И.Е. Куриленко

ФГБОУ ВПО Национальный исследовательский университет «МЭИ», Москва

О РАЗВИТИИ СИСТЕМ АВТОМАТИЗАЦИИ СБОРКИ ПРОГРАММНЫХ ПРОДУКТОВ

АННОТАЦИЯ

Анализируется история возникновения, современное состояние, возможности, особенности, недостатки и тенденции развития систем автоматизации сборки. Эти системы рассматриваются как важнейший компонент процесса разработки качественного программного обеспечения. Предлагается перспективная компонентная система автоматизации сборки, позволяющая улучшить процесс разработки научного и коммерческого программного обеспечения. Работа выполняется при финансовой поддержке РФФИ.

ВВЕДЕНИЕ

На сегодняшний день в сфере разработки программного обеспечения (ПО) большинство реальных проектов содержат большой объем исходного кода, который поддерживается значительным числом разработчиков. Проекты, в которых задействован единственный разработчик стали редкостью, так как они являются крайне рискованными. В настоящее время не подвергается сомнению факт, что для крупных проектов по разработке программного обеспечения актуальной является задача внедрения средств автоматизации с целью ускорения процесса разработки [1].

В данной работе будут рассмотрены представители одного из классов таких средств – средства автоматизации *сборки* проекта. Процесс сборки является неотъемлемой частью процесса разработки ПО. Он включает в себя компиляцию из исходного кода проекта исполнимых файлов и формирование исполнимого образа программного продукта (ПП) (в который помимо бинарных исполнимых файлов могут входить сопроводительные документы (справка), мультимедиа-контент, конфигурационные файлы, ресурсы и т.д.).

Ручная сборка проекта и формирование 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 C++ 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».

То же касается и вопроса многократного повторения однотипных действий по формированию исполнимого образа приложения – при выполнении в ручном режиме человек может забыть выполнить то или иное действие (скопировать файл, запустить компилятор, обновить ресурсы и т.п.), что потенциально может привести к передаче в тест несогласованной сборки (содержащей различные (иногда несовместимые друг с другом) бинарные версии исполнимых файлов, конфликтующие конфигурации, устаревшие ресурсы и т.п.). При этом подобные нестыковки способны привести к плавающим и трудно заметным дефектам, проявляющимся во время выполнения программы. В лучшем случае эти дефекты будут обнаружены в момент тестирования, в худшем – могут проявиться на этапе эксплуатации. Таким образом, ручная сборка современных программных комплексов является неэффективной ввиду значительного объема рисков, вызванных человеческим фактором (например, усталостью). Также такой процесс может усугубляться следующими психологическими проблемами: нетворческая работа приводит к снижению производительности из-за нежелания делать монотонную работу. Ручная сборка программы приводит к удлинению цикла *Edit - Compile – Test*, что усугубляется в проектах с большим объемом кода, использующим шаблонные библиотеки. Кроме этого высоковероятны дополнительные потери при слиянии кода от нескольких разработчиков, при использовании нескольких компиляторов или поддержке нескольких конфигураций.

1. СИСТЕМЫ АВТОМАТИЗАЦИИ СБОРКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

С учетом медлительности процесса ручного запуска на компиляцию, налицо его существенное влияние на скорость процесса разработки ПО. В целом, к настоящему моменту не ставится под сомнение то, 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:

- освободить программистов от рутинной, нетворческой работы;
- ускорить процесс формирования исполнимого образа программного продукта 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 по тестированию обнаруживает связанные с этим ошибки);
- обеспечить наличие работоспособной версии кода проекта в произвольный момент времени (что особенно важно для проектов с открытым исходным кодом);
- минимизировать «плохие (некорректные)» сборки;
- устранить зависимость процесса сборки программного продукта от конкретного человека;
- обеспечить ведение истории сборок и релизов для разбора выпусков;

В совокупности все перечисленные выше возможности позволяют экономить время и деньги.

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. В итоге для автоматизации писался длинный листинг команд по сборке. Существенным преимуществом данного подхода является то, что он достаточно просто реализуется.

Однако, с ростом размера проектов у данного подхода обозначились проблемы. Во-первых, размер таких командных 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 измененных 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.

Таким образом, следует отметить следующие недостатки рассматриваемого подхода к автоматизации процесса сборки: низкий уровень настраиваемости и переносимости.

Кроме перечисленного можно отметить, что для крупного проекта с едиными правилами оформления подпроектов команды сборки в командных файлах будут крайне похожи. Используя это наблюдение можно упростить содержимое соответствующих файлов, применив макрокоманды. Но даже макрокоманды не спасут от неминуемых сложностей 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 ясно, что пакетные файлы для автоматизации сборки перестали соответствовать современным потребностям. Появились системы автоматизации сборки Apache Ant [3] и MS Build [4].

Рассмотрим базовые требования, которые в настоящее время предъявляются к системам автоматизации сборки:

- поддержка частых (в том числе планируемых сборок) для своевременного выявления проблем;
- поддержка управления зависимостями исходного кода (Source Code Dependency Management);
- поддержка разностной сборки;
- уведомление при совпадении результата сборки с имеющимися бинарными файлами;
- ускоренная сборка;
- генерация и публикация (или рассылка) отчета о результатах сборки, включающая отчеты о компиляции и линковке.

Иногда предъявляются и дополнительные требования:

- создание описания изменений (release notes) и другой сопутствующей документации (например, руководства);
- отчет о статусе сборки;
- отчет об успешном/неуспешном прохождении тестов;
- суммирование добавленных/измененных/ удаленных особенностей в каждой новой сборке;
- автоматическое создание инсталлятора.

В настоящее время известны следующие типы систем автоматизации сборки - императивные (командные, низкоуровневые) и декларативные (описательные, композитные). Декларативные системы предполагают описание собираемых сущностей, а не создание множества команд для выполнения действий над ними. Рассмотрим некоторые из известных систем автоматизации сборки.

Apache ANT – императивная командная некомпонентная система, разработанная с учетом кроссплатформенного применения, так как 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 командных файлов. Разрабатывать под каждую платформу свой собственный командный 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].

Управление процессом сборки происходит посредством XML-сценария, также называемого Build-файлом. В первую очередь этот файл содержит определение проекта, состоящего из отдельных целей (***Targets***). Цели сравнимы с процедурами в языках программирования и содержат вызовы команд-заданий (***Tasks***). Каждое задание представляет собой неделимую, атомарную команду, выполняющую некоторое элементарное действие. Между целями могут быть определены зависимости - каждая цель выполняется только после того, как выполнены все цели, от которых она зависит (если они уже были выполнены ранее, повторного выполнения не производится).Типичными примерами целей являются ***clean*** (удаление промежуточных файлов), ***compile*** (компиляция), ***deploy*** (развёртывание приложения на сервере). Конкретный набор целей и их взаимосвязи зависят от специфики проекта. Ant позволяет определять собственные типы заданий путём создания Java-классов, реализующих определённые интерфейсы.

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, так как изначально MS Build разработан для нужд интегрированной среды разработки MS Visual Studio и лишь в последствие 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 командный файл, но при этом сложность реализации макрокоманд (например, чтобы сделать свою собственную команду сборки в MS Build необходимо разработать динамическую 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.

В настоящее время активно развивается декларативная система, автоматизации сборки Apache MAVEN [4]. Она предназначена для автоматизации сборки проектов, специфицированных на XML-языке *POM*. В файлах проекта pom.xml содержится его декларативное описание, а не отдельные команды. Все задачи по обработке файлов Maven выполняет через плагины.

В настоящее время Maven развился в специализированную комплексную систему управления сложным процессом создания программного обеспечения и представляет собой обобщенную систему управления разработкой с огромным количеством дополнительных возможностей, применяемых в большинстве сценариев разработки программного обеспечения. Интеграция с этой системой есть у многих сред разработки для языка Java, например Maven подключается к Eclipse и IntelliJ IDEA.

Неоспоримым преимуществом Maven является автоматическое управление зависимостями, хорошая структурированность проектов и отсутствие скриптов сборки как таковых, а следовательно проблем с ними. К недостаткам этой системы обычно относят сложности в изучении, трудность диагностики проблем при сборке и недостаточная документированность самой системы и ее плагинов. Также следует отметить сложности в поиске нужных плагинов и их настройке.

2. КОМПОНЕНТНАЯ СБОРКА

Для того, чтобы снять означенные недостатки существующих систем предлагается перейти к системе сборки, основанной на компонентном принципе.

Это имеет смысл, так как любой программный проект содержит множество пакетов. Каждый пакет логически может быть разбит на 2 части – исходные материалы и исполнимый образ. Исполнимый образ получается на основе исходных материалов.

Для перехода к компонентной системе сборки необходимо логически разбить исходный код на компоненты $C = \{C_1..C_N\}$, а также физически структурировать размещение исходных и бинарных файлов по каталогам, придерживаясь компонентного принципа. Каждый компонент $Ci = <Name, D, A, S, B, M>$ характеризуется следующими параметрами – именем (*Name*), списком зависимостей ($D = \{C_k\}$ – список компонент, от которых зависит данный компонент), исходным кодом (*S*), конфигурационными файлами (*M*), бинарными файлами (*B*) и списком действий для сборки (*A*).

В результате сборки компонента должен получаться его релиз, содержащий правильным образом структурированные бинарные и конфигурационные файлы: $R(C_i)$ = <конфигурационные файлы,

исполнимые файлы >. Релиз системы (R) определяется как объединение релизов всех входящих в нее компонент $R(C_i)$.

Во избежание проблем при формировании из релиза компонент исполнимого образа следует обеспечить отсутствие пересечения на уровне исполнимых файлов (хотя это правило не является жестко обязательным):

$$\forall\ i \neq j : R(C_i) \cap R(C_j) = \varnothing.$$

Система накладывает следующие ограничения на зависимости компонент: недопустимость лишних зависимостей, отсутствие циклических зависимостей.

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 последовательности сборки и публикации, а затем использовать их в скриптах.

Каждый компонент снабжается описателем (играющим похожую роль, что и pom.xml в Apache Maven). Описатель может редактироваться вручную и имеет структуру, существенно более простую чем скрипты для MS Build или Ant (рис. 1). Он содержит заголовочную и декларативную секции. Заголовок содержит имя компонента, его описание и зависимости.

Декларативная часть содержит перечисление входящих в компонент проектов. В настоящее время система позволяет расширять типы используемых проектов.

На основании декларативной части автоматически создается список действий для сборки.

В приведенном примере vc – является типом собираемого проекта, соотносимого с некоторой параметризованной макрокомандой. 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 описатель.

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, предлагаемый подход предоставляет возможности по распараллеливанию и организации процесса сборки как отдельных компонент, так и компонент со всеми зависимостями, что позволяет очень просто реализовать выпуск редакций собираемого программного продукта, 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.

Рассматриваемая система реализована (на рис. 2 показан экран системы в режиме сборки) и апробирована в учебно-научном процессе кафедры прикладной математики НИУ МЭИ, ФГУП ГосНИИ «Операционных систем» и в ООО ААМ Автоматик [6].

Рис. 2. Пример работы системы автоматизации сборки

ЗАКЛЮЧЕНИЕ

Предлагаемая в работе компонентная система автоматизации сборки, созданная на основе обобщения и переосмысления опыта применения современных систем автоматизации сборки позволяет получить значительные преимущества в процессе разработки программного обеспечения, такие как: покомпонентная проверка и тестирование; выпуск различных редакций программной системы на основе разделяемого исходного кода.

СПИСОК ЛИТЕРАТУРЫ

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. Руководство по Apache Ant – Apache Ant Manual - http://ant.apache.org/manual/index.html

4. Справочные сведения по системе автоматизации сборки MS Build - http://msdn.microsoft.com/ru-ru/library/0k6kkbsd.aspx

5. Murali Kashaboina, Geeth Narayanan An Introduction to Maven. // «Eclipse Developer's Journal», 2007. http://eclipse.sys-con.com/node/393300

6. Куриленко И.Е. О системе автоматической сборки и компоновки программных проектов // Сб. док. IX международной конференции «Информатика: проблемы, методология, технологии» в 2 т. – Т.1 – Воронеж: ВГУ, 2009. - С.434–437.