

КПО

Ant, Maven

Лекция №9 (версия 1.0)

```
sent"/>  
fish.web.present
```

```
<!-- do not forg
```

```
oot}" else="{gfv
```

```
app.context-root]
```

```
resent">
```

```
b}"/>
```

Ant

Apache Ant (англ. ant — муравей и акроним — «Another Neat Tool») — утилита для автоматизации процесса сборки программного продукта. Является платформонезависимым аналогом утилиты make (в качестве «Makefile» применяется «build.xml»).

Ant был создан в рамках проекта Jakarta, сегодня — самостоятельный проект первого уровня Apache Software Foundation.

Первая версия была разработана инженером Sun Microsystems Джеймсом Дэвидсоном (James Davidson (англ.)русск.), который нуждался в утилите, подобной make.

Ant

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). Конкретный набор целей и их взаимосвязи зависят от специфики проекта.

Краткий список заданий

Javaс компиляция Java-кода

Copy копирование файлов

Delete удаление файлов и директорий

Move перемещение файлов и директорий

Replace замещение фрагментов текста в файлах

JUnit автоматический запуск юнит-тестов

Exec выполнение внешней команды

Zip создание архива в формате Zip

CVS выполнение CVS-команды

Mail отправка электронной почты

Xslt наложение XSLT-преобразования

Актуальная версия программы (1.8.0 rc1) содержит около 150 типов заданий

Простой сценарий сборки

```
<project name="simpleCompile"
```

```
  default="deploy" basedir=".">
```

```
  <target name="init">
```

```
    <property name="sourceDir" value="src" />
```

```
    <property name="outputDir" value="classes" />
```

```
    <property name="deployJSP" value="/web/deploy/jsp" />
```

```
    <property name="deployProperties" value="/web/deploy/conf" />
```

```
  </target>
```

```
  <target name="clean" depends="init">
```

```
    <delete dir="${outputDir}" />
```

```
  </target>
```

```
  <target name="prepare" depends="clean">
```

```
    <mkdir dir="${outputDir}" />
```

```
  </target>
```

```
  <target name="compile" depends="prepare">
```

```
    <javac srcdir="${sourceDir}" destdir="${outputDir}" />
```

```
  </target>
```

```
  <target name="deploy" depends="compile,init">
```

```
    <copy todir="${deployJSP}">
```

```
      <fileset dir="${jsp}" />
```

```
    </copy>
```

```
    <copy file="server.properties" tofile="${deployProperties}" />
```

```
  </target>
```

```
</project>
```

```
sent"/>  
fish.web.present
```

```
<!-- do not for
```

```
oot)" else="$gfv
```

```
app.context-root
```

```
resent">
```

```
b)"/>
```

Ant / Project

Первая строка содержит общую информацию о собираемом проекте.

```
<project name="simpleCompile" default="deploy" basedir=".>
```

Самые важные атрибуты элемента project это default (по умолчанию) и basedir (базовая директория).

Атрибут default указывает на target (задание), определенное для выполнения по-умолчанию.

```
sent"/>  
fish.web.present  
<!-- do not forg
```

```
oot}" else="$gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```

Ant / Command line

Ant работает из командной строки, поэтому вполне реально указать только подмножество необходимых задач из всего файла сборки. Например, запустив следующую команду:

```
% ant -buildfile simple.xml init
```

Запустив такую команду, обработчик Ant'a выполнит только задание (target) с атрибутом name, которого равно init. И так, в нашем примере заданием по умолчанию является deploy. Следующий пример команды запуска Ant выполнит именно задание указанное по умолчанию, так как нет указания в командной строке на какое-либо конкретное задание:

```
% ant -buildfile simple.xml
```

```
sent"/>  
fish.web.present  
<!-- do not for
```

```
}) else="s{gfv
```

```
app.context-root
```

```
resent">
```

```
b]"/>
```

Ant / Target

```
2 <project def  
3   <target r  
4     <proj  
5     <ava  
6     <ava  
7     <ava  
8     <tem  
9     <ech  
10  </target  
11 <target name="init">  
12   <target r  
13   <tem  
14 <target name="clean" depends="init">  
15   <!--  
16   <repl  
17   <  
18   <  
19 </re  
20 В общем случае элемент target содержит пять атрибутов:  
21 name, if, unless, depends и description. Обязательным  
22 атрибутом является name, когда как остальные —  
23 </re  
24 атрибутом является name, когда как остальные —  
25 <!--  
26 необязательные.  
27   <cond  
28   <  
29   </con  
30   <cond  
31   <  
32   </con  
33 </target>  
34 <target n  
35   <temp  
36   <copy
```

```
sent"/>  
fish.web.present  
  
<!-- do not forg
```

```
oot}" else="$gfv  
app.context-root]  
  
resent">  
b]"/>
```


Ant

Указывая значение атрибута `depends`, вы можете указать Ant'у, что данное задание зависит от других заданий и не может быть выполнено пока не выполнятся все задания из указанных в атрибуте. В приведенном выше примере, задание `clean` не запустится до тех пор, пока не завершится задание `init`. Атрибут `depends` может включать несколько значений имен заданий через запятую, тем самым указывая, что зависит от нескольких заданий.

Атрибуты `if` и `unless` дают вам возможность указать задания, которые выполнятся если указанное свойство установлено (в случае `if`) или наоборот, в случае `unless` не установлено. Вы можете использовать команду `available` для установки свойств, как показано в следующем примере, либо в командной строке.

```
<available classname="org.whatever.Myclass"  
property="Myclass.present"/>
```

```
sent"/>  
fish.web.present
```

```
<!-- do not for
```

```
ot})" else="$gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```

Ant / Property

Задание init из нашего примера содержит установку четырех свойств вида:

```
<property name="sourceDir" value="src" />
```

Очень часто property (свойствам) присваивают часто используемые директории или файлы. Атрибуты элемента property это пары name(имя) и value(значение). Установка свойств позволит вам логически подвязать необходимые директории или файлы вместо их прямого использования.

Если вам нужно сослаться на свойство с именем sourceDir где-либо позднее в файле, вы можете использовать следующий синтаксис, указывающий Ant'у подставить соответствующее значение установленное в элементе property ранее: `${sourceDir}`.

Ant / Javac

```
2 <project def
3   <target
4     <prop
5     <ava
6     <ava
7     <ava
8     <temp
9     <ech
10  </target>
11  <javac srcdir="${src.dir}"
12    <target
13      <temp
14      <copy
15      <!--
16      <re
17      debug="on"
18      </re
19      <re
20      deprecation="off"
21      optimize="on" >
22      </re
23      <!--
24      <!--
25      <!--
26      <!--
27      <!--
28      <!--
29      <!--
30      <!--
31      <!--
32      <!--
33      <!--
34      <!--
35      <!--
36      <!--
```

```
sent"/>
fish.web.present
```

```
<!-- do not forg
```

```
oot}" else="${gfv
```

```
app.context-root
```

```
resent">
```

```
b]"/>
```

Ant / Exec

```
2 <project def
3   <target r
4     <proj
5     <ava
6     <ava
7     <ava
8     <temp
9     <ech
10  </target>
11
12  <target r
13    <temp
14    <copy
15    <!--
16    <repl
17  <exec executable="cmd">
18    <arg value="/c"/>
19    <arg value="ant.bat"/>
20    <arg value="-p"/>
21  </exec>
22  </repl
23  </copy
24  <xml
25  <get
26  </get
27  </copy
28  </con
29  </con
30  <cond
31  <
32  </con
33  </target>
34  <target n
35  <temp
36  <copy
```

`<exec executable="cmd">`
`<arg value="/c"/>`
`<arg value="ant.bat"/>`
`<arg value="-p"/>`
`</exec>`

```
sent"/>
fish.web.present
```

```
<!-- do not forg
```

```
oot}" else="$gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```

Ant / Copy

```
2 <project def  
3   <target r  
4     <proj  
5     <ava  
6     <ava  
7     <ava  
8     <tem  
9     <ech  
10  </target>  
11  
12 <copy file="myfile.txt" tofile="mycopy.txt"/>  
13  
14 <copy  
15 <!--  
16 <repl  
17 <copy file="myfile.txt" todir="../some/other/dir"/>  
18  
19 </repl  
20 <repl  
21  
22 <copy todir="../dest/dir">  
23   <fileset dir="src_dir">  
24     <excl  
25     <excl  
26     <excl  
27     <excl  
28     <excl  
29     <excl  
30     <excl  
31     <excl  
32     <excl  
33     <excl  
34     <excl  
35     <excl  
36     <excl
```

```
sent"/>  
fish.web.present  
  
<!-- do not forg
```

```
<copy todir="../dest/dir">
```

```
<fileset dir="src_dir">
```

```
<exclude name="**/*.java"/>
```

```
</fileset>
```

```
</copy>
```

```
oot}" else="$gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```

Ant / Echo

```
2 <project def
3   <target r
4     <proj
5     <ava
6     <ava
7     <ava
8     <tem
9     <ch
10 <echo message="Hello, world"/>
11
12 <target r
13   <tem
14   <op
15   <echo message="Embed a line break:$
16   {line.separator}"/>
17
18 </repl
19
20 <repl
21
22 <echo>Embed another:${line.separator}</echo>
23
24 <xml:
25 </xml:
26 <delc
27 <temp
28 </cop
29
30 <ch
31
32 </cop
33
34 <target n
35   <temp
36   <copy
```

`<echo message="Hello, world"/>`

```
sent"/>
fish.web.present
<!-- do not forg
```

`<echo message="Embed a line break:$
{line.separator}"/>`

```
<echo>Embed another:${line.separator}</echo>
```

```
<echo>This is a longer message stretching over
two lines.
```

```
</echo>
```

Ant / Java

```
2 <project def
3   <target r
4     <proj
5     <ava
6     <ava
7     <ava
8     <tem
9     <echr
10  </target
11  <java classname="test.Main">
12    <target r
13      <arg value="-h"/>
14      <cop
15      <!--
16      <classpath>
17      <
18      <pathelement
19      location="dist/test.jar"/>
20      <
21      <pathelement path="$
22      {java.class.path}"/>
23      <!--
24      </classpath>
25      </xml
26      </classpath>
27      <cont
28      </java>
29      <con
30      <con
31      <
32      </con
33      </target>
34      <target n
35      <temp
36      <copy
```

```
<java jar="dist/test.jar"
    fork="true" failonerror="true"
    maxmemory="128m">
  <arg value="-h"/>
  <classpath>
    <pathelement
      location="dist/test.jar"/>
    <pathelement path="$
      {java.class.path}"/>
  </classpath>
</java>
```

```
sent"/>
fish.web.present
```

```
<!-- do not forg
```

```
oot}" else="$ {gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```

Maven

Apache Maven — фреймворк для автоматизации сборки проектов, специфицированных на XML-языке POM (англ. Project Object Model). Слово maven происходит из языка Идиш и означает примерно «собиратель знания».

В файлах проекта pom.xml содержится декларативное описание проекта, а не отдельные команды по сборке проекта. Все задачи по обработке файлов Maven выполняет через плагины.

При исполнении Maven проверяет прежде всего, содержит ли файл pom.xml все необходимые данные и все ли данные синтаксически правильно записаны.

POM.XML

```
<project>
  <!-- версия модели для POM-ов Maven 2.x всегда 4.0.0 -->
  <modelVersion>4.0.0</modelVersion>

  <!-- координаты проекта, то есть набор значений, который
        позволяет однозначно идентифицировать этот проект -->
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>

  <!-- зависимости от библиотек -->
  <dependencies>
    <dependency>

      <!-- координаты необходимой библиотеки -->
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>

      <!-- эта библиотека используется только для запуска и компилирования тестов -->
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

```
sent"/>
fish.web.present
```

```
<!-- do not forg
```

```
oot}" else="$ {gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```

Maven

Конфигурация включает имя проекта, его собственника и его зависимости от других проектов. Возможно, также, конфигурировать индивидуальные фазы процесса построения проекта (build process), реализованные плагинами. Например, можно конфигурировать плагин компилятора так, что он будет использовать определенную версию Java, или специфицировать упаковку проекта даже в случае негативного результата прохождения некоторых тестов.

Крупные проекты должны быть поделены на несколько модулей, или подпроектов, каждый со своим собственным POM. Можно написать затем корневой POM, через который все модули компилируются единой командой. POM-ы могут наследовать конфигурацию от других POM-ов.

Maven

Корневой каталог проекта: файл pom.xml и все дальнейшие подкаталоги

- src: все исходные файлы
- src/main: исходные файлы собственно для продукта
- src/main/java: Java-исходный текст
- src/main/resources: другие файлы, которые используются при компиляции или исполнении, например Properties-файлы
- src/test: исходные файлы, необходимые для организации автоматического тестирования
- src/test/java: JUnit-тест-задания для автоматического тестирования
- target: все создаваемые в процессе работы Мавена файлы
- target/classes: скомпилированные Java-классы

Maven

Большая часть функциональности Maven-а осуществляется плагинами. Плагин обеспечивает достижение ряда целей с помощью следующего синтаксиса:

mvn [имя плагина]:[имя цели]

Например, Java-проект может быть скомпилирован плагином-компилятором путем выполнения команды:

mvn compiler:compile

```
sent"/>  
fish.web.present
```

```
<!-- do not forg
```

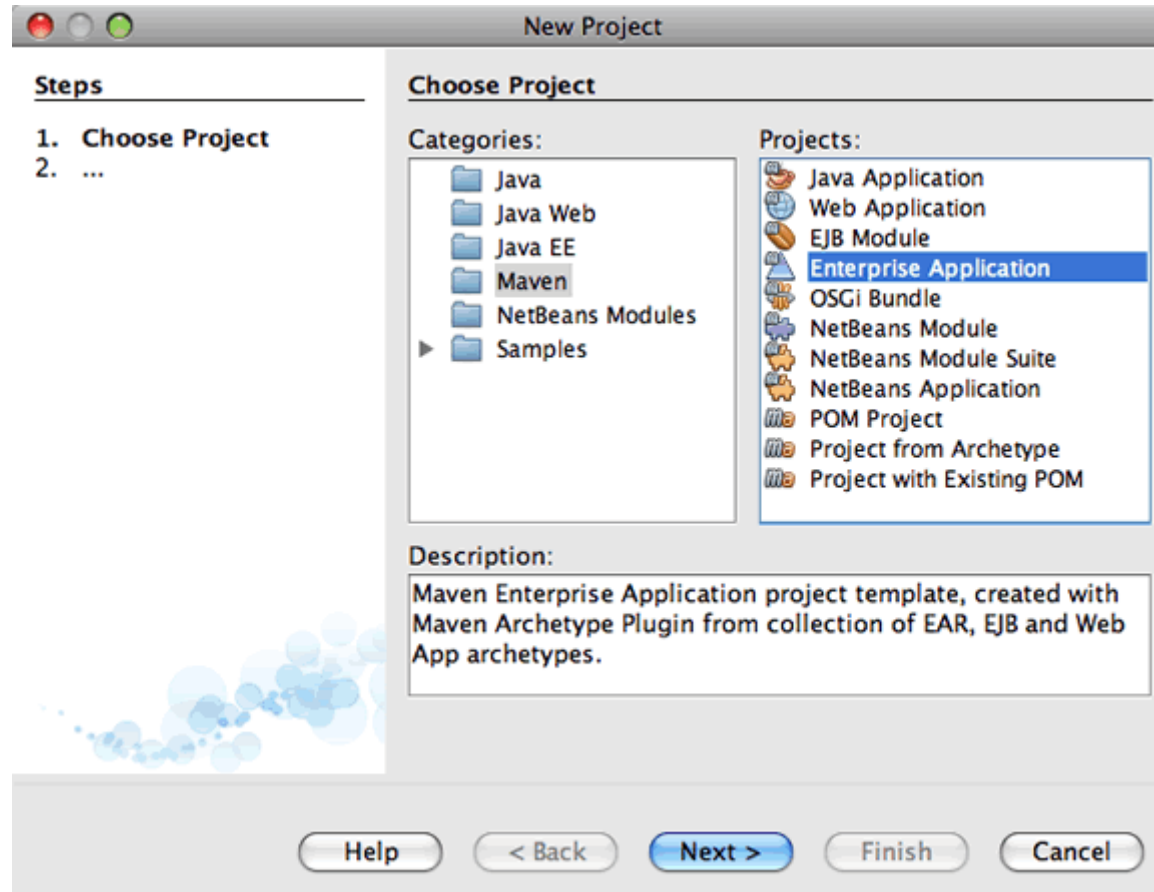
```
oot}" else="$gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```

NB Maven



```
sent"/>  
fish.web.present
```

```
<!-- do not forg
```

```
oot}" else="$gfv
```

```
app.context-root]
```

```
resent">
```

```
b}"/>
```

ЖЦ Maven

1. Создание темплейта и обработка ресурсов (archetype): На этой фазе разрешаются и, при необходимости, скачиваются из интернета зависимости.
2. Компиляция (compile)
3. Обработка тестовых ресурсов.
4. Компиляция тестов. (Тестирующие классы не передаются конечным пользователям.)
5. Тестирование (test)
6. Упаковка (package) Создание JAR- или WAR-файла.
7. Инсталляция проекта в локальном Maven-репозитории (install). Теперь он доступен как модуль для других локальных проектов.
8. Инсталляция в удаленном Maven-репозитории (deploy). Теперь стабильная версия проекта доступна широкому кругу разработчиков.

Maven

Если структура проекта соответствует стандартам Maven-а,
то команда

mvn package

откомпилирует все Java-файлы, запустит предусмотренные тесты, и упакует поставляемый программный код и ресурсы в `target/my-app-1.0.jar` (в предположении, что `artifactId` было определено как `'my-app'` и версия — как `1.0.`)

```
sent"/>  
fish.web.present  
<!-- do not forg
```

```
app.context-root
```

```
resent">
```

```
b]"/>
```

Maven

mvn -up

Обновить и сохранить локально плагины

mvn -o

Работать в режиме offline

```
sent"/>  
fish.web.present
```

```
<!-- do not forg
```

```
oot}" else="$ {gfv
```

```
app.context-root]
```

```
resent">
```

```
b]"/>
```


ИСТОЧНИКИ

1. Описание Maven (http://ru.wikipedia.org/wiki/Apache_Maven)
2. Creating an Enterprise Application Using Maven (<http://netbeans.org/kb/docs/javaee/maven-entapp.html>)
3. Описание Ant (http://ru.wikipedia.org/wiki/Apache_Ant)