Приложение 8

APACHE ANT

Apache Ant – это основанный на Java набор инструментов для сборки приложений.

Ant – теговый язык. Он обрабатывает XML-файлы, организованные особым образом. Каждый тег по сути является Java-классом, и есть возможность создавать свои теги или расширять возможности уже имеющихся.

Ant – многоплатформенный, основанный на использовании командной строки продукт, следовательно, возможна интеграция с другими операционными системами.

Вместо того чтобы наследовать функции командной строки, Ant основан на Java-классах. Конфигурационный файл устроен в виде XML, из которого можно вызвать разветвлённую систему целей, состоящую из множества мелких задач. Каждая задача является объектом, который наследует соответствующий интерфейс класса **`Task`**. Всё это даёт возможность переносить программу с платформы на платформу. И если действительно необходимо вызвать какой-либо процесс у Ant есть задача **`<exec>`**, которая позволяет это сделать в зависимости от платформы.

Требования к системе

Ant может быть успешно использован на многих платформах, включая Linux, коммерческие версии Unix, такие как Solaris и HP-UX, Windows 9x и NT, OS/2 Warp, Novell Netware 6 и MacOS X.

Чтобы обрабатывать и использовать Ant, необходимо иметь JAXP-compliant XML-parser (он есть в любой версии JDK) установленным и включённым в **`classpath`** или лежащим в папке с библиотеками Ant. Для работы необходимо также иметь установленный JDK версии 1.2 или выше.

Установка Ant

Для начала работы достаточно скопировать Ant на компьютер и установить необходимые системные переменные.

Пусть Ant установлен на **`c:\ant\`**. Переменные устанавливаются следующим образом:

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\jdk1.2.2
set PATH=%ANT_HOME%\bin;%PATH%
```

Создание простейшего build-файла

Build-файлы Ant пишутся на языке XML. Каждый **`build`**-файл содержит один проект (**`project`**) и хотя бы одну цель (**`target`**). Цель содержит задачи (**`tasks`**). Каждая задача, встречающаяся в **`build`**-файле, может иметь **`id`** атрибут и может быть позже вызвана по нему. Идентификаторы должны быть уникальными.

Используемые теги:

Project

Тег **`Project`** имеет три атрибута:

Атрибут	Описание	Обязательность
`name`	Имя проекта	Нет
`default`	Цель по умолчанию, которая будет использоваться, если явно не указанно, какую цель выполнять	Да
`basedir`	Основная директория, из которой будут выходить все пути, использующиеся при работе (если она не указана, то будет использоваться текущая директория, в которой находится build-файл)	Нет

Каждый проект содержит одну или несколько целей. Цель представляет собой набор задач, которые необходимо выполнить. При запуске Ant можно выбрать цель, которую(ые) следует выполнить. Если цель не указывать, будет выполнена установленная по умолчанию.

Targets

Цель может зависеть от других целей. Например, имеются две цели: для компиляции и для изъятия файлов с базы данных. Соответственно скомпилировать файлы можно только после того, как они будут извлечены. Ant учитывает такие зависимости.

Следует отметить, что **`depends`**-атрибут Ant только обозначает порядок, в котором цели должны быть выполнены. Ant пробует выполнить цели в порядке, соответствующем порядку их появления в атрибуте **`depends`** (слева направо).

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

Пусть нужно выполнить цель D. Из её атрибута **`depends`** можно узнать, что первой выполнится цель C, затем B и, наконец, A. Неверно: C зависит от B, а B зависит от A, таким образом, первой выполнится цель A, затем B, потом C, а после D.

Цель будет исполнена только один раз, даже если более чем одна цель зависит от неё.

Цель также имеет возможность быть исполненной только в случае, если определённый параметр (**`property`**) был (или не был) установлен. Это позволяет лучше контролировать процесс сборки (например, в зависимости от операционной системы, версии Java и т.д.). Ant только проверяет, установлено ли то либо иное свойство, значение его не важно. Свойство, значением которого является пустая строка, считается заполненным. Например:

```
<target name="build-module-A" if="module-A-present"/>
<target name="build-own-fake-module-A" unless=
            "module-A-present"/>
```

Если не установлены **if** и **unless** атрибуты, цель будет выполняться всегда.

Опциональный атрибут **description** может быть использован как описание цели и будет выводиться при команде– **projecthelp**.

Target имеет следующие атрибуты:

Атрибут	Описание	Обязательность
name	Имя цели	Да
depends	Разделённый запятыми список имён целей, от которых эта цель зависит	Нет
if	Имя параметра, который должен быть установлен, чтобы эта цель выполнялась	Нет
unless	Имя параметра, который не должен быть установлен, чтобы эта цель выполнялась	Нет
description	Небольшое описание функции **function** цели	Нет

Имя цели должно состоять только из букв и цифр, включая пустую строку "", "," и пробел.

Пример **build-**файла:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<project name="MyProject" default="dist" basedir=".">
    <description>
        Простой пример build файла
    </description>
  <!-- установка глобальных параметров -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>
  <target name="init">
    <!-- Создать марку времени -->
    <tstamp/>
    <!-- Создать структуру build директории, которая будет использоваться при компиляции-->
    <mkdir dir="${build}"/>
  </target>
  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Компиляция java кода из ${src} в ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>
  <target name="dist" depends="compile"
        description="генерация дистрибутива" >
    <!-- создание директории для дистрибутива -->
    <mkdir dir="${dist}/lib"/>

    <!-- Положить всё из ${build} в MyProject-${DSTAMP}.jar файл -->
```

```
      <jar   jarfile="${dist}/lib/MyProject-${DSTAMP}.jar"
basedir="${build}"/>
    </target>
    <target name="clean"
          description="очищает рабочие каталоги" >
  <!-- Delete the ${build} and ${dist} directory trees -->
      <delete dir="${build}"/>
      <delete dir="${dist}"/>
    </target>
  </project>
```

Некоторым целям было дано описание. Это значит, что командой **projecthelp** будет получен список этих целей с описанием; остальные цели считаются внутренними и не выводятся.

Чтобы всё работало, исходные коды в **src** поддиректории должны располагаться в соответствии с именами их **package.**

Пример результата выполнения:

```
D:\tmp\1>ant
Buildfile: build.xml
init:
    [mkdir] Created dir: D:\tmp\1\build
compile:
    [javac] Compiling 1 source file to D:\tmp\1\build
dist:
    [mkdir] Created dir: D:\tmp\1\dist\lib
      [jar]  Building  jar:  D:\tmp\1\dist\lib\MyProject-
20070815.jar
BUILD SUCCESSFUL
Total time: 3 seconds
```

Property

Свойства в Ant аналогичны переменным в языках программирования тем, что имеют имя и значение. Однако, в отличие от обычных переменных, свойства в Ant не могут быть изменены после их установки: они постоянны.

```
<property name="path"  value="./project"/>
```

Для обращения к этому свойству в остальных местах нашего файла компоновки можно было бы использовать следующий синтаксис:

```
${path}
```

Например:

```
<property name="libpath" value="${path}/lib"/>
```

Ant также позволяет установить переменные в отдельном property-файле. Пример property-файла:

```
#
# A sample "ant.properties" file
#
month=30 days
year=2004
```

и его использования

```
<?xml version="1.0"?>
```

```
<project name="test.properties" default="all" >
   <property file="ant.properties"/>
   <target name="all" description="Uses properties">
         <echo>This month is ${month}</echo>
         <echo>This year is ${year}</echo>
   </target>
</project>
```

Шаблоны

Часто является полезным выполнить эти операции с группой файлов сразу, например, со всеми файлами в указанном каталоге с названиями, заканчивающимися на .java, но не начинающимися с EJB. Пример копирования таких файлов:

```
<copy todir="archive">
   <fileset dir="src">
         <include name="*.java"/>
         <exclude name= "EJB*.java"/>
   </fileset>
</copy>
```

Filter

При работе с текстовыми файлами можно использовать фильтр для вставки любого текста в определенные места.

```
<filterset id="copy.filterset">
    <filter token="version" value="1.1"/>
</filterset>
<target name="copy">
    <copy file="file1.txt" tofile="file2.txt" filter-
ing="true">
        <filterset refid="copy.filterset" />
    </copy>
</target>
```

Содержимое исходного текстового файла:

```
# file.txt
Version is @version@
```

В результате получаем:

```
# file.txt
Version is 1.1
```

Path-like структуры

Можно определить типы ссылок **path** и **classpath**, используя как ":" (unix-style) так и ";" (windows-style) как разделитель символов. Ant скорректирует их в требуемые текущей операционной системой.

В случае, когда **path-like** значение надо определить, могут использоваться подсоединяемые элементы (nested elements). Это выглядит примерно так:

```
<classpath>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/helper.jar"/>
</classpath>
```

Атрибут **location** определяет отдельный файл или директорию, в то время как атрибут **path** принимает список из **locations**. Атрибут **path** должен использоваться с только с определённым ранее путём.

В качестве сокращения <**classpath**> поддерживает **path** и **location** атрибуты так:

```
<classpath>
  <pathelement path="${classpath}"/>
</classpath>
```

Может быть сокращено до:

```
<classpath path="${classpath}"/>
```

В дополнение **DirSets**, **FileSets** и **FileLists** могут быть использованы как внутренние:

```
<classpath>
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
  <dirset dir="${build.dir}">
    <include name="apps/**/classes"/>
    <exclude name="apps/**/*Test*"/>
  </dirset>
  <filelist refid="third-party_jars"/>
</classpath>
```

Path-like структуры могут содержать ссылки на другие **path-like** структуры с помощью **<path>** элемента:

```
<path id="base.path">
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
</path>

<path id="tests.path">
  <path refid="base.path"/>
  <pathelement location="testclasses"/>
</path>
```

Опции командной строки

Запускать Ant на исполнение той или иной задачи очень просто:

ant [options] [target [target2 [target3] ...]]

Options

-help, -h	Выводит текущее сообщение
-projecthelp, -p	Выводит помощь по проекту
-version	Выводит версию и выходит

-diagnostics	Выводит полезную информацию для диагностики отлова ошибок
-quiet, -q	Очень тихий режим работы
-verbose, -v	Режим, выводящий максимум возможной информации
-debug, -d	Выводить информацию отладки
-lib <path>	Определяет путь, по которому происходит поиск **jar** файлов и **class** файлов
-logfile <file>	Использовать файл для лога
-l <file>	"
-logger <classname>	Класс, который обрабатывает логинг
-noinput	Запрещает интерактивный ввод информации
-buildfile <file>	Использовать данный **build**-файл (по умолчанию берётся **build.xml**)
-file <file>	"
-f <file>	"
-D<property>=<value>	Использовать значение для данного параметра
-keep-going, -k	Выполнять все цели, не имеющие зависимостей при ошибке в одной из них
-propertyfile <name>	Загрузить все параметры из файла с **-D**

Краткий экскурс по задачам Ant

Ant предоставляет слишком много задач, чтобы дать полное описание того, что каждая из них делает. Следующий список дает представление о категориях, на которые можно разделить все задачи.

```
Archive Tasks
Audit/Coverage Tasks
Compile Tasks
Deployment Tasks
Documentation Tasks
EJB Tasks
Execution Tasks
File Tasks
Java2 Extensions Tasks
Logging Tasks
Mail Tasks
Miscellaneous Tasks
.NET Tasks
Pre-process Tasks
Property Tasks
Remote Tasks
SCM Tasks
Testing Tasks
Visual Age for Java Tasks
```

Краткое описание основных:

Archive Tasks

Имя задачи	Описание
Jar	Упаковывает в Jar набор файлов
Unzip	Распаковывает **zip** архивы
Zip	Создаёт **zip** архивы

Compile Tasks

Имя задачи	Описание
Javac	Компилирует определённые исходные файлы внутри запущенной Ant'ом VM, или с помощью новой VM, если **fork** атрибут определён
JspC	Запускает JSP-компилятор. Используется для предварительной компиляции JSP-страниц для более быстрого запуска их с сервера, или при отсутствии JDK на нём, или просто для проверки синтаксиса, без установки их на сервер
Wljspc	Компилирует JSP-страницы, используя Weblogic JSP компилятор

Execution Tasks

Имя задачи	Описание
Ant	Запускает Ant для выбранного **build** файла, возможна передача параметров (или их новых значений). Эта задача может быть использована для запуска подпроектов
AntCall	Запускает другую цель внутри того же **build-**файла, по желанию передавая параметры
Exec	Исполняет системную команду. Когда атрибут **os** определён, команда исполняется, только если Ant запущен под определённую систему
Java	Исполняет Java класс внутри запущенной (Ant) VM или с помощью другой, если **fork** атрибут определён

File Tasks

Имя задачи	Описание
Copy	Копирует файл или **Fileset** в новый файл или директорию
Delete	Удаляет как один файл, так и все файлы и поддиректории в определённом каталоге, или набор файлов, определённых одним или несколькими **FileSet'**ами
Mkdir	Создаёт директорию. Не существующие внутренние директории создадутся, если будет необходимость
Move	Переносит файл в новый файл или каталог, или набор(ы) файлов в новую директорию

Miscellaneous Tasks

Имя задачи	Описание
Echo	Выводит текст в **System.out** или в файл
Fail	Выходит из текущей сборки, генерируя **BuildException**, по желанию печатая сообщение
Input	Позволяет пользователю интерактивно вмешиваться в процесс сборки путём вывода сообщений и считывания строки с консоли
Taskdef	Добавляет задачу в проект, после чего она может быть использована в текущем проекте

Property Tasks

Имя задачи	Описание
Available	Устанавливает параметр, если определенный файл, каталог, **class** в **classpath**, или JVM системный ресурс доступен во время выполнения
Condition	Устанавливает параметр, если определённое условие выполняется
LoadFile	Загружает файл в параметр
Property	Устанавливает параметр (по имени и значению), или набор параметров (из файла или ресурса) в проект

Типы

Краткий список основных типов (на самом деле их больше):

DirSet
FileSet
PatternSet

DirSet представляет собой набор каталогов. Эти каталоги могут находиться в базовой директории, и поиск осуществляется по шаблону. **DirSet** может находиться внутри некоторых задач или выноситься в проект с целью дальнейшего к нему обращения по ссылке.

PatternSet (набор шаблонов) может быть использован как внутренняя задача. В дополнение **DirSet** поддерживает атрибуты **PatternSet** и внутренние **<include>**, **<includesfile>**, **<exclude>** и **<excludesfile>** элементы **<patternset>**.

Атрибут	Описание	Обязательность
dir	Корневая директория этого **DirSet**	Да
includes	Список шаблонов (через запятую или пробел) для каталогов, которые должны быть включены, если атрибут пропущен, все каталоги включаются	Нет
includesfile	Имя файла; каждая строчка этого файла понимается как шаблон для включения в поиск	Нет

excludes	Список шаблонов (через запятую или пробел) для каталогов, которые должны быть исключены, если атрибут пропущен, все каталоги включаются	Нет
excludesfile	Имя файла; каждая строчка этого файла понимается как шаблон для исключения из поиска	Нет
casesensitive	Определяет влияние регистров для шаблонов (**true|yes|on** или **false|no|off**)	Нет; по умолчанию **true**

Примеры:

```
<dirset dir="${build.dir}">
  <include name="apps/**/classes"/>
  <exclude name="apps/**/*Test*"/>
</dirset>
```

Группирует все каталоги с именем **classes**, найденные под **apps** поддиректорией **${build.dir}** директории, пропуская те, что имеют текст **Test** в своём имени.

```
<dirset dir="${build.dir}">
  <patternset id="non.test.classes">
    <include name="apps/**/classes"/>
    <exclude name="apps/**/*Test*"/>
  </patternset>
</dirset>
```

Делает то же самое, но была установлена ссылка на **<patternset>**.

```
<dirset dir="${debug_build.dir}">
  <patternset refid="non.test.classes"/>
</dirset>
```

Таким образом можно к ней обратиться.

FileSet

FileSet есть набор файлов. Эти файлы могут быть найдены в дереве каталогов, начиная с базовой директории и удовлетворяющие шаблонам. **FileSet** может находиться внутри некоторых задач или выноситься в проект с целью дальнейшего к нему обращения по ссылке.

Атрибут	Описание	Обязательность
dir	Корень каталогов этого **FileSet**	Один должен быть обязательно
file	Сокращение для определения **Fileset** из одного файла	
includes	Список шаблонов (через запятую или пробел) для каталогов, которые должны быть включены, если атрибут пропущен, все каталоги включаются	Нет
includesfile	Имя файла; каждая строчка этого файла понимается как шаблон для включения в поиск	Нет

excludes	Список шаблонов (через запятую или пробел) для каталогов, которые должны быть исключены, если атрибут пропущен, все каталоги включаются	Нет
excludesfile	Имя файла: каждая строчка этого файла понимается как шаблон для исключения из поиска	Нет
casesensitive	Определяет влияние регистров для шаблонов (**true|yes|on** или **false|no|off**)	Нет; по умолчанию **true**

Примеры:

```
<fileset dir="${server.src}" casesensitive="yes">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</fileset>
```

Группирует все файлы в каталоге ${**server.src**}, являющимися Java кодами и не содержащими текста **Test** в своём имени.

PatternSet

Шаблоны могут быть сгруппированы в наборы и позже использованы путём обращения по ссылке. **PatternSet** может находиться внутри некоторых задач или выноситься в проект с целью дальнейшего к нему обращения по ссылке.

Шаблоны могут определяться с помощью внутренних **<include>**, или **<exclude>** элементов или с помощью следующих атрибутов:

Атрибут	Описание
includes	Список шаблонов (через запятую или пробел) для каталогов, которые должны быть включены, если атрибут пропущен, все каталоги включаются
includesfile	Имя файла; каждая строчка этого файла понимается как шаблон для включения в поиск. Можно задавать несколько
excludes	Список шаблонов (через запятую или пробел) для каталогов, которые должны быть исключены, если атрибут пропущен, все каталоги включаются
excludesfile	Имя файла; каждая строчка этого файла есть шаблон для исключения из поиска. Можно задавать несколько

Параметры определённые как внутренние элементы **include** и **exclude**

Эти элементы определяют единичный шаблон включений или исключений.

Атрибут	Описание	Обязательность
name	Шаблон, который или включается, или исключается	Нет
if	Использовать этот шаблон, если параметр установлен	Нет
unless	Использовать этот шаблон, если параметр не установлен	Нет

Если брать шаблоны извне, то нужно использовать **includesfile/excludesfile** атрибуты или элементы.

Атрибут	Описание	Обязательность
name	Имя файла, который содержит шаблоны	Нет
if	Читать этот файл, только если параметр установлен	Нет
unless	Читать этот файл, только если параметр не установлен	Нет

Атрибут **patternset** может содержать внутри другой **patternset**.

Примеры:

```
<patternset id="sources">
  <include name="std/**/*.java"/>
  <include name="prof/**/*.java" if="professional"/>
  <exclude name="**/*Test*"/>
</patternset>
```

Включает файлы в подкаталоге **prof**, если параметру **professional** установлено некоторое значение.

Следующих два набора:

```
<patternset includesfile="some-file"/>
```

и

```
<patternset>
  <includesfile name="some-file"/>
<patternset/>
```

одинаковы.

```
<patternset>
  <includesfile name="some-file"/>
  <includesfile name="${some-other-file}"
                if="some-other-file"
  />
<patternset/>
```

Будет читать шаблоны из файлов, один из них только тогда, когда параметр **some-other-file** установлен.

Создание собственной задачи

Создаётся Java-класс, наследуемый от **org.apache.tools.ant.Task** или другого сходного с ним класса.

Для каждого атрибута пишется установочный метод. Он должен быть **public void** и принимать один-единственный параметр. Имя метода должно начинаться с **set**, предшествующего имени атрибута, с первым символом имени, написанным в верхнем регистре, а остальное в нижнем. Так, чтобы подключить атрибут с именем **file**, следует создать метод **setFile()**.

Если задача будет содержать другие задачи в качестве внутренних, класс должен наследоваться от **org.apache.tools.ant.TaskContainer**.

Для каждого внутреннего элемента указывается **create()**, **add()** или **addConfigured()** метод. Метод **create()** должен быть **public** методом,

который не принимает аргументов и возвращает **Object** тип. Метод **add()** (или **addConfigured()**) метод должен быть **public void** методом, который принимает единственный аргумент **Object** типа с конструктором без аргументов.

public void execute() – метод без аргументов, является главным методом в задаче, который генерирует **BuildException.**

В качестве примера можно создать собственную задачу, в результате выполнения которой будет выводиться сообщение в поток **System.out**. У задачи будет один атрибут с именем **message**.

```
package com.mydomain;
import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class MyVeryOwnTask extends Task {
      private String msg;

      // метод, выполняющий задачу
      public void execute() throws BuildException {
            System.out.println(msg);
      }
      // метод установки атрибута
      public void setMessage(String msg) {
            this.msg = msg;
      }
}
```

Чтобы добавить задачу в проект, следует удостовериться, что класс, который подключается, находится в **classpath**.

После добавления **<taskdef>**-элемента в проект элемент добавится в систему, используя предыдущую задачу в оставшемся **build**-файле.

```
<?xml version="1.0"?>

   <project name="OwnTaskExample" default="main" base-
dir=".">
     <taskdef name="mytask" class-
name="com.mydomain.MyVeryOwnTask"/>

     <target name="main">
       <mytask message="Hello World! MyVeryOwnTask
works!"/>
     </target>
   </project>
```

Чтобы использовать задачу повсеместно в проекте, нужно поместить **<taskdef>** внутри проекта, а не задачи. Следует использовать **classpath** атрибут **<taskdef>**, чтобы указать, откуда брать код задачи.

```
   <?xml version="1.0"?>
```

```
   <project name="OwnTaskExample2" default="main" base-
dir=".">

     <target name="build" >
       <mkdir dir="build"/>
       <javac srcdir="source" destdir="build"/>
     </target>

     <target name="declare" depends="build">
       <taskdef name="mytask"
           classname="com.mydomain.MyVeryOwnTask"
           classpath="build"/>
     </target>

     <target name="main" depends="declare">
      <mytask message="Hello World! MyVeryOwnTask works!"/>
     </target>
   </project>
```

В этой книге были рассмотрены основные возможности языка Java применительно к созданию приложений и распределенных систем. Ряд дополнительных возможностей остался вне поля зрения, поэтому читайте спецификации, которые можно загрузить по адресу:

```
java.sun.com/products/servlet/index.jsp
java.sun.com/products/jsp/index.jsp
java.sun.com/products/jdbc/index.jsp
java.sun.com/j2ee/1.4/index.jsp
```