**Внимание:**

**Данная задача преподом придумалась только что, и еще не решалась… Будьте осторожны – может она и не решается совсем… ☺**

**Инструменты сборки приложений.**

**Задание-3.**

**Разработка собственного Task-а для Apache ANT.**

Друзья, согласитесь, открывая папку с заданиями по ANT-у и Maven-у, любой из нас ожидает увидеть внутри ровно две задачи – одну по сборке проекта ANT-ом, и вторую (*и вот тут интрига накаляется до предела, ну что же, что же это будет…*) – по сборке проекта Maven-ом. Однако две эти задачи Вами уже выполнены, а мы все равно продолжаем.

Позвольте представить Вам, скажем так, нестандартную задачу. Ну и мне кажется, достаточно интересную, с точки зрения учебного процесса. Она позволит Вам слегка развеяться и отвлечься от рутинного процесса выполнения домашних работ, как-то:

«Открыл Eclipse, создал болванку для Web-приложения, добавил индексную страницу, перешел на сервлет\JSP, отправил параметры и принял их в соседнем сервлете».

В этой задаче у Вас будет возможность почувствовать себя разработчиком фундаментальной логики. Не просто прикладным программистом, в сотый раз переписывающим один и тот же участок кода, в надежде, что наконец-таки это было последнее изменение требований заказчика, который все просит и просит *«еще немного поиграть с формой квадрата»*, но художником, который просто пишет. Пишет ни для кого и одновременно для всех! Художником, который понимает, что уже наследующий день после того, как он загрузит свой API в интернет, он, пусть и не увидит, и не услышит, но обязательно почувствует миллионы «Спасиб»- ов, «Respect»-ов и «Thanks Bro»-вов со всех уголков планеты, где есть люди, хоть сколь-нибудь способные оценить прелесть хорошего кода для Java-платформы. В конце концов, мы получим возможность почувствовать себя людьми, не согласными довольствоваться тем, что уже есть, и способными выйти за границы дозволенного, потому что сейчас мы будем расширять функциональные возможности Apache ANT. Или, если совсем коротко:

Мы будем писать свой собственный Task для Apache ANT.

Однако перед началом задания стоит потратить еще немного ***виртуальной*** бумаги (*кстати, не вздумайте распечатывать это на бумагу реальную, поверьте, это не стерпит даже она!*) и рассказать об обратной стороне медали для текущего задания.

Вся пафосная тирада, выплеснутая на Вас ранее, была ни чем иным как слабой попыткой скрыть один простой, но не совсем приятный факт – практического смысла в написании ATN Task-ов нет почти никакого, и в реальной жизни, проекты, в которых необходимо расширять существующую функциональность ANT-а встречаются крайне редко. Основная причина здесь в том, что Task для ANT-а невозможно продать. Продать можно нефть, металлы, виртуальные деньги и даже устройство, которое *«позволит тебе при помощи смартфона раздевать одноклассниц и училку!»*.

И вот обидно, согласитесь… Черт с ними с нефтью и металлами… Но почему у устройства, которое раздевает училку на расстоянии, ***и которое физически невозможно***, гораздо больше шансов быть проданным, чем у полезной в общем-то вещи – JAR-файла, расширяющего функциональность одной из самых популярных утилит для сборки на Java платформе? Опять-таки, здесь все просто – у тех людей, которые с радостью покупают удаленные раздевашки, отправляют 100 долларов адвокату в Нигерию, чтобы тот смог начать процедуру переоформления на них наследства их несуществующего дядюшки-миллиардера или ***инвестируют*** (***инвестируют***!!! блин!!!!) в акции МММ, просто нет таких денег, чтобы инициировать разработку на Java платформе. И, как не скорбно об этом говорить, скорее всего, таких денег у них никогда не будет. А тем людям, у которых такие деньги есть - не нужно расширение ANT-а. Им нужны нефть, металлы… Ну дальше, уверен, Вы поняли ☺

Так вот, в результате получаем лишь малую прослойку проектов для тех компаний, которые настолько устойчивы в финансовом плане, что могут позволить себе ***тратить*** человеческие и временные ресурсы на расширение функционала бесплатных решений. Например, для того, чтобы автоматизировать некие специфические процессы в своих проектах или проектах своих заказчиков, которые выделяются из серой массы стандартных требований. В результате получаем простое заключение - расширение ANT-а – это, как правило, внутренний проект, с небольшим объемом требований, и непродолжительный по времени разработки. Соответственно, вероятность участия в таком проекте гораздо ниже, нежели в классическом прикладном программировании.

Окей, я уже потратил лист на то, чтобы рассказать, насколько это задание прекрасно, и два листа на демонстрацию того, что оно бесполезно. И даже находясь вдалеке от вас, все равно слышу раздраженный вопрос:

Дружище, задание где???!!!

Поэтому самое время перейти к формулировке текста задания. Помня о том, что написание собственных Task-ов занятие не слишком полезное, мы будем изо всех сил стараться эту полезность симулировать. А именно:

Мы будем разрабатывать ANT Task, который позволит создавать нам Maven-проекты ☺

Тем самым у Вас будет еще одна возможность потренироваться в привыкании к структуре каталогов Maven-проектов. Итак,

В нашей задаче необходимо разработать и протестировать ANT Task, который называется mavenprj. Task должен уметь создавать дерево каталогов и некоторые файлы по умолчанию для Maven-проекта того типа, который укажет разработчик.

Важным замечанием будет то, что каталоги должны создаваться Вами вручную при помощи классов из пакета java.io.\*.

Task должен поддерживать использование следующих атрибутов:

1. prjType.
2. groupId.
3. artifactId.
4. rootPackage.
5. baseDir.
6. servletVersion.
7. createWebXml.
8. createIndexPage.
9. createMainClass.
10. createTestFolder.

Опишем каждый из них подробнее.

|  |  |
| --- | --- |
| **prjType** | |
| Назначение атрибута | Задает тип создаваемого проекта – JSE или JEE |
| Обязательный? | Да. |
| Допустимые значения | * JSE * JEE |
| Значение по умолчанию | Нет. |
| Примечания. | 1. Если атрибут не указан, сборка должна завершиться с ошибкой (BUILD FAILED). 2. Если указан тип JSE, то должно быть создано дерево каталогов, которое соответствует архетипу maven-archetype-quickstart. (см. ниже). 3. Если указан тип JSE, то должно быть создано дерево каталогов, которое соответствует архетипу maven-archetype-webapp. (см. ниже). |

|  |  |
| --- | --- |
| **groupId** | |
| Назначение атрибута | Аналог параметра -DgroupId в Maven. |
| Обязательный? | Да. |
| Допустимые значения | Любое корректное имя. |
| Значение по умолчанию | Нет. |
| Примечания. | 1. При помощи этого параметра разработчик задает имя группы для создаваемого проекта. 2. В имени группы **не может быть** пробельных символов. 3. Даже так, имя группы может состоять только из:    1. Букв    2. Цифр    3. Символов точка (“.”) 4. Первым символом в имени группы должна быть буква. 5. При некорректном указании имени группы сборка должна завершиться с ошибкой (BUILD FAILED). |

|  |  |
| --- | --- |
| **artifactId** | |
| Назначение атрибута | Аналог параметра -DartifactId в Maven. |
| Обязательный? | Да. |
| Допустимые значения | Любое корректное имя. |
| Значение по умолчанию | Нет. |
| Примечания. | 1. При помощи этого параметра разработчик задает имя артефакта для создаваемого проекта. 2. В имени артефакта **не может быть** пробельных символов. 3. Даже так, имя артефакта может состоять только из:    1. Букв    2. Цифр 4. Первым символом в имени артефакта должна быть буква. 5. При некорректном указании имени артефакта сборка должна завершиться с ошибкой (BUILD FAILED). |

|  |  |
| --- | --- |
| **rootPackage** | |
| Назначение атрибута | Имя корневого пакета для Java классов. |
| Обязательный? | Нет. |
| Допустимые значения | Любое корректное имя пакета. |
| Значение по умолчанию | groupId + “.” + artifactId. |
| Примечания. | 1. При помощи этого параметра корневой пакет для файлов исходного кода. 2. Если параметр не указан, используется значенеие по умолчанию (см. выше). 3. Если значение указано, Ваш Task должен обеспечить валидацию имени пакета. 4. При некорректном указании имени пакета сборка должна завершиться с ошибкой (BUILD FAILED). |

|  |  |
| --- | --- |
| **baseDir** | |
| Назначение атрибута | Директория, в которой будет создано приложение. |
| Обязательный? | Нет. |
| Допустимые значения | Любой корректный путь. |
| Значение по умолчанию | “.”, что соответствует текущему каталогу. |
| Примечания. | Нет. |
| **servletVersion** | |
| Назначение атрибута | Версия Servlet API для создаваемого приложения. |
| Обязательный? | Нет. |
| Допустимые значения | Любая допустимая версия. На момент создания задания, это:   * 2.2 * 2.3 * 2.4 * 2.5 * 3.0 * 3.1 |
| Значение по умолчанию | Последняя версия – 3.1. |
| Примечания. | 1. Данный атрибут может быть использован только для типа приложения «JEE». Если он используется для «JSE» приложения, сборка должна завершиться с ошибкой (BUILD FAILED). 2. При указании некорректного значения версии Servlet API сборка должна завершиться с ошибкой (BUILD FAILED). 3. **Важно**! Для каждой версии должен быть сгенерирован **корректный** web.xml файл. Список файлов web.xml для всех версий Servlet API Вы можете найти [здесь](http://skillsimprover.com/webxmls.htm), или в архиве webxmls.zip текущего задания. |

|  |  |
| --- | --- |
| **createWebXml** | |
| Назначение атрибута | Индикатор того, должен ли в приложении генерироваться Deployment Descriptor – **web.xml**. |
| Обязательный? | Нет. |
| Допустимые значения | * true * false |
| Значение по умолчанию | true. |
| Примечания. | 1. Данный атрибут может быть использован только для типа приложения «JEE». Если он используется для «JSE» приложения, сборка должна завершиться с ошибкой (BUILD FAILED). 2. Файл web.xml стал необязательным, начиная с версии **Servlet API 2.5**. Если разработчик указывает более раннюю версию и задает текущему атрибуту значение «false», сборка должна завершиться с ошибкой (BUILD FAILED). |

|  |  |
| --- | --- |
| **createIndexPage** | |
| Назначение атрибута | Индикатор того, должна ли в приложении генерироваться индексная страница. |
| Обязательный? | Нет. |
| Допустимые значения | * true * false |
| Значение по умолчанию | true. |
| Примечания. | Данный атрибут может быть использован только для типа приложения «JEE». Если он используется для «JSE» приложения, сборка должна завершиться с ошибкой (BUILD FAILED). |

|  |  |
| --- | --- |
| **createMainClass** | |
| Назначение атрибута | Индикатор того, должен ли в приложении генерироваться Главный класс (класс с методом main()). |
| Обязательный? | Нет. |
| Допустимые значения | * true * false |
| Значение по умолчанию | true. |
| Примечания. | Данный атрибут может быть использован только для типа приложения «JSE». Если он используется для «JEE» приложения, сборка должна завершиться с ошибкой (BUILD FAILED). |

|  |  |
| --- | --- |
| **createTestFolder** | |
| Назначение атрибута | Индикатор того, должен ли в приложении генерироваться каталог для тестов (каталог test). |
| Обязательный? | Нет. |
| Допустимые значения | * true * false |
| Значение по умолчанию | true. |
| Примечания. | Нет. |

Описав все возможные атрибуты нашего Task-а, стоит еще немного времени уделить тем файлам, которые Ваша программа должна сгенерировать для создаваемого проекта. Все эти файлы можно разделить на три группы:

**1. Файлы, которые должны генерироваться для любого типа проектов.**

И это только файл **pom.xml**, который должен располагаться в корневом каталоге проекта и иметь корректное содержимое. Получить примеры таких файлов Вы сможете, если выполните команду mvn archetype:generate для архетипов maven-archetype-quickstart или maven-archetype-webapp.

**2. Файлы, которые должны генерироваться для JSE проектов.**

Здесь уже нужно генерировать два файла.

**Главный класс приложения**. Это Java класс с методом main(), который должен располагаться в корневом пакете проекта и иметь такое же имя, как и artefactId (см. выше).

Так же обратите внимание, что необходимо хоть и минимально, но соблюсти Code Conventions. А это значит, что даже если имя артефакта начинается со строчной буквы,

первая буква в имени главного класса все равно должна быть прописной.

В самом методе main() должна быть одна строка кода:

System.out.println(“Hello From <имя\_главного\_класса>”);

Последний момент, на который здесь стоит обратить внимание – главный класс должен генерироваться только тогда, когда значение атрибута createMainClass равно «true».

Второй файл – это **файл манифеста** – MANIFEST.MF. Для этого файла мы не вводили специального атрибута, следовательно, для JSE проектов он должен генерироваться всегда. Файл должен располагаться в каталоге

src/main/resources/META-INF/MANIFEST.MF

И иметь содержимое по умолчанию. Я не имею права воровать у Вас удовольствие, а соответственно, оставляю Вам приятную возможность путем нетривиальных исследований узнать, каким именно должно быть это самое *«содержимое по умолчанию»*, а может даже и – *«А что такое файл манифеста?»* (ну, конечно же, нет – не будет у Вас такого вопроса, я пошутил, не обижайтесь ☺).

Нетривиальные исследования можно начать вот отсюда:

<http://en.wikipedia.org/wiki/JAR_(file_format)>

Людей, которые искренне считают, что Java информацию стоит потреблять только на специализированных сайтах (и это, несомненно, правильно), милости просим в «святая-святых» – Java Tutorial:

<http://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html>

Ну а тем, кто любит неожиданности, путешествия автостопом, да и вообще, живет под девизом – «Если не знать, что будет завтра – больше сюрпризов смог***ё***шь получить» – предлагаем свободное плавание:

<https://www.google.by/?gws_rd=cr,ssl&ei=I28bVNKvMajXyQOQhYDwBA#q=java+manifest+file>

**2. Файлы, которые должны генерироваться для JEE проектов.**

Финальная группа обязательных файлов, которые будут генерироваться для JEE приложения. Здесь Вы уже будете работать с тремя видами файлов.

Прежде всего, Вы должны позаботиться о генерировании **файла web.xml**, для той версии сервлетов, которая используется в текущем приложении. Не забываете о том, что этот файл должен генерироваться только тогда, когда значение атрибута createWebXml равно «true». Ну а о корректном местоположении этого файла я даже заикаться не стану – я не посмею Вас так оскорблять!

Также помните, пожалуйста, о том, что для каждой версии сервлетов должен быть сгенерирован соответствующий web.xml. Список файлов для всех версий Вы найдете в архиве webxmls.zip, радом с этим заданием.

Вторым файлом, который должен генерироваться для JEE приложения, является **индексная страница** – index.html. Внутри файла должен быть стандартный HTML документ с текстом в секции body:

<h1>Hello From <имя\_артефакта></h1>

И последний файл – это **манифест**, который уже был полностью описан в предыдущей секции.

Прочитав требования о генерировании файлов по умолчанию для того или иного вида проектов у Вас может возникнуть вопрос:

А какую стратегию мне избрать, чтобы наиболее эффективно решить поставленную задачу?

Тут я могу предложить две стратегии – одну «решение в лоб» и вторую – «решение не в лоб».

Тривиальное решение представляет собой набор Java-классов, которые просто создают файлы и напихивают в них необходимый набор символов, слов, строк – называйте, как хотите. Недостатки этого подхода, уверен, видны Вам и без меня – Hard Code в Java коде, сложности с поддержкой проекта и его расширением и т. д.

В качестве одного из нетривиальных решений я могу предложить Вам следующее – Вы укладываете все необходимые файлы вместе с классом (или классами) своего Task-a в один JAR файл. Здесь, правда, уместно будет заметить, что обычно все вспомогательные файлы укладываются в отдельный пакет, например – resources. После этого, код Вашего Task-а открывает эти файлы как потоки ввода, читает их содержимое, и записывает его в потоки вывода тех файлов, которые должны быть сгенерированы для проекта. Так же не забывайте о том, что в некоторых файлах Вам потребуется заменять некоторые стандартные заглушки реальными данными. Например, в HTML документе, строка

<h1>Hello From <имя\_артефакта></h1>

действительно должна содержать имя артефакта, который используется для генерирования приложения.

Ну и крайний момент в задании – как тестировать проделанную Вами работу? Что ж, я ожидаю получить от Вас JAR файл, в котором будет все необходимое, чтобы генерировать Maven-проекты при помощи ANT-а, уложить этот JAR в свой CLASS PATH и запустить build.xml, который попытается сгенерировать несколько различных проектов. Если все отработает, как я ожидаю, то можно переходить к рассмотрению исходного кода.

Ниже несколько тестов, которые я собираюсь использовать. Я приведу Вам примеры только для тестирования JSE приложений (и то, не все). Тестирование JEE проектов, пожалуйста, продумайте сами. В присланной Вами работе кроме исходников и JAR файла должен быть еще и build.xml файл, который, на Ваш взгляд, тестирует все аспекты разработанного Вами Task-a. Ну что же, успешной Вам разработки! Помните, Java-миру нужен Ваш код!

1. Создаем JSE проект в текущей директории:

<mavenprj prjType="jse"

groupId="com.skillsimprover"

artifactId="testJSE" />

2. Создаем JSE проект в указанной директории:

<mavenprj prjType="jse"

groupId="com.skillsimprover"

artifactId="testJSE"

baseDir="C:\TestJSE" />

3. Создаем JSE проект в текущей директории, но с нестандартным корневым пакетом:

<mavenprj prjType="jse"

groupId="com.skillsimprover"

artifactId="testJSE"

rootPackage="com.trainings.ant.task" />

4. Пытаемся создать JSE проект, но получаем BUILD FAILED и соответствующее сообщение об ошибке:

<mavenprj prjType="jse"

groupId="com.skillsimprover"

artifactId="testJSE"

servletVersion="2.5" />