Программирование на языке Java. Часть 2. Лекция 4

Курс для самостоятельного изучения

Темы лекции

- Сегодня нет программирования!
- Средства управления проектами:
  - Автоматизация процесса компиляции
  - Средства управления исходным кодом
- Главные задачи:
  - Улучшить структуру проекта
  - Автоматизировать процесс компиляции проекта
  - Поместить исходный код в систему контроля версий

Процесс компиляции проекта

- Для компиляции проекта требуется выполнить несколько операций:
  - Компиляция кода, кода модульных тестов
  - Запуск javadoc для генерации документации API
  - Запуск модульных тестов и проверка их результатов
- Здесь много работы!
  - Автоматизация этого процесса сделает его быстрее и проще
- Текущая структура проекта довольно запутана!
  - Код программы и тестов лежит в одном каталоге
  - Там же находятся библиотеки и .class файлы!

Apache Ant

- Ant это платформо-независимый инструмент для компиляции программ
- Написан целиком на Java
- Ему нужен файл build.xml с описанием процесса компиляции
- Модульная архитектура с большим количеством задач компиляции
  - Компиляция исходных кодов Java
  - Запуск javadoc
  - Запуск тестовых пакетов JUnit или TestNG и генерация отчетов
  - Генерация кода (например для проектов J2EE)
  - Перемещение/копирование/удаление файлов, создание/удаление каталогов
  - Отправка электронной почты или другие виды уведомлений
  - Взаимодействие с репозиториями исходного кода

Пример файла build.xml

```
<project name="myproject" default="compile" basedir=".">
    <!-- Global properties used in build -->
    <property name="srcDir" location="src" />
    <property name="buildDir" location="build"/>
    <property name="buildClassesDir"
          location="${buildDir}/classes"/>
    <target name="-init"> <!-- Initialization target -->
          <tstamp/>
          <mkdir dir="${buildDir}" />
    </target>
    <target name="compile" depends="-init"
          description="Build the project sources.">
          <mkdir dir="${buildClassesDir}" />
          <javac destdir="${buildClassesDir}">
                <src path="${srcDir}" />
          </javac>
    </target>
</project>
```

Свойства Ant используются для того чтобы собрать значения настроек в одном месте.

Цели могут содержать зависимости. Они определяют список задач которые следует выполнить

Запуск Ant

- Исполняемый файл Ant называется ant
- Для компиляции цели по умолчанию надо запустить ant без параметров
  - Цель по умолчанию указана в build.xml
- В командной строке можно явно указать цель (цели)

```
ant clean test doc
```

- Можно задать другие параметры
  - Подробный вывод: -v или -verbose
  - Задать свойства Java: -DpropName=value
  - И многое другое!

Свойства Ant

- Свойства это просто пары имя-значение
  - И имя и значение это строки
  - Их можно указывать в начале файла
  - Их можно задать внутри задачи
- Значение свойства извлекается так: ${propName}
- Пример:

```
<property name="buildDir" value="build" />
<property name="codegenDir" value="${buildDir}/codegen" />
```

- Значение свойства можно задать только один раз!
  - Если свойство указано где либо еще оно молча игнорируется
  - (Для того чтобы увидеть, где задаются свойства или где они заданы по ошибке несколько раз запустите ant –verbose)

Свойства Ant (2)

- Хороший пример:

```
<target name="debug" description="Set up for debug build">
<property name="java.debug" value="on" />
<property name="java.opt" value="off" />
</target>
<target name="release" description="Set up release build">
<property name="java.debug" value="off" />
<property name="java.opt" value="on" />
</target>
<target name="compile" depends="debug">
<javac debug="${java.debug}" optimize="${java.opt}" ... />
</target>
```

  - По умолчанию, при компиляции используются настройки для отладки.
  - Для того чтобы переопределить эту настройку в командной строке введите:

```
ant release compile
```

Цели Ant

- Тэги <target> задают цели компиляции
  - У каждой цели есть имя:

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

- Цель также может иметь описание

```
<target name="compile"
  description="Compile the sources!">
```

- Имена начинающиеся с прочерка нельзя указывать в командной строке (цели только для “внутреннего использования”)

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

Зависимости целей

- Цели могут иметь зависимости
  - Ant выполняет анализ зависимостей перед компиляцией
  - Исполняет все задачи в правильном порядке

```
<target name="-init" />
<target name="clean" depends="-init" />
<target name="compile" depends="-init" />
<target name="test" depends="compile" />
```

  - Запускаем ant test
- Ant исполняет цели –init, compile, и test именно в этом порядке

Информация о проекте!

- Не знаете какие в проекте цели?

```
ant -projecthelp
```

  - Выводит список всех целей которые имеют описания
  - Заодно выводит описание которое вы поместили в начале файла build.xml
- Пример:

```
<project name="paint" default="compile"
  basedir=".">
<description>
A simple program for drawing images.
</description>
...
</project>
```

Структура каталогов проекта

- Итак, пока все у нас находится в одном каталоге
  - Исходные файлы проекта, исходные файлы тестов, .class файлы, …
- Гораздо лучше: использовать разные каталоги
  - Исходный код расположен в своей структуре каталогов
  - Создаваемые файлы (.class файлы, и др.) помещаются куда то еще!
  - Исходные файлы защищены от перезаписи во время компиляции
  - Облегчается процесс очистки: Просто стереть каталог с результатами компиляции!
- Аналогично, следует отделить исходные файлы тестов от исходных файлов проекта
  - Они не должны входить в окончательный пакет, поэтому их надо держать отдельно
- Все другие ресурсы, документы, картинки, и пр. должны находится в своем собственном каталоге

Пример структуры проекта

- src исходные файлы проекта
- lib библиотеки которые требуются проекту
- test исходные файлы тестов
- res ресурсы: изображения, тексты, конфигурация, …
- doc проектная документация, инструкции (не javadocs)
- build сюда помещаются результаты компиляции
  - codegen сгенерированные файлы исходного кода Java (если есть)
  - classes . class файлы созданные javac
  - Javadoc сгенерированная документация API
  - Tests скомпилированные javac тестовые классы
  - results логи тестов
  - создаваемые файлы jar могут оставаться в каталоге build

Ant и структура каталогов проекта

- С помощью свойств Ant можно указать каталоги в начале файла build.xml

```
<property name="srcDir" location="src" />
<property name="buildDir" location="build"/>
<property name="buildClassesDir"
   location="${buildDir}/classes"/>
```

- Ранее объявленные свойства Ant можно использовать для того чтобы указать пути к подкаталогам
- В целях Ant, следует указывать каталоги с помощью свойств

```
<target name="compile" depends="debug">
   <mkdir dir="${buildClassesDir}" />
   <javac destdir="${buildClassesDir}"
    classpathref="libs.path">
    <src path="${srcDir}" />
   </javac>
</target>
```

Общие концепции и типы

- Ant имеет несколько концепций которыми пользуются большинство задач
- FileSet: группа файлов в каталоге
  - Задается с помощью элемента <fileset>
    - Базовый каталог FileSet обычно задается атрибутом dir
  - Поддерживает очень гибкую систему масок
    - Можно включать или исключать файлы по заданному шаблону
- Очень простой пример:
  - file-set всех исходных кодов тестов, кроме тех которые расположены в subpackage foo, и еще не готовы:

```
<fileset dir="${testSrcDir}">
<include name="**/Test*.java" />
<exclude name="**/foo/**" />
</fileset>
```

Шаблон ** выбирает ноль или несколько уровней каталогов

- Многие задачи Ant могут работать с FileSet

Общие концепции и типы (2)

- Структуры содержащие пути:
  - Механизм для создания сложных путей к классам и других путей
  - Часто используется в задачах компиляции и запуска кода Java
- Пример: пути к классам

```
<classpath>
   <pathelement location="${libDir}/foo.jar" />
   <pathelement location="${buildClassesDir}" />
</classpath>
```

- Могут также содержать списки FileSet:

```
<classpath>
   <fileset dir="${libDir}" includes="*.jar" />
</classpath>
```

Общие концепции и типы (3)

- Можно создавать ссылки на пути
  - Для определения нескольких путей которые зависят друг от друга
- Пример:

  Один путь для запуска самого проекта, а другой путь для запуска тестов

```
<path id="libs.path">
   <fileset dir="${libDir}" includes="*.jar" />
...
</path>
<path id="test.path">
<path refid="libs.path" />
...
</path>
```

- Задачи ссылаются на такое с помощью атрибутов типа:

```
<javac classpathref="test.path" ... >
```

Обзор возможностей Ant

- Ant широко используется во многих проектах на языке Java!
- Он предоставляет много возможностей
  - Условная компиляция в зависимости от операционной системы
  - Задачи Ant реализованные с помощью скриптового языка
  - Конфигурация загружается из файла
  - Обновляет номера версий и заменяет значения в коде
  - Исполняет задания контроля версий
  - Обновляет вебсайты
  - Исполняет задания SSH/FTP
  - …
- http://ant.apache.org

Управление исходным кодом

- Вы работаете над большим программным проектом…
- Проблема 1: Вы испортили код
  - Нужно вернуться к предыдущей рабочей версии
- Проблема 2: Другие люди тоже работают над проектом
  - …возможно над тем же файлом что и вы
- Проблема 3: Централизованный источник информации о проекте?
- Возможно веб-сайт, на котором выложены текущие результаты тестов, последняя версия документации API и прочее.
- Система управления исходным кодом может решить все эти задачи и многие другие

Управление исходным кодом

- Основная идея:
  - Сохраняем все файлы проекта в репозитории
  - Репозиторий хранит информацию о всех изменениях файлов
  - Копии проекта извлекаются (check out) из репозитория
  - Каждый разработчик изолирован от изменений сделанных другими
- Изменения в файлах проекта загружаются (check in/commit) обратно в репозиторий, после того как они сделаны.
- Множественные изменения в одном файле совмещаются
  - Если это возможно автоматически, иначе вручную!

Распределенный контроль версий

- Новый тренд в системах контроля версий:
  - Центральный сервер репозториев не используется!
- Распределенная система контроля версий
  - Каждый пользователь имеет собственный локальный репозиторий
    - Загружается рабочая копия, она редактируется, затем выгружается обратно
  - Пользователи могут легко синхронизироваться с другими репозиториями
- Удобно для широко распределенного проектирования программного обеспечения
  - Например для проектирования программ open-source
- Реже используется при разработке коммерческих продуктов
  - Программные компании предпочитают хранить все на одном центральном сервере
  - Распределенную систему контроля версий можно использовать и в централизованной конфигурации

Системы контроля версий

- Коммерческие централизованные системы контроля версий:
  - Perforce, Visual SourceSafe, BitKeeper, …
- Свободно распространяемые централизованные системы контроля версий:
  - Subversion – написана как замена CVS
- Свободно распространяемые распределенные системы контроля версий:
  - Git – автор Линус Торвальдс
    - Используется для разработки ядра Linux, Eclipse, PostgreSQL, …
  - Mercurial (hg) – распределенная СКВ написанная на Python
    - Используется в проектах Python, vim, OpenOffice, GNU Octave, …
  - Bazaar – также написана на языке Python
    - Используется в проектах Ubuntu, GNU Emacs, MySQL, …

Использование Subversion

- Две основные команды Subversion:
  - svn
    - Программа которую разработчики используют для доступа к репозиторию
    - Может загружать и выгружать файлы, перемещать удалять и пр.
  - svnadmin
    - Инструмент для администрирования репозитория
    - Используется редко администратором репозитория
- Обе программы имеют набор команд
  - Пример svn checkout ...
  - Обе программы имеют команду help:
    - svn help или svnadmin help

Настройка репозитория

- Начинать надо с создания репозитория
  - Репозиторий содержит все настройки и файлы данных
  - Команда:

```
svnadmin create /path/to/repository
```

  - Путь можно указать абсолютный или относительный
- Subversion может использовать различные типы хранилищ
  - Файловую систему, или BerkeleyDB
  - По умолчанию используется файловая система

Доступ к репозиторию

- Subversion использует формат URL для адресации репозиториев
  - Если требуется можно работать через HTTP
- Для доступа к локальному хранилищу используйте префикс file:// URL
- Subversion поддерживает удаленный доступ
  - svn://… URL для доступа с сервера Subversion
  - Или, svn+ssh://… URL для доступа через SSH

Импорт исходного кода

- Исходный код проекта нужно импортировать в репозиторий
  - Это делает команда svn import
- Рекурсивно добавляет все дерево каталога в репозиторий
- В репозитории следует создать логичную структуру каталогов
  - Каждый проект (или суб-проект) должен иметь свой собственный каталог
  - Подкаталоги проектов должны также иметь хорошую структуру
- Для проекта Боггл:
  - boggle/src
  - boggle/test
  - И пр. (boggle/build не нужен!)
- Subversion позволяет вам при необходимости переместить файлы и каталоги
  - Обычно это нужно если вы ошиблись…

Импорт исходного кода (2)

- Перейдите в каталог с исходными файлами
  - Удалите файлы *.class, *~, и пр.
  - Эти файлы не нужно импортировать!
- Импортируйте дерево каталога в репозиторий
  - Обычно указывается имя суб-проекта

```
svn import file:///home/user/cs11/advjava/svnrepo
  \
boggle
```

  - Subversion добавляет файлы из локального каталога (и подкаталоги!) в подкаталог boggle в репозитории

Работа с проектом

- Теперь репозиторий становится централизованным хранилищем всех версий всех файлов проекта
  - Можно в любое время извлечь любую версию
  - Обычно нужна последняя версия
- Нужно получить локальную копию проекта
  - Локальную копию версии файлов
  - В этой копии можно делать изменения
  - Можно периодически синхронизировть сделанные изменения
  - Когда копия заработает выгрузите (check in) ее на сервер!

Загрузка файлов (check out)

- Для загрузки файлов:
  - Выполните команду svn checkout url
  - В URL указывается адрес репозитория и каталог внутри него
- Пример загрузки проекта Боггл из репозитория:

```
svn checkout \
file:///home/joe/advjava/svnrepo/boggle
```

  - Создает локальный каталог с именем boggle, и загружает в него файл проекта
- Для обновления локальной рабочей копии
  - svn update
  - Если вызвать из каталога рабочей копии, не нужно указывать URL!

Работа с локальными файлами

- Командой add можно добавить новые файлы
  - Из рабочей копии:

```
svn add path1 path2 ...
```

  - Можно добавлять каталоги
    - Subversion рекурсивно обрабатывает содержимое каталога
- Команда delete удаляет файлы
  - Снова из рабочей копии:

```
svn delete path1 path2 ...
```

- Команда move перемещает файлы

```
svn move frompath topath
```

Выгрузка изменений

- Изменения в рабочей копии должны быть выгружены на сервер, чтобы они стали видны всем остальным
  - включая результаты команд add/delete/move
- Subversion сначала проверяет что ваша рабочая копия содержит последнюю версию
  - Нельзя выгружать изменения до тех пор пока вы не получили последнюю версию
- Для выгрузки изменений выполняется команда

  ```
  svn commit
  ```

  - Можно, если требуется, указать файлы для выгрузки
- По умолчанию, операция выгрузки работает рекурсивно

Логи выгрузки

- Subversion перед выгрузкой изменений просит ввести сообщение лога выгрузки
  - В нем вы должны описать сделанные изменения
- Всегда даже для небольших изменений добавляйте это описание!
  - Другим людям нужно знать, что вы сделали
  - Вам тоже иногда нужно напомнить об этом
- Клиент Subversion запускает для вас редактор для этого
  - Можно указать используемый редактор в **переменной среды окружения SVN_EDITOR** (или **EDITOR)**
  - Для коротких сообщений, используйте ключ **–m “сообщение” в командной строке**

Отмена изменений

- Используйте команду svn revert для отмены изменений сделанных в локальной копии
  - Subversion сохраняет копии оригинальных файлов, потому операция не требует обращения к репозиторию
  - Нельзя восстановить все изменения (например, нельзя восстановить удаленные каталоги)
- Другой способ
  - Просто удалить рабочую копию и загрузите новую
- Для этого нужен доступ к репозиторию, поэтому он выполняется немного медленнее чем svn revert

Код в репозитории

- Всегда компилируйте и проверяйте свой код перед выгрузкой в репозиторий
  - Ваши ошибки будут мешать другим людям.
  - Версия в репозитории должна компилироваться и в идеале также хорошо работать.
- Обновляйте свою рабочую копию до последней версии хранящейся в репозитории
  - Это позволит избежать проблем из за рассинхронизации с процессом проектирования

Документация Subversion

- Веб-сайт Subversion:
  - http://subversion.tigris.org
- The Subversion Book (очень полезная!)
  - http://svnbook.red-bean.com
- Не забывайте о команде svn help!

Задание этой недели

- Создайте хорошую структуру каталогов проекта
- Создайте скрипт Ant для компиляции проекта Boggle
  - Создайте задачи для:
    - Удаления всех результатов компиляции
    - Компиляции файлов исходного кода
    - Компиляции тестов
    - Запуска тестов
    - Генерации документации Javadoc
- Загрузите ваш исходный код (и скрипт компиляции) в репозиторий Subversion
- На этот раз обойдемся без программирования! ☺