**8 ЛАБОРАТОРНАЯ РАБОТА №8**

**АВТОМАТИЗАЦИЯ СБОРКИ JAVA ПРОЕКТОВ С ПОМОЩЬЮ ANT**

**Цель работы:** изучить скриптовый инструмент Ant для настройки процесса сборки под свои требования и сборки Java приложений.

**8.1 Краткие теоретические сведения**

Ant – это платформо-независимый скриптовый инструмент, который позволяет создавать сценарии сборки очень похожие на скрипты "make" в C или C++.

Определенный процесс сборки — важная составляющая любого цикла разработки, поскольку позволяет сократить нестыковки между различными стадиями разработки как-то: написание кода, интеграция, тестирование, продажа. Наличие определенного процесса сборки позволит ускорить переход из одной стадии в другую. Также исчезают разного рода проблемы связанные с компиляцией, переменной classpath, и т.д., которые в совокупности могли бы быть причиной растраты в пустую, множества времени и ресурсов. Ant немного похож на фабрику. Сырьем для него является то, что создают программисты – исходный код на Java, программы, написанные на других языках, дескрипторы развертывания и т.д. Что должно получится на выходе - скомпилированные программы, готовые к использованию. В процессе получения из исходного кода готового продукта, Ant’у необходима помощь в виде build файла. Build файл (также известный как «Ant файл», или «build сценарий» («сценарий сборки») или «файл build.xml») показывает Ant’у что надо делать, чтобы превратить то что есть (как правило, исходный код) в то что вы хотите. Build файл похож на детальный план – он говорит как собрать из частей единое целое.

Build файл содержит правила, которые указывают Ant’у делать вещи

наподобие таких:

• В соответствии с тем что нужно, скопируй doit.war в каталог релиза dist;

• но, сначала ты должен собрать doit.war;

• а, для того, чтобы собрать doit.war, примени команду jar к \*.class файлам

• кстати, тебе нужно скомпилировать class файлы, если ты этого еще не

сделал.

• Для того, чтобы сделать \*.class файлы, запусти компилятор Java для \*.java

файлов.

Ant не определяет свой собственный синтаксис; вместо этого его файлы компоновки записываются в XML. Существует определенный набор XML- элементов, которые понимает Ant, и для расширения возможностей Ant могут быть определены новые элементы.

Каждый файл компоновки состоит из одного элемента высшего уровня project (проект), который, в свою очередь, состоит из одного или более элементов target (цель). target - это определенный этап компоновки, выполняющий любое количество операций, например, компилирование исходных файлов. Эти операции, сами по себе выполняются другими специализированными тегами task (задание). Эти задания затем группируются вместе в элементы target. Все операции, необходимые для компоновки, могли бы быть расположены в одном элементе target, но это уменьшило бы гибкость.

Обычно предпочтительнее разбить операции на логические этапы компоновки, и каждый этап поместить в свой собственный элемент target. Это позволит выполнять одну определенную часть общей компоновки без необходимости выполнения других частей. Например, активируя только определенные элементы target, можно откомпилировать исходный код проекта без создания загрузочных образов проекта.

Элемент высшего уровня project должен содержать атрибут default (по умолчанию), указывающий цель, которая должна быть выполнена, если при запуске Ant ничего не указывается. Затем должна быть определена сама цель при помощи элемента target.

**8.2 Текст Ant-скрипта**

<?xml version="1.0"?>

<project name="Ant-Test" default="main" basedir=".">

<property name="projectName" value="DateUtils" />

<!-- Java sources -->

<property name="src.dir" location="src" />

<!-- Java classes -->

<property name="build.dir" location="bin" />

<!-- Output, Jar -->

<property name="dist.dir" location="dist" />

<target name="init">

<!-- Create the time stamp -->

<tstamp />

<!-- Create the build directory structure used by compile -->

<mkdir dir="${build.dir}" />

</target>

<target name="compile" depends="init" description="compile the source ">

<!-- Compile the java code from ${src.dir} into ${build.dir} -->

<javac includeantruntime="false" srcdir="${src.dir}" destdir="${build.dir}" />

</target>

<target name="dist" depends="compile" description="package, output to JAR">

<!-- Create the distribution directory -->

<mkdir dir="${dist.dir}" />

<!-- Put everything in ${build} into the {$projectName}-${DSTAMP}.jar file -->

<jar jarfile="${dist.dir}/${projectName}-${DSTAMP}.jar" basedir="${build.dir}" >

<manifest>

<!-- create an executable Jar -->

<attribute name="Main-Class" value="com.mkyong.core.utils.DateUtils" />

</manifest>

</jar>

</target>

<target name="doc" depends="init" description="generate documentation">

<javadoc sourcepath="${src.dir}" destdir="${dist.dir}"/>

</target>

<target name="clean" description="clean up">

<delete dir="${build.dir}" />

<delete dir="${dist.dir}" />

</target>

<!-- Default, run this -->

<target name="main" depends="clean, compile, dist" />

</project>

**8.3 Выводы**

В ходе лабораторной работы было написан Ant-скрипт для сборки приложения, который производит компиляцию, сборку jar файла.