1. Управление проектом: сборщики проектов

1.1. В предыдущих сериях...

В прошлом разделе мы рассмотрели:

— краткую историю;
— что скачать, и как это выбирать;
— какие шестерёнки крутятся внутри;
— структуру простого и обычного проекта;
— стандартную утилиту для создания документации;
— сторонние инструменты автоматизации сборки.

1.2. В этом разделе

Будет коротко рассмотрена мотивация создания и использования сборщиков проектов (зачем это нужно), какой эволюционный путь прошли специализированные сборщики проектов, какие новые понятия появились при их использовании. Рассмотрим два самых популярных на сегодняшний день сборщика и один экзотический, но достаточно быстро набирающий обороты. Поймём какой путь проделывает чужая библиотека, чтобы попасть в наш проект и научимся публиковать собственный код, делая тем самым вклад в сообщество.

— Ant
— Ivy;
— Maven;
— Gradle;
— Bazel;
— DSL
— XML
— Артефакт;
— Репозиторий;
— Прокси;

1.3. Мотивация и схема (зачем это нужно и как это работает)

Реальный проект - это не только непосредственный код, который пишется в IDE[1], но и большое количество всего, что мы при этом используем - библиотеки, фреймворки и т.п. То есть, мы не пишем каждый проект «с чистого листа», использование JRE/JDK нам это явно демонстрирует, поскольку в них есть уже готовые написанные классы, реализующие некоторую функциональность (подробнее на странице **??**).

[1] Чаще всего, называется бизнес-кодом или бизнес-логикой

Всё, что так или иначе используется в проекте, кроме непосредственного кода, написанного в IDE, называется **зависимостями проекта**, поскольку код всегда «опирается» на заранее созданный инструментарий. Например, мы просто пишем строку с приветствием миру, не думая, как именно она раскладывается в массив символов. Или мы просто выводим приветствие миру в терминал, а инструментарий берёт на себя открытие вспомогательных потоков и прочие сложности.

Из-за этого, *единый исполняемый файл* проекта[2], включающий в себя все зависимости, может разрастаться до нескольких сотен мегабайт, а о том, чтобы скомпилировать его со всеми зависимостям, и речи быть не может.

Сборщики проекта - это сложный инструментарий, который может скрывать обширный объём работы, который при ручном подходе требует значительных затрат и может оказаться весьма утомительным.

Они позволяют не привязываться к конкретным IDE и интерфейсу среды разработки. Имеют текстовое управление, что часто ускоряет процесс работы с ними на пред-продакшн серверах, где часто терминальный интерфейс.

Система сборки это программа, которая собирает другие программы. На вход система сборки получает исходный код, написанный разработчиком, а на выход выдаёт программу, которую уже можно запустить. Отличается от компилятора тем, что вызывает его при своей работе, а компилятор о системе сборки ничего не знает, то есть является инструментом более широкого спектра.

Сборщик часто решает целый спектр задач, для решения которых компилятор не предусмотрен. Например:

— загрузить зависимые библиотеки;
— скомпилировать классы модуля;
— сгенерировать дополнительные файлы: SQL-скрипты, XML-конфиги и пр.;
— упаковать скомпилированные классоы в архивы;
— компилировать и запустить модульные тесты и рассчитать процент покрытия;
— развернуть (deploy) файлы проекта на удаленный сервер;
— генерировать документацию и отчеты.

Системы сборок имеют схожую верхнеуровневую архитектуру:

1. Конфигурации - собственная конфигурация, где хранятся «личные» настройки системы. Например, такие как информация о месте установки или окружении, информация о репозиториях и прочее;
2. Конфигурация модуля, где описывается место расположения проекта, его зависимости и задачи, которые требуется выполнять для проекта;
3. Парсеры конфигураций - парсер способный «прочитать» конфигурацию самой системы, для её настройки соответствующим образом;

[2]Чаще всего, такой файл нужен для быстрого переноса и запуска проекта на устройстве, на котором не установлен JDK

4. Парсер конфигурации модуля, где некоторыми «понятными человеку» терминами описываются задачи для системы сборки;
5. Сама система — некоторая утилита + скрипт для её запуска в вашей ОС, которая после чтения всех конфигураций начнет выполнять тот или иной алгоритм, необходимый для реализации запущенной задачи;
6. Система плагинов — дополнительные подключаемые надстройки для системы, в которых описаны алгоритмы реализации типовых задач;
7. Локальный репозиторий — репозиторий (некоторое структурированное хранилище данных), расположенный на локальной машине, для кэширования запрашиваемых файлов на удаленных репозиториях.

Для языка Java основных систем сборок три:

— Иногда можно встретить Ant в сочетании с инструментом по управлению зависимостями Ivy, в чистом виде Ant почти никогда не применялся, описывает задачи сборки в виде XML файлов;
— Самый популярный инструмент - это Maven, используется для JavaEE и приложений с использованием Spring Framework, основана на специальном подмножестве XML, называемом POM[3];
— Gradle основной инструмент в мобильной разработке, часто описания сборки на специальном DSL на основе Groovy/Kotlin, оказываются удобнее XML;

Экзотическая Bazel. Её отличает полная кроссплатформенность и кросстехнологичность, что делает её особенно привлекательной для микросервисных проектах, использующих несколько языков программирования.

1.4. С чего всё начиналось (Ant, Ivy)

Первый специализированный инструмент автоматизации задач для языка Java. По сути, это аналог `Makefile`, то есть набор скриптов (которые в разговорной речи называются тасками[4]). В отличие от `make`, утилита Ant имеет только одну зависимость — JRE. Ещё одним важным отличием от `make` является отказ от использования команд операционной системы. Всё что пишется в Ant пишется на XML, что обеспечивает переносимость сценариев между платформами.

Управление процессом сборки происходит посредством XML-сценария, также называемого Build-файлом. В первую очередь этот файл содержит определение проекта, состоящего из отдельных целей (жарг. таргетов, стр. **??**)). Цели в терминах Ant сравнимы с процедурами в языках программирования и содержат вызовы команд-заданий (жарг. тасков). Каждое задание представляет собой неделимую, атомарную команду, выполняющую некоторое элементарное действие.

Между целями могут быть определены зависимости[5] — 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

[3] (англ. Project Object Model) модель объектов проекта
[4] от англ task - задача, задание
[5] не путать с зависимостями проекта, здесь речь исключительно о задачах и командах Ant

— clean (удаление промежуточных файлов);
— compile (компиляция всех классов);
— deploy (развёртывание приложения на сервере).

Скачивание и установка, Ant для Linux-подобных ОС выполняется командой установки в пакетном менеджере (для Debian-подобных дистрибутивов `sudo apt install ant`), а для Windows необходимо перейти на сайт https://ant.apache.org, скачать архив с приложением, распаковать его и прописать соответствующий путь в переменные среды. Проверить установку и версию можно командой

```
ant -version
```

Теперь можно написать простой сценарий HelloWorld, для этого необходимо создать файл `build.xml` (см листинг 1). В нём мы указываем следующие данные:

1. версию XML, иначе XML не работает;
2. название проекта в теге `project` параметр `name`;
3. таргет по умолчанию в теге `project` параметр `default`;
4. внутри тега `project` описания таргетов, в теге `target` параметр `name` указывает имя цели;
5. внутри тега `target` команды, которые необходимо выполнить, например, <echo>.

Листинг 1: сценарий Ant echo «Hello, World!»

```
<?xml version="1.0"?>
<project name="HelloWorld" default="hello">
  <target name="hello">
    <echo>Hello, World!</echo>
  </target>
</project>
```

Затем, нужно создать каталог `hello` и сохранить туда только что созданный файл с именем `build.xml`, который содержит сценарий. Далее надо перейти в каталог и вызвать `ant`.

Следует обратить внимание, что команды навигации (а также, копирования, создания, редактирования файлов) в терминале могут (и, скорее всего, будут) отличаться.

Полный перечень команд, например:

```
mkdir hello
cd hello
cp ../build.xml .
ant
```

В результате выполнения получим следующий вывод.

```
Buildfile: /home/hello/build.xml

hello:
[echo] Hello, World!
BUILD SUCCESSFULL
```

Что именно произошло? система сборки нашла файл сценария с именем по умолчанию, по удачному стечению обстоятельств система ищет файл с именем build.xml и выполнила цель указанную по умолчанию, здесь уже никаких удачных стечений обстоятельств, мы явно указали, что нужно по умолчанию выполнять таргет с именем hello. Чтобы не запутаться в более сложном проекте, мы указали имя проекта, с помощью атрибута name.

```
<!-- Очистка -->
<target name="clean" description="Removes all temporary files">
  <!-- Удаление файлов -->
  <delete dir="${build.classes}"/>
</target>
```

В стандартной поставке ant присутствует более 100 заранее созданных заданий, таких как: удаление файлов и директорий (delete), компиляция java-кода (javac), вывод сообщений в консоль (echo) и т.д. Вот пример реализации удаления временных файлов, используя задание delete

1.5. Репозитории, артефакты, конфигурации

1.6. Классический подход (Maven)

1.7. Всем давно надоел XML (Gradle)

1.8. Собственные прокси, хостинг и закрытая сеть

1.9. Немного экзотики (Bazel)

Ant, Ant+Ivy Первым для автоматизации этих задач появился Ant. Это аналог make-файла, а по сути набор скриптов (которые называются tasks). В отличие от make, утилита Ant полностью независима от платформы, требуется лишь наличие на применяемой системе установленной рабочей среды Java — JRE. Отказ от использования команд операционной системы и формат XML обеспечивают переносимость сценариев. 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 всех классов), deploy (развёртывание приложения на сервере). Скачивание и установка, в случае каких-то неисправностей, ant для Linux выполняется командой вроде sudo apt-get install ant, а для Windows можно перейти на сайте ant.apache.org и скачать архив. Проверить версию можно командой ant -version Теперь можно написать простой сценарий HelloWorld

<?xml version="1.0"?> <project name="HelloWorld"default="hello» <target name="hello» <echo>H
World!</echo> </target> </project>

Затем, нужно создать подкаталог hello и сохранить туда файл с именем build.xml, который содержит сценарий. Далее надо перейти в каталог и вызвать ant. Полный перечень команд:

```
  $ mkdir hello
  $ cd hello
  $ ant
  Buildfile: /home/hello/build.xml

hello:
[echo] Hello, World!
BUILD SUCCESSFULL
```

Теперь нужно бы пояснить, что произошло: система сборки нашла файл сценария с указанным по умолчанию именем build и выполнила цель с именем hello, который указан в теге проекта, с помощью атрибута default со значением hello, а также имя проекта, с помощью атрибута name. В стандартной версии ant присутствует более 100 заданий, такие как: удаление файлов и директорий (delete), компиляция java-кода (javac), вывод сообщений в консоль (echo) и т.д. Вот пример реализации удаления временных файлов, используя задание delete:

```
  <!-- Очистка -->
  <target name="clean" description="Removes all temporary files">
  <!-- Удаление файлов -->
  <delete dir="${build.classes}"/>
  </target>
```

На данный момент Ant используют в связке с Ivy, которая является гибким, настраиваемым инструментом для управления (записи, отслеживания, разрешения и отчетности) зависимостями Java проекта. Некоторые достоинства Ivy:

гибкость и конфигурируемость – Ivy по существу не зависит от процесса и не привязан к какой-либо методологии или структуре; тесная интеграция с Apache Ant – Ivy доступна как автономный инструмент, однако он особенно хорошо работает с Apache Ant, предоставляя ряд мощных задач от разрешения до создания отчетов и публикации зависимостей; транзитивность – возможность использовать зависимости других зависимостей.

Немного о функциях Ivy:

управление проектными зависимостями; отчеты о зависимостях. Ivy производит два основных типа отчетов: отчеты HTML и графические отчеты; Ivy наиболее часто используется для разрешения зависимостей и копирует их в директории проекта. После копирования зависимостей, сборка больше не зависит от Apache Ivy; расширяемость. Если настроек по умолчанию недостаточно, вы можете расширить конфигурацию для решения вашей проблемы. Например, вы можете подключить свой собственный репозиторий, свой собственный менеджер конфликтов; XML-управляемая декларация зависимостей проекта и хранилищ JAR; настраиваемые определения состояний проекта, которые позволяют определить несколько наборов зависимостей.

Apache Ivy обязана быть сконфигурирована, чтобы выполнять поставленные задачи.

Конфигурация задается специальным файлом, в котором содержится информация об организации, модуле, ревизии, имени артефакта и его расширении. Некоторые модули могут использоваться по разному и эти различные пути использования могут требовать своих, конкретных артефактов для работы программы. Кроме того, модуль может требовать одни зависимости во время компиляции и сборки, и другие во время выполнения. Конфигурация модуля определяется в Ivy файле в формате .xml, он будет использоваться, чтобы обнаружить все зависимости для дальнейшей загрузки артефактов. Понятие «загрузка» артефакта имеет различные варианты выполнения, в зависимости от расположения артефакта: артефакт может находиться на веб-сайте или в локальной файловой системе вашего компьютера. В целом, загрузкой считается передача файла из хранилища в кеш Ivy. Пример конфигурации зависимостей с библиотекой Lombok:

<ivy-module version="2.0» <info organisation="org.apache"module="hello-ivy"/> <dependencies>
<dependency org="org.projectlombok"name="lombok"rev="1.18.24"conf="build->master"/> </depe
</ivy-module>

Его структура довольно проста, первый корневой элемент <ivy-module version="2.0» указывает на версию Ivy, с которой данный файл совместим. Следующий тег <info organisation="o
ivy"/> содержит информацию о программном модуле, для которого указаны зависимости, здесь определяются название организации разработчика и название модуля, хотя можно написать в данный тег что угодно. Наконец, сегмент <dependencies> позволяет определить зависимости. В данной примере модулю необходимо одна сторонняя библиотека: lombok. Однако, у такой системы, как Ant есть и свои минусы: Ant-файлы могут разрастаться до нескольких десятков мегабайт по мере увеличения проекта. На маленьких проектах выглядит всё достаточно неплохо, но на больших они длинные и неструктурированные, а потому сложны для понимания. Что привело к появлению новой системы - Maven.

Термины, определения и сокращения

Ant	Apache Ant (англ. ant — муравей и акроним — «Another Neat Tool») — утилита для автоматизации процесса сборки программного продукта. Является кросс-платформенным аналогом утилиты make, где все команды записываются в XML-формате.
Bazel	свободный программный продукт для автоматизации сборки и тестирования приложений. Компания Google использует внутренний инструмент под названием Blaze, и выпустила свободно распространяемый вариант с открытым исходным кодом под названием Bazel (анаграмма Blaze). Bazel впервые выпущен в марте 2015.
DSL	(англ. domain-specific language, DSL — «язык, специфический для предметной области», предметно-ориентированный язык) — компьютерный язык, специализированный для конкретной области применения (в противоположность языку общего назначения, применимому к широкому спектру областей и не учитывающему особенности конкретных сфер знаний). Построение такого языка и/или его структура данных отражают специфику решаемых с его помощью задач.
Gradle	система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но предоставляющая специалььный предметно-ориентированный диалект на языках Groovy и Kotlin вместо традиционной XML-образной формы представления конфигурации проекта.
Ivy	Apache Ivy переходный пакетный менеджер. Является подпроектом Apache Ant, которому нужен для разрешения зависимостей. Внешний XML файл определяет зависимости проекта и перечисляет ресурсы, необходимые для сборки проекта. Ivy ищет и скачивает ресурсы из репозитория (приватного или публичного).
Maven	Apache Maven — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (англ. Project Object Model), являющемся подмножеством XML. В отличие от Ant, декларативный. Проект Maven издаётся сообществом Apache Software Foundation. Название системы является словом из языка идиш, смысл которого можно примерно выразить как «собиратель знания».
XML	англ. eXtensible Markup Language) — «расширяемый язык разметки». Рекомендован Консорциумом Всемирной паутины (W3C). Спецификация XML описывает документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов как программами, так и человеком, с акцентом на использование в Интернете.

Артефакт — Артефакт — это любой созданный искусственно элемент программной системы. К элементам программной системы, а, следовательно, и к артефактам, могут относиться исполняемые файлы, исходные тексты, веб страницы, справочные файлы, сопроводительные документы, файлы с данными, модели и многое другое, являющееся физическим носителем информации.

Прокси — (англ. proxy — уполномоченный, заместитель) промежуточный комплекс, выполняющий роль посредника между пользователем и целевым сервером, позволяющий клиентам как выполнять косвенные запросы к другим сетевым службам, так и получать ответы. Сначала клиент подключается к прокси и запрашивает какой-либо ресурс (например артефакт), затем прокси либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кэша.

Репозиторий — (англ. repository — хранилище) — место, где хранятся и поддерживаются какие-либо данные. Чаще всего данные в репозитории хранятся в виде файлов, доступных для дальнейшего распространения по сети