О системе автоматической сборки и компоновки программных проектов

Куриленко И.Е., ivan@iosys.ru
ФГУП ГосНИИ «Операционных систем»

На сегодняшний день в сфере разработки программного обеспечения (ПО) большинство серьезных проектов такие, что они содержат большой объем исходного кода, который поддерживается большим числом разработчиков. Проекты, в которых задействован единственный разработчик стали редкостью, так как они являются крайне рискованными. Для крупных же проектов актуальной является задача внедрения средств автоматизации с целью ускорения процесса разработки [1]. В данной работе будет рассмотрено одно из таких средств – средство автоматизации *сборки* проекта. Процесс сборки является неотъемлемой частью процесса разработки ПО. Он включает в себя компиляцию из исходного кода проекта бинарных исполнимых файлов и формирование исполнимого образа программного продукта (ПП) (в который помимо бинарных исполнимых файлов могут входить мультимедиа-контент, конфигурационные файлы, ресурсы и т.д.).

Ручная сборка проекта и формирования исполнимого образа для большого проекта (содержащего тысячи исходных файлов) являются крайне рутинной работой. При этом высока вероятность, что на этом этапе будут внесены ошибки. Например, если программист изменил в заголовочном файле прототип какой-то функции, то ему требуется перекомпилировать все, что зависит от этого заголовочного файла. Если таких файлов в проекте много, то пропустить один делая это в ручную очень легко, а в результате можно получить неработоспособный исполнимый образ, что заставит повторить процесс сборки «с нуля».

С учетом медлительности процесса ручного запуска на компиляцию, налицо существенное влияния на скорость процесса разработки ПО. В целом, к настоящему моменту не ставится под сомнение, что программисту требуется некая программа, позволяющая отслеживать версии файлов и перекомпилировать модули, зависящие от изменившихся файлов [2]. Чтобы избежать подобных проблем и ускорить процесс разработки применяются системы автоматизации процесса сборки. Автоматизация процесса сборки позволяет решить следующие задачи:

- Освобождение людей от рутинной, нетворческой работы;
- Ускорить процесс формирования исполнимого образа ПП и тем самым ускорить процесс разработки;

- Уменьшить число ошибок, вызванных рассинхронизациями (типичным примером рассинхронизации можно считать ситуацию когда в исполнимый образ по невнимательности программиста не попадает один из обновленных бинарных модулей, а специалист тестирования обнаруживает связанные с этим ошибки);
- Обеспечивает наличие работоспособной версии кода проекта в произвольный момент времени (что особенно важно для проектов с открытым исходным кодом).

Как автоматизировать процесс сборки? Традиционно для сборки маленьких проектов использовались компилятор командой строки и пакетные файлы, содержащие его вызов. Также в пакетном файле могут быть расположены команды формирующие исполнимый образ. Однако с ростом размера проектов у данного подхода обозначились проблемы. Во-первых, размер таких коммандных файлов вырос. Во-вторых, практически сразу возникла проблема сборки debug (отладочной) и release (поставляемой) версий продукта. Для ее решения стали использовать два подхода – либо в пакетный файл встраиваются условные конструкции (что его сильно загромождает), либо разрабатывается два пакетных файла один для debug другой для release версии (что порождает проблему синхронизации содержимого этих файлов). Третьей проблемой для крупных проектов явилось то, что зачастую не требуется полная перекомпиляция (которая является очень медлительной). В некоторых случаях можно обойтись перекомпиляцией только измененых исходных файлов и всех частей проекта, которые от них зависят, что в данном подходе оставляется на откуп компилятору (способен ли он самостоятельно понять что сборка проекта не требуется), что обеспечивается далеко не всеми из них.

Кроме выше перечисленного можно отметить, что для крупного проекта с едиными правилами оформления подпроектов команды сборки в коммандных файлах будут крайне похожи. Используя это наблюдение можно упростить содержимое этих файлов, применив макрокоманды. Но даже макрокоманды не спасут от неминуемого кошмара в случае, если необходимо из одного и того-же множества исходных файлов получать разные исполнимые редакции ПП, отличающиеся по функциональности (примером таких редакций может послужить пакет MS Office, в который входят несколько крупных приложений, которые могут распространяться и устанавливаться как все вместе, так и по отдельности).

С развитием интегрированных сред разработки стало окончательно понятно, что пакетные файлы для автоматизации сборки перестали

соответствовать современным потребностям. Появились системы автоматической сборки Apache Ant [3] и MS Build [4].

Ant разработан с учетом кросплатформенного применения, т.к. изначально разрабатывался для решения задачи автоматизации сборки и компоновки java-проектов. Основной проблемой применения пакетных файлов для java-программистов стало то, что они сильно завязаны на команды операционной системы (ОС). В случае, если необходимо обеспечить сборку на разных операционных системах, отличаются не только наборы и параметры команд ОС, но и формат коммандных файлов. Разрабатывать под каждую платформу свой собственный коммандный файл сборки – не лучший выбор для кроссплатформных приложений [2]. В результате Ant спроектирован таким образом, что часто применяемые при сборке команды ОС обернуты внутренними командами Ant, а скрипты сборки описываются в формате XML [3].

MS Build разработан компанией Microsoft после появления Ant как прямой конкурирующий продукт для платформы MS Windows и по идее, и функционалу очень похож на Ant. В MS Build также имеются команды и формат файла XML. Однако формат исходного XML-файла для MS Build кажется более низкоуровневым, что не удивительно, т.к. изначально MS Build разработан для нужд интегрированной среды разработки MS Visual Studio и лишь в последствии был представлен для разработки скриптов сборки не средствами означенной среды.

Являясь развитием идеи скриптовых или командных файлов сборки и MS Build и Ant обладают рядом недостатков:

- низкоуровневость (особенно это касается MS Build) – скрипты сборки для этих систем создать легче чем коммандный файл, но при этом сложность реализации макрокоманд (например, чтобы сделать свою собственную команду сборки в MS Build необходимо разработать динамическу библиотеку с применением C# или иного языка, поддерживающего работу с .NET) и применение xml приводит к необходимости частого повторения блоков команд в этих файлах;
- процесс настройки сборки разных редакций продукта достаточно трудоемок и основывается на применении переменных сборки.

Для того чтобы снять означенные недостатки существующих систем предлагается перейти к системе сборки, основанной на компонентном принципе. Для перехода к такой системе сборки необходимо логически разбить исходный код на компоненты $C = \{C_1..C_N\}$. Каждый компонент Ci = <Name, D, A> характеризуется следующими параметрами – именем (N), списком зависимостей (D =

{Ck} – список компонент, от которых зависит данный компонент), списком действий для сборки A. Действия, входящие в список действий для сборки являются параметризованными пользовательскими макрокомандами. Например таким действием может быть «скомпилировать проект Visual C++» (для сравнения отметим, что в MS Build подобная конструкция соответствует целому списку команд). Для задания макрокоманд используется интерпретируемый функциональный язык, что позволяет пользователю максимально просто и один раз задать типовые последовательсности сборки и публикации, а затем использовать их в скриптах. В результате скрипт сборки может редактироваться в ручную и имеет структуру, существенно более простую чем скрипты для MS Build или Ant (рис. 1). В приведенном примере vc – является именем параметризованной макрокоманды. В зависимости от задачи такая команда может обеспечить, компиляцию , публикацию, запуск модульных тестов или любую комбинацию поддерживаемых средой действий. Следует отдельно отметить, что если необходим анализ состояния переменных – он может быть скрыт в макрокоманде и не вынесен в основной скрипт.

component: SystemManager ; Конфигуратор
dependencies: Core, Factory
vc: Manager -> bin\system
vc: ManagerActions -> bin\system\core

Рис. 1. Пример описания компонента

Кроме упрощения процесса разработки скриптов сборки данный подход дает возможности: по распараллеливанию процесса сборки, по организации процесса сборки как отдельных компонент, так и компонент со всеми зависимостями, что позволяет очень просто реализовать выпуск редакций собираемого ПП, как комбинации разных компонент. Таким образом, переход к системе сборки на компонентном принципе позволяет снизить сложность разработки скриптов сборки и дает дополнительные возможности, не доступные или трудно реализуемые в других системах автоматизации процесса сборки.

Литература

1. Федотова Д., Семенов Ю., Чижик К. Case-технологии: практикум. И.: «Горячая линич-Телеком», 2005 г. – 180 с.
2. Чистяков В. MS Build – RSDN Magazine, 2004, №6, с. 1-1.
3. Apache Ant Manual - http://ant.apache.org/manual/index.html
4. Справочные сведения о MS Build - http://msdn.microsoft.com/ru-ru/library/0k6kkbsd.aspx