0.1. В предыдущих сериях...

На прошлом уроке мы рассмотрели:

— краткую историю и причины возникновения языка;
— что скачать, и как выбирать инструментарий, хотя чаще всего инструментарий выбирает вас, а не вы его;
— какие шестерёнки крутятся внутри и как примерно всё это работает;
— структуру простого и обычного проекта;
— стандартную утилиту для создания документации;
— сторонние инструменты автоматизации сборки, экзотический мейк и очень современный докер.

0.2. На этом уроке

поговорим о том, чем же люди ползуются, если мейк экзотический, а докер был с нами не всегда. для этого будет коротко рассмотрена мотивация создания и использования специализированных сборщиков проектов, какой эволюционный путь прошли специализированные сборщики проектов, какие новые понятия появились в программировании на языке джава 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.

0.3. Мотивация

Итак, зачем собирать проект без IDE, почему нужно использовать специальные инструменты и что такое система сборки? Научившись выполнять компиляцию и запуск с использованием большого количества ключей, можно приступить к изучению сложного инструментария, который в свою очередь может скрывать очень обширный объем работы, который при ручном подходе требует значительных затрат и может оказаться весьма утомительным. Можно было бы задать вопрос: "Зачем знать команды, если в IDE всё есть?". Здесь есть несколько нюансов:

Как упоминалось на прошлом уроке, существует огромное количество IDE, которые отличаются расположением управляющих кнопок и в принципе своим внешним видом. Если в проекте несколько участников, они все должны использовать одну и ту же IDE, чтобы не запутаться и иметь возможность синхронизировать настройки при каждом изменении. То есть, можно запускать приложения в IDE, которую нужно будет поставить на все ПК, разложить исходные коды по нужным папкам, установить все требуемые библиотеки и т.д.; Есть очень секретная информация о том, что не все программисты в команде (даже очень маленькой) одинаково сильно любят одинаковые наборы инструментов.

У сред разработки постоянно меняется интерфейс, часто даже с каждым обновлением. Помимо этого, часто оказывается, что мышкой в кучу разных диалогов долго и неудобно, поскольку основной инструмент программиста это всё-таки клавиатура;

К тому же часто бывает нужно пересобрать приложение после незначительной правки

прямо на предпродакшн-сервере, а там вовсе не подключена мышка, поэтому терминальный интерфейс сборщика - это спасение.

Уметь собирать код без среды разработки — суровая необходимость. Настолько суровая и важная, что для решения этой задачи существует особый класс программного обеспечения, называемого системами сборки.

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 она от компилятора тем, что вызывает его при своей работе, а компилятор о системе сборке и её существовании ничего не знает.

Если чуть конкретнее, то сборка, помимо компиляции, содержит в себе ещё целый спектр задач, для решения которых компилятор не предусмотрен. Например, первое, что приходит в голову

— загрузить зависимые библиотеки для вашего проекта из репозитория, возможно даже из интернета;
— скомпилировать классы модуля или всего проекта как вместе, так и по отдельности;
— сгенерировать дополнительные файлы: SQL-скрипты, XML-конфиги и т.п.;
— удалять/создавать директории и копировать в них указанные файлы;
— упаковка скомпилированных классов проекта в архивы различных форматов: zip, rar, rpm, jar, ear, war и др.;
— компиляция и запуск модульных тестов (unit-test) вашего проекта с результатами выполнения тестов и расчетом процента покрытия;
— развёртывание (deploy) файлов проекта на удаленный сервер;
— генерация документации и отчетов.

В конце концов они могут даже вообще код не компилировать, но делать автоматическую рутинную работу: генерация, архивация, операции с файлами, установка на сервер — что позволяет разработчикам эффективнее тратить своё не самое дешёвое для работодателя время.

Также, стоит упомянуть, что системы сборок имеют схожую верхнеуровневую архитектуру, которая сводится к следующему:

1. Конфигурации: собственная и модульная. Там хранятся настройки системы вроде места установки и адресов репозиториев, в модульной же информация о проекте, зависимостях и задачах;
2. Парсеры конфигураций как системный для чтения конфигурации системы и её настройки, так и модульный, преобразующий человекочитаемые задачи в исполняемые;
3. 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;
4. Система плагинов — дополнительные подключаемые надстройки для системы, в которых описаны алгоритмы реализации типовых задач;
5. Локальный репозиторий — репозиторий (некоторое структурированное хранилище некоторых данных), расположенный на локальной машине, для кэширования запрашиваемых файлов на удаленных репозиториях.

Для Java систем сборок, по большому счёту, три:

— Ant уже можно увидеть только в очень старых проектах, а в чистом видн, наверное, и вовсе не найти. Иногда можно встретить Ant в сочетании с инструментом по управлению зависимостями Ivy;
— Пожалуй, самый популярный инструмент - это Maven, им собирают не только спринг приложения, но и весь энтерпрайз, поэтому я скорее всего не ошибусь, сказав, что он заниимает процентов 70 рынка джава приложений;
— Gradle стремительно завоёвывает рынок, плотно закрепившись в мобильной разработке, андроид студия собирает свои проекты гредлом, да и в остальных проектах люди чаще находят груви и котлин более удобными, чем декларативный хмл;

Пожалуй неправильно было бы не упомянуть не совсем стандартную для Java систему сборки Bazel. Её отличает полная кроссплатформенность и кросстехнологичность, что делает её особенно привлекательной для микросервисных проектах, использующих несколько языков программирования. Получается забавно, ант уже почти не используется и сходит на нет, базель возможно наберёт обороты, а инструментов всё равно останется три. О нём мы поговорим в самом конце.

0.4. С чего всё начиналось (Ant, Ivy)

Первым специализированным инструментом автоматизации задач для языка Java появился Ant. По сути, это аналог make-файла, то есть набор скриптов (которые называются тасками, от английского таск - задача, задание). В отличие от make, утилита Ant имеет только одну зависимость — JRE. Ещё одним важным отличием от мейка является отказ от использования команд операционной системы. Всё что пишется в ант пишется на XML, что обеспечивает переносимость сценариев между платформами.

Управление процессом сборки как только что было сказано, происходит посредством XML-сценария, также называемого Build-файлом. В первую очередь этот файл содержит определение проекта, состоящего из отдельных целей (таргетов, с этим понятием мы познакомились когда говорили о мейкфайлах). Цели в терминах ант сравнимы с процедурами в языках программирования и содержат вызовы команд-заданий (тасков). 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, ant для Linux-подобных ОС выполняется привычной командой установки в вашем пакетном менеджере, вроде sudo apt install ant, а для Windows можно перейти на сайте ant.apache.org и скачать архив, распаковав его и прописав соответствующий путь в переменные среды. 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>Hello, 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
```

В результате выполнения которых получим следующий вывод

```
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.

0.5. Репозитории, артефакты, конфигурации

0.6. Классический подход (Maven)

0.7. Всем давно надоел XML (Gradle)

0.8. Собственные прокси, хостинг и закрытая сеть

0.9. Немного экзотики (Bazel)