0.0.1. В предыдущих сериях...

краткой истории и причинах возникновения язка что нужно скачать, откуда и как это всё выбирать из чего всё состоит изучим простую структуру проекта и способы его запуска коротко рассмотрим утилиту джавадок

Рассмотрели примитивный инструментарий и базовые возможности платформы для разработки приложений на языке Java.

0.0.2. На этом уроке

Мотивация Ant, Ivy Репозитории, артефакты, конфигурации Maven Gradle Собственные прокси, хостинг и закрытая сеть Немного экзотики: Bazel

0.0.3. Понятия и термины урока

Системы сборки Ant, Ivy Maven Gradle bazel Артефакт Репозиторий прокси

0.0.4. Мотивация

Зачем собирать проект без IDE и что такое система сборки? (на предыдущем уроке я говорю, цитирую ”для этого достаточно через пробел написать соответствующие ключи компиляции и запуска. Конечно, писать каждый раз эти ключи довольно утомительно и для автоматизации этого процесса придумали сборщики проектов, но всегда круто знать и уметь делать вещи в отсутствие сложного инструментария”) эту мысль как раз можно и нужно развить Теперь, 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 ручном подходе требует значительных затрат. Можно было бы задать вопрос: ”Зачем знать команды, если в IDE всё есть?”. Здесь есть несколько нюансов:

Как мы упоминали 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 и 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.; У сред разработки постоянно меняется интерфейс, с каждым обновлением. Помимо этого, тыкать мышкой в кучу разных диалогов долго и неудобно;

Уметь собирать код без среды разработки — суровая необходимость. Настолько суровая, 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. Небольшой список задач для понимания:

загрузить зависимые библиотеки для вашего проекта из сети (репозитория); скомпилировать классы модуля или всего проекта; 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; установка (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 время.

0.0.5. Ant, Ivy

0.0.6. Репозитории, артефакты, конфигурации

0.0.7. Maven

0.0.8. Gradle

0.0.9. Собственные прокси, хостинг и закрытая сеть

0.0.10. Немного экзотики: Bazel

Так же, все системы сборок имеют схожую верхнеуровневую архитектура, которая сводится к следующему:

Конфигурации

собственная конфигурация, где хранятся «личные» настройки системы. Например, такие как информация о месте установки или окружении, информация о репозиториях и прочее; конфигурация модуля, где описывается место расположения проекта, его зависимости и задачи, которые требуется выполнять для проекта;

Парсеры конфигураций

парсер способный «прочитать» конфигурацию самой системы, для её настройки соответствующим образом; парсер конфигурации модуля, где некоторыми «понятными человеку» терминами описываются задачи для системы сборки;

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; Система плагинов — дополнительные подключаемые надстройки для системы, в которых описаны алгоритмы реализации типовых задач; Локальный репозиторий — репозиторий (некоторое структурированное хранилище некоторых данных), расположенный на локальной машине, для кэширования запрашиваемых файлов на удаленных репозиториях.

Какие есть системы сборки (чисто обзор и возможно по два предложения с особенностями) Для Java систем сборок по большому счёту три:

Ant или Ant в сочетании с инструментов по управлению зависимостями Ivy; Maven; Gradle и Gradle Wrapper; Нестандартная для Java система сборки 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>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
  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> <depend
org=”org.projectlombok”name=”lombok”rev=”1.18.24”conf=”build->master”/> </dependencies> </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=”org.apache”module=”hell
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.