcgi_index index.php;

        # И некоторые параметры, которые нужно передать серверу
        # fastcgi, чтобы он понял, какой скрипт и с какими
        # параметрами выполнять:
        fastcgi_param SCRIPT_FILENAME _
            /usr/www/html$fastcgi_script_name; # имя скрипта
        fastcgi_param QUERY_STRING _
            $query_string; # строка запроса
        # И параметры запроса:
        fastcgi_param REQUEST_METHOD $request_method;
        fastcgi_param CONTENT_TYPE $content_type;
        fastcgi_param CONTENT_LENGTH $content_length;
    }

    # Благодаря тому, что location с регулярными выражениями
    # обладают большим «приоритетом», сюда будут попадать все
    # не-PHP-запросы:
    location / {
        root /var/www/html/
    }
}
```

Статика

Для того, чтобы меньше нагружать бэкенд, статические файлы лучше отдавать только через nginx – он с этой задачей справляется лучше, т.к. на каждый запрос тратит существенно меньше ресурсов (во-первых, не надо порождать новый процесс, а во-вторых, процесс nginx, как правило, потребляет меньше памяти, а обслуживать может множество соединений).

В конфигурационном файле это выглядит примерно так:

```
server {
    listen *:80;
    server_name myserver.com;
```

```
location / {
    proxy_pass http://127.0.0.1:80;
}

# Предположим, что все статические файлы находятся в /files:
location /files/ {
    root /var/www/html/; # Путь в файловой системе
    expires 14d; # Заголовок Expires
    # Если файл не найден, отправляем его в именованный
    # location @back
    error_page 404 = @back;
}

# Запросы из /files, для которых не было найдено файла,
# отправляем на backend, а он может либо сгенерировать
# нужный файл, либо показать красивое сообщение об ошибке
location @back {
    proxy_pass http://127.0.0.1:80;
}
```

Если вся статика не помещена в какой-то определенный каталог, можно воспользоваться регулярным выражением:

```
location ~* ^.+\. (\.jpg|jpeg|gif|png|ico|css|zip|tgz|gz|rar|bz2|_
doc|xls|xsl|exe|pdf|ppt|txt|tar|wav|bmp|rtf|js)$ {
    # Аналогично предыдущему варианту, но в этот location
    # будут попадать все запросы, оканчивающиеся на один
    # из указанных суффиксов
    root /var/www/html/;
    error_page 404 = @back;
}.
```

К сожалению, в nginx не реализована асинхронная работа с файлами. Иными словами, nginx worker блокируется на операциях ввода-вывода. Так что если у вас очень много статических файлов и, в особенности если они читаются с разных дисков, лучше увеличивать количество рабочих процессов (до числа, которое в 2-3 раза больше, чем суммарное число головок на диске). Это, конечно, ведет к увеличению нагрузки на ОС, но в целом производительность увеличивается. Для работы с типичным количеством статички (не очень большое количество сравнительно небольших файлов: CSS, JavaScript, изображения) вполне хватает одного-двух рабочих процессов.

Заключение

Во второй части статьи, которая выйдет в следующем выпуске «Open Source», будут рассмотрены другие возможности nginx, среди которых – балансировка, события, оптимизация веб-сервера и приложений.

1. <http://sysoev.ru/nginx> – официальный сайт nginx.
2. <http://wiki.codemongers.com> – Nginx Wiki.
3. <http://openhack.ru/nginx-patched> – nginx с дополнительными сторонними модулями.
4. <http://blogs.mail.ru/community/nginx> – src.rpm для последних версий nginx.
5. <http://sysoev.ru/nginx/links.html> – еще несколько хороших ссылок.
6. <http://sysoev.ru/nginx/docs/maillists.html> – список рассылки nginx-ru.

Владимир Русинов
(vladimir@greenmice.info)

Беседы о Qt. Часть 1: Забота о «детях»

Эта статья начинает серию материалов о программировании на C++ с использованием замечательной библиотеки Qt. Статьи рассчитаны на лю-

дей, которые уже знают основы C++ и знакомы с Qt на начальном или среднем уровне. Я постараюсь осветить те вопросы, которые возникают при серьезной работе

с Qt и понимание которых важно для правильного решения поставленных задач. В статье будут приводиться примеры, которые можно сразу пробовать и смотреть, что получается. Наглядные примеры – лучший способ изучения.

Недавно я общался с одним программистом, который предпочитает другим языкам C#. Он утверждал, что там есть

сборщик мусора, а отслеживать парность вызовов new и delete — слишком хлопотно. Я возразил, что, во-первых, ничуть не хлопотно, а во-вторых, библиотеки высокого уровня вроде Qt во многом берут на себя автоматическое управление памятью. Ведь почти все Qt-классы основаны на общем предке — классе QObject. В Qt при объявлении производного от QObject класса хорошим правилом является использование макроса Q_OBJECT. Помещайте его сразу в начале объявления класса — вот так:

```
class CTest: public QObject
{
    Q_OBJECT

public:
    //и так далее
}
```

Этот макрос нужен, если у вас в классе задействован механизм слотов, сигналов, свойств и прочих Qt-надстроек над C++. Если класс описан не в заголовочном, а в src-файле, использование макроса Q_OBJECT приведет к ошибкам во время сборки. Однако в примерах я буду опускать этот макрос там, где он в самом деле не очень нужен, — чтобы не увеличивать понапрасну размер статьи.

Итак, класс QObject обладает замечательной способностью. Он может хранить в себе ссылки на экземпляры других классов (производных от QObject) — список «детей», для которых он является «родителем». При создании любого наследника QObject мы можем задать для него родительский объект, передав таковой в конструкторе или позже. «Родитель» при своем уничтожении будет уничтожать и «детей», вызывая для них оператор delete. Что это дает в Qt на практике? Указав для объекта «родителя», вы можете не заботиться о ручном освобождении памяти для объекта — это дело возьмет на себя «родитель». Такое положение касается не только «невидимых» объектов, но и виджетов, которые тоже являются наследниками QObject.

Как вы знаете, лучший способ поместить много виджетов в один (например, в виджет-окно) заключается в следующем:

1. Создание объекта макета (layout) — например, для вертикального расположения помещенных туда виджетов (QVBoxLayout):

```
QVBoxLayout *layout = new QVBoxLayout;
```

2. Добавление виджетов в макет:

```
layout->addWidget (new QLabel „
```

```
("first label"));
layout->addWidget (new QLabel „
("second label"));
```

3. Установка макета для виджета-контейнера:

```
window.setLayout (layout);
```

На этом последнем шаге, когда вы присваиваете макет виджету-контейнеру, все помещенные в макет виджеты становятся (переназначаются в недрах макета) «детьми» для виджета-контейнера. Предвижу вопрос: как быть с объектами, которые объявлены внутри какого-то класса и не помещены в иерархию детей QObject? Пример:

```
class CMyClass: public Object
{
public:
    QLabel label1;
}
```

Что происходит с label1, когда уничтожается экземпляр CMyClass? Очень просто: работают обычные правила ООП, согласно которым при уничтожении объекта удаляются все его поля-объекты — причем при их удалении вызываются деструкторы. Давайте проверим на небольшом примере. Есть классы CA и CB. В CA деструктор пишет в консоль строку «~CA» — собственно, только ради этого он нам и нужен. Класс CB имеет переменную obj_a — экземпляр класса CA.

```
class CA: public QObject
{
public:
    ~CA()
    {
        qDebug() << "CA";
    }
};

class CB: public QObject
{
public:
    CA obj_a;

    ~CB()
    {
        qDebug() << "CB";
    }
};
```

Теперь напишем проверочную функцию:

```
void test()
{
    CB obj_b;
}
```

Что происходит при выполнении этой функции? Создается экземпляр класса CB — автоматически вызывается его конструктор по умолчанию. А что происходит при выходе из функции test? Вызы-

вается деструктор класса CB и деструктор для obj_b.obj_a. Что мы и наблюдаем в выводе на консоль.

Я обратился к основам ООП, потому что многие люди знают C++ лишь настолько, насколько язык им нужен на практике. «Работает, и ладно». Более того, во многих книгах по C++ пишут про конструкторы и деструкторы очень поверхностно, уделяя внимание второстепенным вещам. Поэтому я продолжу развивать тему до полного прояснения.

А что, если поле класса — указатель на класс и создан он динамически? То есть в CB переменная obj_a объявлена как указатель:

```
CA *obj_a;
```

Что будет с obj_a при разрушении экземпляра CB? Ничего не будет: obj_a останется в памяти. Это и есть утечка памяти. Есть два варианта решения этой проблемы. Первое — явное включение obj_a в иерархию «детей». Перепишем наши опытные классы следующим образом. В CA появляется конструктор: сам он ничего не делает кроме вызова конструктора своего предка — класса QObject с передачей ему параметра:

```
class CA: public QObject
{
public:
    CA (QObject *parent = 0): „
        QObject (parent) {};

    ~CA()
    {
        qDebug() << "CA";
    }
};
```

А в конструкторе класса CB мы динамически создаем obj_a как экземпляр CA и передаем ему параметр this, указывающий на внутренний экземпляр CB. Тем самым this станет родителем для obj_a:

```
class CB: public QObject
{
public:
    CA *obj_a;

    CB()
    {
        obj_a = new CA (this);
    }

    ~CB()
    {
        qDebug() << "CB";
    }
};
```

При разрушении экземпляра CB, когда будет автоматически вызван виртуальный деструктор QObject'a, он (деструктор) пройдет по списку «детей», обнаружит в них obj_a и освободит от него память.

Второй вариант более прост. Класс CA остается прежним (как в первом примере), а в деструкторе CB мы явно вызываем delete для obj_a:

```
delete obj_a;
```

В первом варианте решения задачи мы затронули важную тему, создав для класса CA конструктор:

```
CA (QObject *parent = 0): _l  
    QObject (parent) {};
```

Что произойдет, если не объявить конструктор для CA и попытаться создать экземпляр CA, передав ему параметр? В коде это будет выглядеть так:

```
obj_a = new CA (this);
```

Такое выражение не сработает – компилятор сообщит об ошибке. На первый взгляд можно возмутиться: «Как же?» Ведь в предке CA, QObject'e есть подходящий для этого конструктор:

```
QObject::QObject (QObject *parent = 0);
```

Здесь скрывается один из краеугольных камней ООП и C++. Этот конструктор QObject не может быть вызван при создании класса-наследника, если мы хотим передать ему параметр. Разберем на упрощенном примере. Отвлечемся от QObject и его наследников – рассмотрим чистое ООП без привязки к Qt (кроме «палочки-выручалочки» qDebug для вывода отладочного текста). Опишем основной класс CBase и его класс-потомок, CTest, который пуст и просто наследует CBase:

```
class CBase  
{  
public:  
  
    CBase()  
    {  
        qDebug() << "CBase";  
    };  
  
    ~CBase()  
    {  
        qDebug() << "~CBase";  
    }  
};  
  
class CTest: public CBase  
{  
};
```

Проба пера – создание экземпляра класса CTest:

```
void test()  
{  
    CTest ct;  
}
```

Запускаем и видим в консоли, как со-

общили о своем выполнении конструктор и деструктор. Обратите внимание: CBase() – это так называемый «конструктор по умолчанию». Обычно у конструктора по умолчанию нет параметров либо все параметры имеют значения по умолчанию. Перепишем конструктор:

```
CBase (int i = 777)  
{  
    qDebug() << "CBase";  
};
```

Вот теперь у него есть параметр i, и мы задаем ему значение по умолчанию – 777. Снова пробуем создать экземпляр класса-потомка:

```
void test()  
{  
    CTest ct;  
}
```

Все работает. Теперь попробуем передать параметр 555:

```
void test()  
{  
    CTest ct (555);  
}
```

Компилятор в замешательстве! Он сообщает: «error: no matching function for call to 'CTest::CTest(int)'». Итак, компилятор может для класса-потомка вызывать конструктор класса-предка лишь в следующих случаях:

- ☑ конструктор предка не имеет параметров;
- ☑ конструктор предка имеет параметры, но каждому из них задано значение по умолчанию;
- ☑ при создании экземпляра потомка в конструктор не передаются параметры.

Следовательно, если мы создаем потомок некоего класса и хотим передать этому потомку при создании параметр, надо в потомке явно описать конструктор.

Примечание: если вы не описали для класса конструктор, то компилятор создает пустой конструктор (конструктор по умолчанию, без параметров) самостоятельно. Иначе говоря, если для CTest не задать конструктор явно, то:

- ☑ Компилятор создаст для CTest пустой конструктор CTest();.
- ☑ При создании экземпляра CTest сначала будет вызван конструктор CBase (с параметрами по умолчанию), а затем конструктор CTest.

Но хватит о конструкторах. Теперь – о деструкторах, об уничтожении объектов-наследников класса QObject. Деструк-

тор QObject объявлен виртуальным, как и в нашем простейшем примере с классом CBase. Чем отличается виртуальный конструктор от обычного? Разберем на минимальном примере. Объявим два класса, деструкторы которых сообщают в консоль о своем выполнении:

```
class CBase  
{  
public:  
  
    virtual ~CBase() {qDebug() << "~CBase";}  
};  
  
class CTest: public CBase  
{  
public:  
  
    ~CTest() {qDebug() << "~CTest"; }  
};
```

Теперь проверочная функция. В ней создаем экземпляр класса CTest, однако переменная ct имеет тип CBase. Типичный случай полиморфизма, когда через переменную некоего основного класса-родителя мы работаем с экземплярами классов-потомков. Итак, создаем экземпляр CTest и удаляем его:

```
void test()  
{  
    CBase *ct = new CTest;  
    delete ct;  
}
```

На консоль выводится:

```
~CBase
```

Почему не вызвался деструктор CTest? Потому что не задействован механизм виртуальных функций! Перепишем объявление деструктора CBase, предва-рив его словом virtual:

```
virtual ~CBase() {qDebug() << "~CBase";}
```

Снова запустим проверочную функцию. Теперь в консоль выдалось:

```
~CTest  
~CBase
```

Теперь порядок: сначала вызвался деструктор потомка, затем – предка.

Зачем у QObject деструктор именно виртуальный? Чтобы правильно очищать память в «детях». Qt удаляет каждый из них оператором delete, а тот благодаря механизму виртуальных функций вызывает все деструкторы потомка, а не только деструктор основного класса QObject. Конечно, не все экземпляры классов удаляются Qt автоматически. Есть объекты вне иерархии детей, их надо удалять вручную. Вот тут возникает вопрос: вызывать ли для них delete или можно иначе? Удаление экземпляра QObject – внутренне «громоздкая» процедура. Ведь

объекты в Qt связаны «живой», действующей сетью сообщений и откликов – сигналов и слотов. Внутренне это система взаимно посылаемых сообщений. И в деструкторе QObject происходит отключение экземпляра класса от этой системы.

Однако иногда возникает опасное положение, когда экземпляр уже удален, но Qt не успела выполнить все нужные «отключения», и в очередном проходе главного цикла удаленный экземпляр продолжает использоваться Qt в системе сообщений, что приводит к ошибке и вылету программы. Это «опасное положение» не бывает случайным: если на каком-то участке кода оно проявляется, то всегда, а не от случая к случаю. И там, где оно проявилось, для удаления экземпляра используйте не delete, а функцию-слот deleteLater() самого объекта. Вот так:

```
my_object.deleteLater();
```

DeleteLater() в конечном итоге приводит к вызову того же delete, однако обходным путём. Внутренне deleteLater() выглядит как посылка в очередь сообщений сигнального сообщения DeferredDelete. И объект удаляется внутри очереди сообщений только после того, как объект получит все причитающиеся ему сообщения.

Завершая тему QObject, рассмотрим правильное приведение классовых типов

в Qt с помощью макроса qobject_cast. Например, у нас есть функция-слот для обработки сигнала, исходящего от выбора пользователем пункта меню. В этом обработчике мы по некоторой причине хотим знать, как называется этот пункт меню и какая на нем надпись. На первый взгляд просто, но на второй – не совсем. Ведь наш слот-обработчик выглядит приблизительно так:

```
void CMyClass::my_slot()
{
    //тело функции
}
```

Никаких параметров слоту не дано. Сигнал от пункта меню принимает класс CMyClass. Он, конечно же, потомок QObject. У QObject есть такая функция:

```
QObject* QObject::sender() const
```

Она возвращает указатель на объект, пославший последний (в текущее время) сигнал. Стало быть, если мы обратимся к sender() в нашем слоте, получим указатель на пункт меню, пославший слоту свой привет. Однако текст пункта меню мы можем получить функцией text(). Какого класса наш пункт меню? Грубо говоря, QAction. Именно у QAction есть такая функция – text(). Какого класса указатель нам возвращает QObject::sender()? Чистый, как февральский снег,

QObject, у которого нет никакой функции text(). Нужно приведение типа! И вот каким образом:

```
QAction *act = .j
qobject_cast<QAction *>(sender());
QDebug() << act->text();
```

Итак, для чистоты опыта мы объявили указатель act и привели к нему объект, возвращенный функцией sender(). А потом вывели на консоль текст, возвращаемый функцией text(). Макрос qobject_cast внешне действует так же, как и обычный оператор C++ dynamic_cast. Напомню, что это в Си вы могли запросто приводить к типу, написав что-то вроде «(тип) переменная». В C++ правилом хорошего тона считается использование операторов static_cast, dynamic_cast, reinterpret_cast и const_cast, например:

```
int i = 127;
char c = static_cast<char>(i);
```

И QObject прибавляет к этой четверке операторов еще и qobject_cast. Работает он только для потомков QObject и при наличии макроса Q_OBJECT в определении класса.

Продолжение статей «Беседы о Qt» – в следующих выпусках «Open Source».

Петр Семилетов
(tea@list.ru)

Тестирование программного обеспечения Open Source-средствами. Часть 3: CppUnit

Это третья, заключительная, часть статьи по тестированию ПО с помощью Open Source-средств, посвященная CppUnit. Первые две части, в которых рассказывается о JUnit и NUnit, вы можете найти в прошлых выпусках «Open Source»: №039 от 20.02.2009 и №040 от 06.03.2009 соответственно. – Прим. ред.

CppUnit

CppUnit – это библиотека для организации модульного тестирования программ на C++. Она является полностью перенесенной на язык C++ библиотекой JUnit, поэтому если вы знакомы с библиотекой JUnit версии 3.x, без особого труда сможете освоиться с тем, как создавать тесты с использованием данной библиотеки. Как это делается, мы рассмотрим в следующем разделе. Архив с исходным кодом библиотек можно скачать с сайта разра-

ботчиков (<http://sourceforge.net/projects/cppunit>). После этого потребуется собрать ее на целевой платформе:

☑ Для сборки на платформах семейства *NIX нужен компилятор GCC. Установка производится стандартно:

Таблица 1. Абстрактные классы, позволяющие сформировать набор тестов

Класс	Описание
TestCase	Класс является абстрактным и содержит один метод runTest, который необходимо перегрузить. С помощью этого класса можно реализовать простейший тест. К сожалению, вам придется реализовать класс, который должен функционировать так же, как TestCase
TestFixture	Данный класс служит основой для создания тестов. Содержит виртуальные методы, которые необходимо перегрузить при создании теста. Методы setUp, tearDown: первый отвечает за инициализацию и получение ресурсов, необходимых для выполнения теста, а второй – за освобождение полученных ресурсов. Именно этот класс вы будете чаще всего использовать в качестве основы для своих тестов
TestCaller	Данный параметризованный класс (шаблон) отвечает за выполнение конкретного метода, производящего тестирование. В конструкторе этого конкретного класса указывается название теста и передается указатель на функцию, выполняющую тест
TestResult	Данный класс служит для сбора статистики в процессе выполнения конкретного теста. Для этого в метод run объекта класса TestCaller передается ссылка на объект данного класса
TestSuite	Класс служит для объединения отдельных тестов в набор тестов. Для этого объекты класса TestCaller помещаются во внутренний список объекта данного класса методом addTest. Для запуска тестов достаточно вызвать метод run и передать в него ссылку на объект класса TestResult, который будет отвечать за сбор статистики
Test	Данный класс содержит только виртуальные методы и является предком для классов TestSuite и TestCase

```
./configure --prefix=┐  
└─> директория для установки  
make && make install
```

- ☑ Для сборки на Windows требуется Visual C++ 6.0 (более поздние версии не компилируют исходный код корректно) или C++ Builder. В составе исходных кодов имеются файлы проектов для данных сред разработки.

Состав библиотеки

Для создания тестов в библиотеке предусмотрен ряд абстрактных классов, позволяющих сформировать набор тестов, которые будут позднее исполняться специальной утилитой. Данные классы представлены в **таблице 1**. Методы, производящие непосредственное тестирование, необходимо помещать в классы, расширяющие абстрактный класс `TestFixture`. Данные методы должны иметь следующую сигнатуру: `void method_name(void)`, иначе они не будут опознаны методом, производящим исполнение тестов.

Теперь, когда стало понятно, как создавать отдельные тесты и наборы тестов, необходимо научиться выполнять эти тесты и собирать результаты их выполнения. Для этих целей в библиотеке предусмотрен специальный класс `TestRunner`.

TestRunner

Данный класс содержит всего два общедоступных метода: `addTest`, который позволяет добавлять во внутренний список отдельные тесты и наборы тестов (объекты, расширяющие абстрактный класс `Test`), и второй метод `run`, который позволяет запустить тест на исполнение. Класс является абстрактным и может иметь несколько конкретных реализаций. В настоящий момент существуют три реализации данного класса: `TextTestRunner`, `QtTestRunner` и `MfcTestRunner`. Первый, как видно из названия, осуществляет вывод информации в текстовую консоль, в то время как оставшиеся два отображают информацию в графическом окне.

Для запуска тестов необходимо создать экземпляр конкретного класса, добавить к нему тесты и запустить на выполнение, как показано в примере для `TextTestRunner`:

```
int main( int argc, char **argv)  
{  
    // Создаем объект класса TextTestRunner  
    CppUnit::TextUi::TestRunner runner;  
    // Добавляем тест в список тестов  
    runner.addTest( SimpleTestSuite::suite() );  
    // Запускаем выполнение тестов  
    runner.run();  
    return 0;  
}
```

Здесь создан класс `TestRunner`, а не `TextTestRunner` из-за того, что в таком случае мы «застрахованы» от изменений в конкретном классе, и от нас скрыта его реальная структура¹.

В отличие от рассмотренных ранее библиотек модульного тестирования² в данном случае пользователь библиотеки сам

Таблица 2. Набор специальных макросов для осуществления проверок на выполнение

Макрос	Описание
CPPUNIT_ASSERT	Проверяет условие на истинность. Если условие, переданное макросу, окажется ложным, тест будет провален и макрос сообщит об этом сообщением по умолчанию
CPPUNIT_ASSERT_MESSAGE	Работает аналогично макросу <code>CPPUNIT_ASSERT</code> – с единственным отличием: в случае ложного условия он выведет переданное ему в качестве параметра сообщение
CPPUNIT_FAIL	Вызывает провал теста и выводит сообщение, переданное ему в качестве параметра. Часто используется при отладке тестов и для оповещения о нереализованных тестах
CPPUNIT_ASSERT_EQUAL	Проверяет два значения на равенство. В случае, если значения окажутся неравными, тест будет провален и выведется сообщение по умолчанию
CPPUNIT_ASSERT_EQUAL_MESSAGE	Работает аналогично описанному ранее макросу <code>CPPUNIT_ASSERT_EQUAL</code> , но в случае провала выводит сообщение, переданное в качестве параметра
CPPUNIT_ASSERT_DOUBLES_EQUAL	Проверяет на равенство два значения, передаваемых в качестве параметров. Значения могут быть и близкими, если передаваемый параметр «дельта» задает не нулевое значение
CPPUNIT_ASSERT_THROW	Проверяет выражение на выбрасывание исключения указанного типа. Выражение и тип исключения передаются в качестве параметров
CPPUNIT_ASSERT_NO_THROW	Работает противоположно макросу <code>CPPUNIT_ASSERT_THROW</code> , проверяя, что выражение не выбрасывает никаких исключений
CPPUNIT_ASSERT_ASSERTION_FAIL	Является очень интересным тем, что он проверяет другой макрос на сбой. В случае, если проверяемый макрос выбросит исключение, данный макрос выполнится без генерации ошибки. Макрос ожидает возникновения исключения типа <code>CppUnit::Exception</code>
CPPUNIT_ASSERT_ASSERTION_PASS	Данный макрос работает противоположно макросу <code>CPPUNIT_ASSERT_ASSERTION_FAIL</code> , ожидая, что исключение типа <code>CppUnit::Exception</code> не будет выброшено

создает программу для запуска тестов, встраивая в нее сами тесты и объекты классов, отвечающие за выполнение тестов.

Макросы-утверждения

Для осуществления проверок на выполнение тех или иных условий библиотека предусматривает набор специальных макросов. Тем, кто уже успел познакомиться с библиотеками модульного тестирования `JUnit` и `NUnit`, они покажутся знакомыми, так как их отличает только то, что они являются макросами, а не статическими методами класса `Assert`. Для удобства представления все макросы сведены в одну **таблицу 2**.

В официальной документации на сайте разработчиков библиотеки вы найдете множество примеров кода, демонстрирующих работу данных макросов. А мы в свою очередь рассмотрим лишь простой пример теста.

Пример использования библиотеки

Теперь, когда мы рассмотрели основные классы и функции, необходимые для написания теста, займемся написанием простого теста. В **листинге 1** показан исходный код класса, который мы будем тестировать. Класс является очень простым: описывает автомобиль, у которого есть модель, цвет и регистрационный номер. В классе присутствует дружественный оператор сравнения двух автомобилей. Они являются равными, если равны все три параметра автомобиля: цвет, номер и модель.

Листинг 1. Исходный класс для тестирования

```
class Car  
{  
    friend bool operator== ( const Car& a, const Car& b );  
public:  
    Car( std::string model ){  
        mModel = model;  
    }  
    Car( const Car& orig ){
```

1. Данный метод применяется в некоторых случаях для того, чтобы отвязать конкретную реализацию от интерфейса, доступного пользователю. Внутри класса `TestRunner` из пространства имен `CppUnit::TextUi` инкапсулирована ссылка на реальный класс `TextTestRunner`.
2. В предыдущих частях были рассмотрены `JUnit` и `NUnit` (см. «Open Source» 039 и 040 соответственно).

```

        mColor = orig.mColor;
        mModel = orig.mModel;
        mNumber = orig.mNumber;
    }

    void SetColor( std::string color ){
        mColor = color;
    }

    void SetNumber( unsigned int number ){
        mNumber = number;
    }

    virtual ~Car( );
private:
    std::string mColor;
    std::string mModel;
    unsigned int mNumber;
};

bool operator== ( const Car& a, const Car& b ){
    return (a.mColor == b.mColor) && (a.mModel == b.mModel) &&
        (a.mNumber == b.mNumber);
}

```

Теперь протестируем класс:

- ☑ проверим работу метода сравнения двух автомобилей;
- ☑ проверим работу копирующего конструктора.

Напишем небольшой тест, расширив класс `TestFixture`, как показано в следующем **листинге 2**. При этом нам необходимо перегрузить два метода абстрактного класса: `setUp` и `tearDown`, которые отвечают за выделение и освобождение ресурсов. Первоначально мы создадим два поля класса (`car1` и `car2`) и выделим память для хранения двух объектов. При освобождении ресурсов мы должны не забыть удалить эти объекты, тем самым освободив ресурсы. Затем мы реализуем два тестирующих метода:

- ☑ **testEquality** – тестирует уже созданные объекты с помощью оператора равенства, тем самым доказывая работоспособность метода сравнения на равенство объектов;
- ☑ **testConstructor** – проверяет копирующий конструктор, для чего мы создаем новый объект класса `Car` из существующего `car1` и проверяем их на равенство.

Листинг 2. Тест класса автомобиль

```

class CarSimpleTest : public CppUnit::TestFixture{
private:
    Car *car1, *car2;
public:
    void setUp(){ // Создаем объекты класса Car
        car1 = new Car("Tribeca");
        car1.SetColor("Black");
        car1.SetNumber(12345);
        car2 = new Car("Fiesta");
        car2.SetColor("Red");
        car2.SetNumber(67890);
    }

    void tearDown(){ // Удаляем объекты класса Car
        delete car1;
        delete car2;
    }

    void testEquality(){
        CPPUNIT_ASSERT(*car1 == *car2);
        CPPUNIT_ASSERT(*car1 == *car1);
    }

    void testConstructor(){
        Car *car3 = new Car(*car1);
        CPPUNIT_ASSERT(*car3 == *car1);
        delete car3;
    }
};

```

Теперь создадим набор тестов, в который включим тест. Для этого можно поступить интересным образом, добавив в класс теста статический метод `suite`, с помощью которого можно получить набор тестов данного теста. Как это делается, показано в следующем **листинге 3**.

Листинг 3. Статический метод, создающий набор тестов

```

static CppUnit::Test *suite(){
    CppUnit::TestSuite *suite = new CppUnit::TestSuite("CarTests");
    suite->addTest(new CppUnit::TestCaller<CarSimpleTest>(
        "testEquality", &CarSimpleTest::testEquality));
    suite->addTest(new CppUnit::TestCaller<CarSimpleTest>(
        "testConstructor", &CarSimpleTest::testConstructor));
    return suite;
}

```

Внутри метода мы создаем объект-контейнер для хранения названия тестов и исполняющих их методов. Затем помещаем в контейнер тесты, задавая имя и передавая ссылку на метод с тестом. Теперь необходимо создать программу, которая будет вызывать исполнение теста. Это обычная программа на языке C/C++ и приведена в **листинге 4**.

Листинг 4. Приложение для запуска теста

```

int main( int argc, char **argv)
{
    CppUnit::TextUi::TestRunner runner;
    // Добавляем набор тестов
    runner.addTest( CarSimpleTest::suite() );
    // Запускаем на исполнение
    runner.run();
    return 0;
}

```

Компиляция тестов

Теперь необходимо скомпилировать класс и тест. Для этого потребуется подключить необходимые заголовочные файлы и библиотеку. В среде Windows вам надо в параметрах проекта добавить пути к директории с заголовочными файлами и библиотеками сборки (.lib). В среде *NIX – передать пути компилятору и линкеру для того, чтобы он смог найти соответствующие заголовочные файлы и библиотеки .so. Если вы установили библиотеку в стандартные директории, указывать их не нужно (в таком случае компилятор сам найдет указанную библиотеку).

Необходимые библиотеки:

- ☑ **libcppunit-x-xx-x.dll (.so)** – основная библиотека, необходимая для создания тестов.

Необходимые заголовочные файлы:

- ☑ **TestFixture.h** – содержит описание абстрактного класса `TestFixture`;
- ☑ **TestCaller.h** – содержит описание класса, необходимого для запуска теста `TestCaller`;
- ☑ **TestSuite.h** – содержит описание класса набора тестов `TestSuite`;
- ☑ **TextTestRunner.h** – содержит описание класса, отвечающего за исполнение набора тестов `TextTestRunner`.

Выводы

Эта часть статьи посвящена библиотеке `CppUnit` для осуществления модульного тестирования классов и модулей, написанных на языке C++. Мы рассмотрели, какие классы и специальные макросы составляют библиотеку, изучили на простом примере, как ими пользоваться.

`CppUnit` имеет отличную документацию, содержащую огромное количество примеров, которые вы можете использовать при реализации своих собственных тестов. Каждый класс отлично документирован (к сожалению, в отличие от макросов, где ситуация несколько хуже). Данная библиотека будет отличной находкой для программистов, работающих с C/C++, и позволит создавать более качественные компоненты для построения программного обеспечения.

Александр Шайхразеев
(alexander.shaykhrayeev@gmail.com)

«Open Source» приглашает к сотрудничеству!

Электронное приложение «Open Source» всегда открыто для сотрудничества с новыми авторами, с читателями и их конструктивными предложениями по улучшению издания, обоснованной критикой и любыми отзывами, с компаниями, занимающимися разработкой и продвижением программного обеспечения с открытым кодом. Приветствуются все энтузиасты, желающие опубликовать у нас свои статьи. Тематика нужных материалов очевидна из предназначения приложения,

то есть FOSS (Free and Open Source Software): теория и практическое применение; исторические сведения, анализ сегодняшнего положения, прогнозы на будущее и другие аспекты, связанные с открытым ПО.

Среди наиболее интересных на данный момент общих тем можно выделить:

- ✓ общие обзоры новых и/или интересных проектов Open Source и конкретных приложений, свежих версий дистрибутивов Linux, *BSD и других систем;

- ✓ советы и рекомендации новичкам в GNU;
- ✓ истории успеха применения/распространения ПО с открытым кодом;
- ✓ философия и идеология Free Software;
- ✓ разработка приложений с применением средств Open Source.

Желательный объем статей: 6000 или 12000 символов (с пробелами). Примеры актуальных сейчас тем для статей публикуются на <http://osa.samag.ru/todo>. Но не стоит строго ограничиваться приведенными выше рамками!

Публичное обсуждение «Open Source» проводится на форуме сайта журнала «Системный администратор» по адресу: <http://osa.samag.ru/forum>. Связаться с редакцией можно по электронной почте osa@samag.ru.

Подписные индексы:

20780*

+ диск с архивом статей
2008 года

81655**

без диска

по каталогу агентства
«Роспечать»

88099*

+ диск с архивом статей
2008 года

87836**

без диска

по каталогу агентства
«Пресса России»

* Годовой
** Полугодовой
*** Диск вкладывается
в февральский
номер журнала,
распространяется только
на территории России

Подписка на журнал «Системный администратор»

Российская Федерация

- ✓ Подписной индекс: годовой – **20780**,
полугодовой – **81655**

Каталог агентства «Роспечать»

- ✓ Подписной индекс: годовой – **88099**,
полугодовой – **87836**

Объединенный каталог «Пресса
России»

Адресный каталог «Подписка за
рабочим столом»

Адресный каталог «Библиотечный
каталог»

- ✓ Альтернативные подписные агентства:
агентство «Интер-Почта»

(495) 500-00-60, курьерская доставка
по Москве

агентство «Вся Пресса»

(495) 787-34-47

агентство «Курьер-Пресссервис»

агентство «ООО Урал-Пресс»

(343) 375-62-74

- ✓ Подписка On-line

<http://www.arzi.ru>

<http://www.gazety.ru>

<http://www.presscafe.ru>

СНГ

В странах СНГ подписка принимается в почтовых отделениях по национальным каталогам или по списку номенклатуры «АРЗИ»:

- ✓ **Азербайджан** – по объединенному каталогу российских изданий через предприятие по распространению печати «Гасид» (370102, г. Баку, ул. Джавадхана, 21)

- ✓ **Казахстан** – по каталогу «Российская пресса» через ОАО «Казпочта» и ЗАО «Евразия пресс»

- ✓ **Беларусь** – по каталогу изданий стран СНГ через РГО «Белпочта» (220050, г. Минск, пр-т Ф. Скорины, 10)

- ✓ **Узбекистан** – по каталогу «Davriy nashrlar», российские издания через агентство по распространению печати «Davriy nashrlar» (7000029, г. Ташкент, пл. Мустакиллик, 5/3, офис 33)

- ✓ **Армения** – по списку номенклатуры «АРЗИ» через ГЗАО «Армпечать» (375005, г. Ереван, пл. Сасунци Давида, д. 2) и ЗАО «Контакт-Мамул» (375002, г. Ереван, ул. Сарьяна, 22)

- ✓ **Грузия** – по списку номенклатуры «АРЗИ» через АО «Сакпресса» (380019, г. Тбилиси, ул. Хошараульская, 29) и АО «Мацне» (380060, г. Тбилиси, пр-т Гамсахурдия, 42)

- ✓ **Молдавия** – по каталогу через ГП «Пошта Молдовой» (МД-2012, г. Кишинев, бул. Штефан чел Маре, 134)

по списку через ГУП «Почта Приднестровья» (МД-3300, г. Тирасполь, ул. Ленина, 17)

по прайс-листу через ООО агентство «Editil Periodice» (МД-2012, г. Кишинев, бул. Штефан чел Маре, 134)

- ✓ Подписка для **Украины**:

Киевский главпочтамт

Подписное агентство «KSS»

Телефон/факс (044)464-0220