Автоматизируйте процесс сборки Java программ с помощью Ant

Введение в мощный инструмент описания сборки кода на базе XML — Ant.

Build Processes

Майкл Цимерман

Ant (с англ. — муравей) — это мощный скриптовый инструмент, который позволит вам настроить процесс сборки приложений под свои требования, используя имеющийся в поставке набор задач (tasks), а также возможность выполнения гораздо более сложных действий путем расширения его функциональности. Эта статья является введением в Ant и его язык описания задач, который базируется на XML. Использование Ant позволит вам и вашей команде сконцентрироваться на бизнес логике и разработке кода, а обо всем остальном позаботиться Ant. *(3,000 слов в англ. оригинале)*

Замечание:

От переводчика:
На момент перевода этой статьи (начало марта 2001 года), стабильная версия Ant была под номером 1.3, автор же описывает версию под номером 1.1, поэтому этот перевод адаптирован под версию 1.3 (т.е. описаны некоторые новые возможности, убраны неработающие) и, соответственно, некоторые описания возможностей, появившихся после версии 1.1, отсутствуют в *оригинальной* статье.

Определенный процесс сборки это один из важнейших, но не повсеместно используемых элементов разработки программного обеспечения. По своей природе это накладное занятие, тесно связаное с разработкой. Определенный процесс сборки гарантирует, что при запуске сборки разрабатываемого проекта каждый раз проделываются одни и те же действия. С усложнением процесса сборки, например при разработке EJB приложений или схожих по сложности, становится крайне необходимо

определить определенный стандарт сборки. Вы должны как можно точнее определить, задокументировать и автоматизировать точный набор определенных шагов.

1. Зачем мне нужен определенный процесс сборки?

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 денег.

2. Что такое Ant?

Ant это платформо-независимый скриптовый инструмент, который позволит вам создавать сценарии сборки очень похожие на скрипты "make" в C или C++. Вы можете использовать большое количество уже имеющихся в поставке Ant скриптов сценариев без каких либо изменений. Некоторые из этих сценариев представлены в следующей таблице, но подробнее обсуждаются далее.

Вот несколько полезных команд, встроенных в поставку Ant:
(всего, на момент выпуска версии 1.3, стандартных команд было около 50-ти, плюс еще около 30-ти в optional pack)

Команда	Описание
Ant	Используется для запуска еще одного процесса Ant из текущего (например, для сборки подпроектов).
AntCall	Выполняет задачу (target), определенную в текущем файле сборки с возможностью передачи свойств (параметров).
Copy	Копирует файл[ы], директории.
Cvs	Позволяет работать с пакетами/модулями, извлеченными из репозитария CVS.

Delete	Удаляет либо один файл/директорию либо все файлы указанной директории и ее поддиректорий.
Move	Перемещает файл/директорию из одного места в другое.
Exec	Выполняет системную команду. Если указан атрибут os, тогда команда выполняется лишь в случае если Ant запущен под указанной операционной системой.
Get	Забирает файл по указанному URL.
Jar	Создает архивы Jar из набора файлов.
Java	Запускает Java класс в текущей ВМ (виртуальная машина в которой работает Ant) либо если необходимо порождает новую ВМ.
Javac	Компилирует дерево исходников под текущей ВМ (Ant).
Javadoc/Javadoc2	Генерирует документацию к коду с помощью утилиты javadoc.
Mkdir	Создает директорию.
Property	Устанавливает свойства проекта (property) (по имени и значению), или набор свойств (из файла или ресурса (resource)).
Rmic	Запускает компилятор rmic для определенного класса.
Tstamp	Устанавливает в текущем проекте значение свойств DSTAMP, TSTAMP, и TODAY.
Style	Пропускает набор документов через XSLT.

Хотя и существует множество разных инструментов для сборки программного обеспечения, Ant позволит вам сделать это легко и быстро в течение нескольких минут. В дополнение Ant позволяет легко расширить свои возможности путем наследования своих определенных классов. Небольшой пример, иллюстрирующий эту возможность, мы рассмотрим далее.

3. Что вам необходимо чтобы использовать Ant?

Вам необходимо проинсталлировать три составляющих: JDK, XML парсер, и Ant (смотрите в раздел Ресурсы для ссылок).

Очень часто XML парсер имеется в поставках сервлет контейнеров, в любом случае свободно распространяемый XML парсер java.sun.com будет вполне достаточен.

Инсталляция Ant заключается в загрузке, добавлении библиотек классов к переменной classpath и добавлении пути запускаемых файлов Ant к переменной path, а также в установке переменной окружения ANT_HOME.

4. Примерный сценарий работы

Этот небольшой пример должен помочь вам оценить Ant и пролить свет на его преимущества и способы использования.

Поскольку огромное количество текущих проектов на Java — это серверные приложения, я выбрал для примера именно такое приложение. Разработчики, работающие над серверными приложениями, обычно сталкиваются с компиляцией сервлетов, размещением (deployment) JSP файлов, размещением (deployment) HTML файлов, конфигурационных файлов, файлов с изображениями.

Обычная практика в таких случаях — это написание платформо-зависимых скриптов сборки на одном из языков доступных на конкретной операционной системе. Например разработчик, работающий на NT, может создать командный файл, в котором прописывает команды для компиляции проекта и затем его размещение. В то же время разработчик под Unix/Linux системы будет вынужден переписывать этот скрипт под возможности и особенности этих ОС, при этом постоянно синхронизируя изменения.

5. Ok. Теперь покажите как это работает на деле

Итак, я надеюсь, что смог убедить вас в необходимости использовать Ant и в простоте его инсталляции. Теперь я постараюсь пошагово показать насколько просто его использовать.

Простой сценарий сборки Ant (simple.xml)

```
<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>
```

Несмотря на то, что в этом примере необходимо многое обьяснить, первое на что вы должны обратить внимание — это на структура файла simple.xml. Этот файл не что иное, как правильно оформленный XML файл, содержащий сущность project (проекта) включающий несколько сущностей target (задания).

Первая строка содержит общую информацию о собираемом проекте.

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

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

Атрибут `default` указывает на target (задание), определенное для выполнения по-умолчанию. Дело в том, что 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`. Итак, 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
```

Атрибут `basedir` указывает на базовую директорию, от которой будут вычисляться относительные пути используемые далее в файле сборки. Каждый проект может иметь только один атрибут `basedir`, поэтому вы можете либо указывать полные пути либо разбивать сборку на несколько файлов сборки, в каждом из которых указывать свою базовую директорию.

Теперь переходим к элементам `target`. И видим две основные версии этого элемента:

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

В общем случае элемент `target` содержит пять атрибутов: `name`, `if`, `unless`, `depends` и `description`. Обязательным атрибутом является `name`, когда как остальные — необязательные.

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"/>
```

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'y подставить соответствующее значение установленное в элементе property ранее: `${sourceDir}`.

Две другие команды содержащиеся в рассмотренных заданиях:

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

Эти команды используются для уверенности в том, что в пути `outputDir` нет каких либо лишних файлов (или в данном случае в директрории `classes`). Первая команда удаляет все дерево с директорией `outputDir`. Вторая команда создает директорию снова.

В команде `copy` вы видите вложенный элемент <fileset>. Элемент fileset определяет группы файлов. Файлы, которые включаются в эти группы, находятся в поддиректориях относительно базовой, определенной в атрибуте basedir корневого элемента project. Для файлов можно указать набор масок (PatternSet) по которым будут отбираться необходимые файлы.

PatternSets (набор масок) может быть включен как вложенный элемент <patternset> в элемент Fileset. PatternSet может иметь атрибут id по которому можно ссылаться на данный набор из других мест файла сборки. Например:

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

Элементы include и exclude указывают на те маски, результат применения которых нужно включать и не включать в набор соответственно.

Следующая, интересная для разработчика, команда отвечает за компиляцию:

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

Команда `javac` требует наличие директории с исходниками *.java файлами и директории для складывания откомпилированных *.class файлов. Важно запомнить один момент: на момент запуска этой команды, директории с которыми она работает должны быть уже созданы, например командой `mkdir`. Ant не создает самостоятельно директории при выполнении этой команды, поэтому всегда создавайте их заранее с помощью команды `mkdir`.

После завершения задания `compile` проект скомпилен, следующее задание `deploy` выполнит копирование всех JSP файлов из исходной директории в директорию их необходимого размещения. Используя команду `copy` , вы можете скопировать как всю директорию из одного места в другое, так и для копирования одного файла свойств в директорию размещения.

Хотя объяснение содержания файла сборки и заняло несколько абзацев, вам должно быть понятно, что Ant очень легок в использовании. Используя приведенный пример в качестве отправной точки, вы сможете быстро адаптировать его под свои нужды. Используемые в примере команды обладают большей функциональностью, но о ней вы сможете узнать, обратившись к оригинальной документации поставки Ant.

6. Важные задачи

Чтение остальной документации оставим за вами, но я все-таки остановлюсь на двух наиболее используемых командах имеющихся в поставке Ant и представляющие дополнительные возможности без каких либо изменений.

6.1. Компиляция кода (включая EJB)

В приведенном ранее примере мы использовали только часть из возможных атрибутов команды `javac`. Если же вы рассмотрите эту команду подробнее, то заметите, что она имеет гораздо больше атрибутов, например, такие как deprecation, debug, или optimize, а также элементы указывающие на то какие файлы включать в компиляцию, а какие наоборот исключать.

```
<javac srcdir="${src.dir}"
           destdir="${build.classes}"
           classpath="${classpath}"
           debug="on"
           deprecation="off"
           optimize="on" >
      <include name="**/*.java"/>
      <exclude name="**/Script.java"
        unless="bsf.present" />
      <exclude name="**/version.txt" />
    </javac>
```

Вы можете использовать вложенные в `javac` команду элементы `include/exclude` для включения/исключения файлов подходящих под те или иные маски указанные в атрибуте `name` этих элементов. В приведенном примере мы хотим включить все файлы из всех поддиректорий имеющие расширение java и исключить все файлы с именем version.txt и не включать в компиляцию файлы Script.java если свойство `bsf.present` равно false.

Вы можете установить свойство `bsf.present` с помощью следующего задания: (смысл задания: идет поиск по classpath в поисках класса указанного в значении атрибута classname и возвращается результат поиска (true|false)):

```
<available property="bsf.present" classname="com.ibm.bsf.BSFManager" />
```

6.2. Запуск javadoc

Еще одна команда Ant позволит автоматизировать генерацию документации на код с помощью запуска утилиты javadoc:

```
<javadoc packagenames="${packages}"
             sourcepath="${basedir}/${src.dir}"
```

```
            destdir="${build.javadocs}"
            author="true"
            version="true"
            windowtitle="${Name} API"
            doctitle="${Name}"
            bottom="Copyright © 2000 GroupServe.
        All Rights Reserved."
    />
```

Атрибут packagenames определяет общие пакеты для включения. Атрибут `sourcepath` указывает на путь к исходным файлам. Команда `javadoc` также предоставляет атрибуты, давая вам возможность указать заголовок окна и документа. Вы также можете включить заметку о copyright в конце сгенерированных страниц, используя атрибут `bottom`.

7. А может ли Ant сделать вот э-э-это?

а данный момент, вы рассмотрели несколько задач сборки, которые Ant поможет вам автоматизировать. Эти задачи/команды уже включены в поставку Ant. Возможно, вы захотите изменить Ant и, например, добавить более сложные задачи как-то: сборка EJB и реализация удаленного управления. Некоторым из вас, возможно, захочется улучшить его логи или написать GUI.

Простым ответом на вопрос "А может ли Ant сделать вот э-э-это?" будет "Да, но вам, возможно, придется немного расширить его функциональность."

8. Расширяем Ant

Далее обсудим два расширения Ant. Первое улучшает составление отчетов, второе позволяет удаленно распространять код с помощью Ant.

8.1. Улучшение отчетности

Если вы хотите расширить функциональность Ant'a и добавить посылку сообщений по выполнению определенных шагов сборки или в процессе их выполнения, вы можете создать класс, прослушивающий Ant процессы как показано в следующем примере:

Вы можете создать класс, реализующий интерфейс `BuildListener`. Используя этот класс, вы сможете ловить любое событие `Listener`:

```
public void buildStarted(BuildEvent event);
    public void buildFinished(BuildEvent event);
    public void targetStarted(BuildEvent event);
    public void targetFinished(BuildEvent event);
    public void taskStarted(BuildEvent event);
    public void taskFinished(BuildEvent event);
    public void messageLogged(BuildEvent event);
```

Объект события `BuildEvent` имеет следующие методы, с помощью которых вы сможете получить информацию о текущем статусе сборки.

```
public Project getProject() ;
    public Target getTarget() ;
    public Task getTask();
    public String getMessage();
    public int getPriority();
    public Throwable getException();
```

Итак, если вам необходимо написать утилитку генерации отчетов по выполнению сборки, вам лишь необходимо создать класс, реализующий интерфейс `BuildListener` и обрабатывать события `BuildEvents` как вам угодно. Так как Ant запускает загрузчик классов, вам нужно указать ваш класс в командной строке. Например:

```
ant -listener org.apache.tools.ant.XmlLogger
```

Этот listener класс включен в поставку Ant и выводит логи сборки в формате XML в файл "log.xml".

8.2. Несколько машин

Теперь давайте рассмотрим как расширить сам процесс сборки. Для этого разработаем пример, в котором два файла копируются с одной машины на другую и затем они обрабатываются Ant'ом.

Нам нужно расширить Ant путем создания своего `Task` класса для удаленных команд

copy и ant. Далее разберем все немного поподробнее.

Класс RemoteTask наследует класс Task. RemoteTask реализует всю функциональность по созданию соединения между локальной и удаленной машинами. В нашем случае соединение будет на уровне сокетов.

Класс RemoteTask содержит следующее обьявление метода, который должен иметь каждый класс наследующий Task :

```
public void execute() throws BuildException
```

Ant-процессор вызывает этот метод после определения всех аттрибутов. Любой класс наследующий Task должен переопределять этот метод.

RemoteCopyTask выполняет необходимые действия для удаленного копирования. Операция копирования работает либо на локальной машине либо копирует с локальной на удаленную. На что необходимо обратить внимание в реализации RemoteCopyTask, так это на три метода для доступа (accessor methods) которые позволят создателю файла сборки определять три переменные как то имя, директория, тип файла который необходимо скопировать.

RemoteCopyTask, запущеный на локальной машине, создает объект Command. Объект Command загружает файл в массив байтов для подготовки к передаче. При выполнении этот объект сериализуется и передается на удаленную машину.

Объект RemoteAntHandler получает объект Command из ObjectInputStream. RemoteAntHandler, затем десериализует объект и определяет, какую команду нужно выполнить. На данный момент я упростил пример и включил обе команды в один и тот же обработчик как разные ветки выражения if. В идеале нужно использовать иную логику для эффективной обработки команд.

Так как полученная команда — это команда copy (копировать), обработчик запишет файл на диск в указанную директорию и с указанным именем в (эти данные есть в полученном объекте Command).

В дополнение к удаленной команде copy, я включил удаленную команду ant. В этом случае, локальная машина может выполнить команду ant на удаленной машине.

Я использую RemoteAntTask, которая расширяет объект RemoteTask. RemoteAntTask просто устанавливает значение command в Ant. В будущем

расширения этого задания будут включать дополнительные атрибуты команды `ant`, т.е. те, что имеет оригинальная команда.

Когда объект `RemoteAntHandler` получает и десериализует объект `Command`, он определяет что необходимо выполнить команду `ant`. Это реализуется путем порождения еще одного процесса `ant` на сервере. Зачем порождать еще один процесс? Из-за архитектуры кода Ant не реально вызывать `ant` в рамках той же Java ВМ. Поэтому я порождаю еще один процесс, используя объект `RunProcess`. Пожалуйста заметьте, что скрипт вызывающий `RemoteServer` предполагает наличие нескольких командных параметров, таких как директория размещения (deployment directory). Это было так сделано для уменьшения потенциальной опасности которую может привнести на удаленную машину посылаемая команда.

В заключительном шаге расширения Ant'a для выполнения удаленных команд необходимо интегрировать новые команды в сценарий сборки. Что и делает следующий файл:

```
<project name="foo" default="ant" basedir=".">

  <target name="init" >
      <taskdef name="remoteCopy" classname="local.RemoteCopyTask"/>
      <taskdef name="remoteAnt" classname="local.RemoteAntTask"/>
  </target>

  <target name="deploy" depends="init2">
      <remoteCopy machine="machinename.groupserve.com" port="9090"
directory="e:\deve\ant\article\deploy" filetype="text"
filename="build.xml" />
      <remoteCopy machine="machinename.groupserve.com" port="9090"
directory="e:\deve\ant\article\deploy" filetype="binary"
filename="app.jar" />
  </target>

  <target name="ant" depends="deploy">
      <remoteAnt machine="machinename.groupserve.com" port="9090" />
  </target>
</project>
```

Первая интересующая нас строка:

```
<taskdef name="remoteCopy" classname="local.RemoteCopyTask" />
```

Эта строка описывает команду `taskdef`, которая подвязывает имя `remoteCopy` с классом находящимся в файле `local.RemoteCopyTask`. С этого момента я могу вызывать команду `remoteCopy`, и мой код должен запуститься. Например:

```
<remoteCopy machine="machinename.groupserve.com" port="9090"
directory="e:\deve\ant\article\deploy" filetype="text"
filename="build.xml"/>
```

`remoteCopy` команда запустит `RemoteCopyTask` который законнектится к машине/порт и скопирует указанный файл. В своих свойства идентифицирующие `RemoteAntServer`. Также необходимо заметить, что те самые три метода доступа к свойствам устанавливают значения, взятые из атрибутов команды. Присвоенные атрибутам значения конвертируются в вызовы соответствующих методов доступа.

Команда `remoteAnt` определяется похожим `taskdef` объектом:

```
<taskdef name="remoteAnt" classname="local.RemoteAntTask"/>
```

Эта строка запускает команду `ant` на удаленной машине:

```
<remoteAnt machine="machinename.groupserve.com" port="9090" />
```

Снова обратите внимание, что для запуска задания необходимо указать в командной строке входные параметры.

Для запуска этого примера, выполните следующие шаги:

1. Запустите ant -buildfile build.xml для сборки кода на локальной машине
2. Скопируйте файл app.jar на удаленную машину
3. Запустите remote.bat на удаленной машине
4. Отредактируйте файл build.xml изменив адрес машины (по умолчанию 127.0.0.1) и порт соответственно с вашими установками.
5. Запустите снова на клиенте ant -buildfile build.xml для запуска сборки и размещения сначала на локальной, а потом на удаленной машине

Если у вас выполнятся все шаги правильно, тогда вы заметите что файлы app.jar и build.xml скопировались на удаленную машину и затем был запущен ant с файлом сборки build.xml там же (на удаленной машине).

От переводчика: Приведенный пример работает под клонами Unix и версией Ant 1.1.

Для его работы под другими ОС и последними версиями Ant необходимо заменить неиспользуемые в новых версиях Ant команды в корневом build.xml на их новые аналоги (вы найдете их в документации по Ant), а также заменить везде в коде знак "/", используемый автором в качестве разделителя путей файловой системы на константу "File.separator" определенную в пакете java.io.

9. Заключение

Итак, дочитав до конца статьи, что вы узнали нового ?

Самый важный момент состоит в том, что вы поняли о важности определенного процесса сборки для ваших проектов, а в случае Ant, о простоте реализации этого процесса. По моему личному мнению, Ant очень легкий и гибкий инструмент с простым XML подобным файлом конфигурации, множеством встроенных команд. Если что-то вас не устраивает вы всегда сможете расширить его возможности.

Пожалуйста, просмотрите документацию к Ant, что бы узнать об остальных его командах и встроенных возможностях. Я надеюсь, что дал вам стартовые понятия о работе Ant и о способах его использования. Также думаю, что после прочтения этой статьи вы обязательно будете использовать Ant в вашем процессе разработки.

10. Об авторе

Майкл Цимерман — руководитель отдела исследований и разработки при GroupServe, г. Вашингтон, телекоммуникационной фирме специализирующейся на разработке интернет-ориентированных приложений.

11. Ресурсы

- Исходный код программ для данной статьи: http://www.javaworld.com/ jw-10-2000/ANT/ant.zip
- Альтернативный источник исходного кода: http://www.itpath.com/articles/ant.zip
- JDK и XML парсер можно загрузить отсюда: http://java.sun.com/
- Версию 1.1 Ant можно загрузить:

http://jakarta.apache.org/ builds/ant/release/v1.1/bin/

- "Benefit from Platform-Independent Builds," Sanjay Mahapatra (*JavaWorld,* August 2000) статья о процессе сборки и Ant: http://www.javaworld.com/javaworld/ jw-08-2000/jw-0804-builds.html
- По этим ссылкам вы сможете найти другие статьи Майкла Цимермана опубликованные на *JavaWorld*:
  - "Secure a Web Application, Java-Style," (April 2000): http://www.javaworld.com/javaworld/ jw-04-2000/jw-0428-websecurity.html
  - "Building a Java Servlet Framework Using Reflection, Part 1," (November 1999): http://www.javaworld.com/javaworld/ jw-11-1999/jw-11-servlet.html
  - "Building a Java Servlet Framework Using Reflection, Part 2," (February 2000): http://www.javaworld.com/javaworld/ jw-02-2000/jw-02-servlets2.html
  - "Smarter Java Development," (August 1999): http://www.javaworld.com/javaworld/ jw-08-1999/jw-08-interfaces.html

View the original article at: http://www.javaworld.com/ jw-10-2000/jw-1020-ant.html