Лабораторная №6

Тема: «Системы сборки проектов»

Цель

Целью настоящей лабораторной работы является знакомство и освоение практических навыков использования инструментов для сборки Java проектов.

Задание

В рамках лабораторной работы требуется разработать и реализовать веб-приложение на основе материалов прошлой лабораторной работы используя систему сборки проектов Maven. Проект генерируется на основе archetype.

Например: тема "интернет магазин". Из XML файла подгружается список товаров с описанием, ценой и указанием группы товара (техника, быт. химия и т.д.), количеством и пр. На web станице показывается список товаров, отчёт о общей стоимости, количестве товаров и иная информация рассчитываемая в рамках предыдущей лабораторной работы.

Содержание отчёта

Отчёт о выполнении работы должен включать в себя:

- титульный лист;
- вариант задания;
- краткое описание результатов

Теория

Maven

Maven - это инструмент для сборки Java проекта: компиляции, создания jar, создания дистрибутива программы, генерации документации. Простые проекты можно собрать в командной строке. Если собирать большие проекты с командной строки, то команда для сборки будет очень длинной, поэтому её иногда записывают в bat/sh скрипт. Но такие скрипты зависят от платформы. Для того чтобы избавиться от этой зависимости и упростить написание скрипта используют инструменты для сборки проекта.

Основные преимущества Maven

- Независимость от OS. Сборка проекта происходит в любой операционной системе. Файл проекта один и тот же.
- Управление зависимостями. Редко какие проекты пишутся без использования сторонних библиотек(зависимостей). Эти сторонние библиотеки зачастую тоже в свою очередь используют библиотеки разных версий. **Maven** позволяет управлять такими сложными зависимостями. Что позволяет разрешать конфликты версий и в случае необходимости легко переходить на новые версии библиотек.

- Возможна сборка из командной строки. Такое часто необходимо для автоматической сборки проекта на сервере (Continuous Integration).
- Хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты которые собираются с помощью maven. При этом зачастую проект настраивать не нужно - он сразу готов к дальнейшей разработке. Как следствие - если с проектом работают в разных средах разработки, то maven удобный способ хранения настроек. Настроечный файл среды разработки и для сборки один и тот же - меньше дублирования данных и соответственно ошибок.
- Декларативное описание проекта.

Установка

- Зайдите на официальный сайт maven в раздел загрузка (http://maven.apache.org/download.html) и скачайте последнюю стабильную версию.
- Распакуйте архив в инсталляционную директорию. Например в C:\Program Files\maven\ в Windows или /opt/maven в Linux
- Установите переменную окружения M2_HOME:
  - В Windows кликните правой кнопкой мыши на "мой компьютер" ->свойства->дополнительные параметры->переменные среды->системные переменные и там добавьте "M2_HOME" и " C:\Program Files\maven\" .
  - В Linux можно добавить строку "export M2_HOME=/opt/maven"в файл /etc/profile .
- Установите переменную окружения PATH В Windows в переменной PATH добавьте к списку директорий строку %M2_HOME%\bin". В Linux можно добавить строку "export PATH=$PATH:$M2_HOME/bin"в файл /etc/profile .
- Проверьте корректность установки, набрав в командной строке
  ```
  mvn -version
  ```

** Во многих дистрибутивах Linux, maven устанавливается автоматически, с помощью менеджера пакетов.*

Если что-то не работает

- Проверьте установлен ли у вас JDK.
  - Для этого наберите в консоли «java -version»
- Проверьте установлена ли переменная окружения JAVA_HOME
  - Если у вас Windows наберите в консоли: echo %JAVA_HOME%
  - Если у вас Linux наберите в консоли: echo $JAVA_HOME

Создание проекта по archetype

Давайте сгенерируем простое web приложение с помощью плагина maven-archetype:plugin.

```
mvn archetype:generate
```

В результате нам вывалится очень большой список архитипов, и maven предложит вам выбрать. Выбирать в таком списке довольно сложно, поэтому будет удобно если вы отфильтруете этот список. Наберите

```
maven-archetype-webapp
```

и maven предложит вам список из нескольких архитипов.

```
Choose archetype:
1: internal -> org.apache.maven.archetypes:maven-archetype-webapp (A simple Java
web application)
```

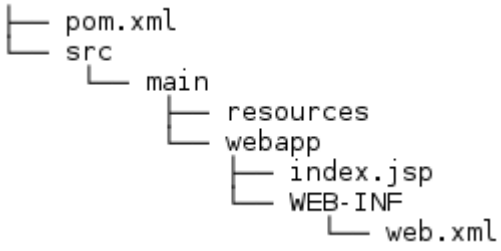
Выберите

```
 [1] org.apache.maven.archetypes:maven-archetype-webapp.
```

Напишите

```
groupId : ru.apache-maven
artifactId : webapptest
```

а остальные значения можно оставить по умолчанию.
в результате получится проект с такой структурой директорий:



Поздравляю, простейший веб-проект готов. Теперь можно:

- собрать с помощью
  ```
  mvn package
  ```
  в результате в директории target образуется testwebapp.war, который можно деплоить в сервлет контейнер, например в Apache Tomcat
- запустить напрямую:
  ```
  mvn tomcat:run
  ```
  В этом случае запустится tomcat и приложение будет сразу доступно по адресу http://localhost:8080/webapptest/

Что такое pom.xml

pom.xml - это основной файл, который описывает проект. Вообще могут быть дополнительные файлы, но они играют второстепенную роль.

Давайте резберём из чего состоит файл pom.xml

Корневой элемент и заголовок.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  ..............
</project>
```

Корневой элемент <project>, схема, которая облегчает редактирование и проверку, и версия POM.

Внутри тэга project содержится основная и обязательная информация о проекте:

```
  <!-- The Basics -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
```

В Maven каждый проект идентифицируется парой groupId artifactId. Во избежание конфликта имён, groupId - наименование организации или подразделения и обычно действуют такие же правила как и при именовании пакетов в Java - записывают доменное имя организации или сайта проекта. artifactId - название проекта. Внутри тэга version, как можно догадаться хранится версия проекта. Тройкой groupId, artifactId, version (далее - GAV) можно однозначно идентифицировать jar файл приложения или библиотеки. Если состояние кода для проекта не зафиксировано, то в конце к имени версии добавляется "-SNAPSHOT" что обозначает что версия в разработке и результирующий jar файл может меняться. <packaging>...</packaging> определяет какого типа файл будет создаваться как результат сборки. Возможные варианты pom, jar, war, ear

Также добавляется информация, которая не используется самим мавеном, но нужна для программиста, чтобы понять, о чём этот проект:

```
<name>powermock-core</name>
```
название проекта для человека

```
<description>PowerMock core functionality.</description>
```
Описание проекта

```
<url>http://www.powermock.org</url>
```
сайт проекта.

Зависимости

Зависимости - следующая очень важная часть pom.xml - тут хранится список всех библиотек (зависимостей) которые используюся в проекте. Каждая библиотека идентифицируется также как и сам проект - тройкой groupId, artifactId, version (GAV). Объявление зависимостей заключено в тэг <dependencies>...</dependencies>.

```
                <dependencies>
                        <dependency>
                            <groupId>junit</groupId>
                            <artifactId>junit</artifactId>
                            <version>4.4</version>
                            <scope>test</scope>
                        </dependency>
                        <dependency>
                            <groupId>org.powermock</groupId>
                            <artifactId>powermock-reflect</artifactId>
                            <version>${version}</version>
```

```
                        </dependency>
                        <dependency>
                            <groupId>org.javassist</groupId>
                            <artifactId>javassist</artifactId>
                            <version>3.13.0-GA</version>
                            <scope>compile</scope>
                        </dependency>
                    </dependencies>
```

Как вы могли заметить, кроме GAV при описании зависимости может присутствовать тэг <scope>. Scope задаёт для чего библиотека используется. В данном примере говорится, что библиотека с GAV junit:junit:4.4 нужна только для выполнения тестов.

Тэг <build>

Тэг <build> не обязательный, т. к. существуют значения по умолчанию. Этот раздел содержит информацию по самой сборке: где находятся исходные файлы, где ресурсы, какие плагины используются. Например:

```
        <build>
        <outputDirectory>target2</outputDirectory>
         <finalName>ROOT</finalName>
        <sourceDirectory>src/java</sourceDirectory>
           <resources>
               <resource>
                   <directory>${basedir}/src/java</directory>
                   <includes>
                   <include>**/*.properties</include>
                   </includes>
               </resource>
           </resources>
                <plugins>
                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-pmd-plugin</artifactId>
                        <version>2.4</version>
                    </plugin>
                </plugins>
            </build>
```

Давайте рассмотрим этот пример более подробно.

`<sourceDirectory>`
определяет, откуда maven будет брать файлы исходного кода. По умолчанию это src/main/java, но вы можете определить, где это вам удобно. Директория может быть только одна (без использования специальных плагинов)

`<resources>`
и вложенные в неё тэги <resource> определяют, одну или несколько директорий, где хранятся файлы ресурсов. Ресурсы в отличии от файлов исходного кода при сборке просто копируются . Директория по умолчанию src/main/resources

`<outputDirectory>`

определяет, в какую директорию компилятор будет сохранять результаты компиляции - *.class файлы. Значение по умолчанию - target/classes

```
<finalName>
```

- имя результирующего jar (war, ear..) файла с соответствующим типу расширением, который создаётся на фазе package. Значение по умолчанию — artifactId-version.

Maven плагины позволяют задать дополнительные действия, которые будут выполняться при сборке. Например в приведённом примере добавлен плагин, который автоматически делает проверку кода на наличие "плохого" кода и потенциальных ошибок.

Отображение информации на web-странице

JavaServer Pages (JSP) позволяют вам отделить динамическую часть ваших страниц от статического HTML. Вы, как обычно, пишете обычный код в HTML, используя для этого любую программу для создания Web страниц. Затем вы заключаете динамическую часть кода в специальные таги, большинство которых начинаются с "<%" и завершаются "%>". В качестве примера рассмотрим секцию JSP страницы, результатом которой будет что-то вроде "Спасибо за покупку Core Web Programming" по запросу с URL: http://host/OrderConfirmation.jsp?title=Core+Web+Programming:

```
Спасибо за покупку
<I><%= request.getParameter("title") %></I>
```

Вы даете вашему файлу расширение .jsp и размещаете там же, где должны размещаться обычные Web страницы. Хотя то что вы написали больше похоже на обычный HTML файл чем на сервлет, просто за кадром JSP страница преобразуется в обычный сервлет с статическим HTML который просто направляется в поток вывода, связанный с методом сервлета service. Обычно это происходит при первом запросе страницы, и разработчики могут сразу после установки сами выполнить этот запрос, если хотят чтобы первый реальный пользователь при обращении к странице не столкнулся с небольшой задержкой, вызванной трансляцией JSP страницы в сервлет и его последующей компиляцией и загрузкой. Также отметим что большинство Web серверов позволяет вам задавать ссылки (aliases), так что адрес URL, указывающий на HTML файл в действительности будет указывать на сервлет или страницу JSP.

Помимо стандартных HTML конструкций существуют еще три основных типа конструкций JSP, которые вы можете включить в страницу: элементы скриптов, директивы и действия. Элементы скриптов позволяют вам указать код на языке Java, который впоследствии станет частью в конечный сервлет, директивы дадут вам возможность управлять всей структурой сервлета, а действия служат для задания существующих используемых компонентов, а также для контроля поведением движка JSP. Для упрощения элементов скриптов, вы имеете доступ к нескольким заранее определенным переменным, таким, например, как переменная request, использованная в приведённом выше отрывке.

Синтаксис

Элемент JSP	**Синтаксис**	**Описание**	**Примечание**
Выражение JSP	*<%= выражение*	Выражение	Эквивалент на XML:

	%>	обрабатывается и направляется на вывод	<jsp:expression> expression </jsp:expression>. Заранее определенные переменные: request, response, out, session, application, config и pageContext (также доступны в скриплетах).
Скриплет JSP	*<% код %>*	Код добавляется в метод service.	Эквивалент на XML: <jsp:scriptlet> код </jsp:scriptlet>.
Объявление JSP	*<%! код %>*	Код добавляется в тело класса сервлета, вне метода service.	Эквивалент на XML: <jsp:declaration> код </jsp:declaration>.
Директива JSP page	*<%@ page att="значение" %>*	Директивы для движка сервлета с информацией об основных настройках.	Эквивалент на XML: <jsp:directive.page att="val"\>. Допустимые атрибуты (жирным выделены значения, принимаемые по умолчанию): import="*пакет.class*" contentType="*MIME-Type*" isThreadSafe="**true**|false " session="**true**|false" buffer="*размер*kb|none" autoflush="**true**|false" extends="*пакет.class*" info="сообщение" errorPage="*url*" isErrorPage="true|**false**" language="java"
Директива JSP include	*<%@ include file="url" %>*	Файл в локальной системе, подключаемый при трансляции JSP в сервлет.	Эквивалент на XML: <jsp:directive.include file="url"\>. URL должен быть относительным . Для подключения файла в процессе запроса а не в ходе трансляции используйте действие jsp:include.
Комментарий JSP	*<%-- комментарий --%>*	Комментарий; игнорируется при трансляции JSP страницы в сервлет.	Если вы хотите чтобы комментарий сохранился в конечном HTML, используйте обычный синтаксис HTML для описания комментариев: <-- комментарий-->.
Действие jsp:include	*<jsp:include page="относительный URL" flush="true"/>*	Подключает файл при запросе страницы.	Если вы хотите чтобы файл подключался в процессе трансляции страницы, используйте директиву page совместно с атрибутом include. **Внимание:** некоторые сервера требуют чтобы подключаемые файлы были в

			формате HTML или JSP, в зависимости от настроек сервера (обычно данное ограничение базируется на указании расширений файлов).
Действие jsp:useBean	*<jsp:useBean att=значение*/> or <jsp:useBean att=значение*> ... </jsp:useBean>*	Найти или создать Java Bean.	Возможные атрибуты: id="*имя*" scope="**page**|request|session|application" class="*пакет.class*" type="*пакет.class*" beanName="*пакет.class*"
Действие jsp:setProperty	*<jsp:setProperty att=значение*/>*	Устанавливает свойства bean, или явно, или указанием на соответствующее значение параметра, передаваемое при запросе.	Допустимые атрибуты: name="*имяBean*" property="*имяСвойства*|*" param="*имяПараметра*" value="*значение*"
Действие jsp:getProperty	*<jsp:getProperty name="ИмяСвойства" value="значение"/>*	Получение и вывод свойств bean.	
Действие jsp:forward	*<jsp:forward page="относительный URL"/>*	Передает запрос другой странице.	
Действие jsp:plugin	*<jsp:plugin attribute="значение" *> ... </jsp:plugin>*	Генерирует тэги OBJECT или EMBED, в зависимости от типа броузера, в котором будет выполнятся апплет использующий Java Plugin.	

Текст шаблона: Статический HTML

Как правило большую часть вашей JSP страницы составляет статический HTML, называемый текстом шаблона. Во всех отношениях (кроме одного) этот HTML выглядит как обычный HTML, использующий те же правила синтаксиса, и он просто "передается" клиенту сервлетом, создаваемым для обработки страницы. При этом не только сам HTML выглядит нормальным, он может создаваться с применением тех инструментов, которые вы ранее использовали при создании Web страниц. Я, например, при создании большинства JSP страниц для этого руководства использовал Allaire's HomeSite.

Единственным печальным исключением из правила что "текст шаблона передается в

неизменном виде" является ситуация, когда в тексте вы хотите отобразить последовательность символов "<%", для этого в тексте шаблона надо использовать сочетание символов "<\%".

Элементы скриптов JSP

Элементы скриптов JSP позволяют вам вставлять код на Java в сервлет, создаваемый из текущей JSP страницы. Существуют три формы:

Выражения, имеющие форму <%= выражение %>, которые обрабатываются и направляются на вывод, Скриплеты, имеющие форму <% код %>, которые вставляются в метод service сервлета Объявления, имеющие форму <%! код %>, которые вставляются в тело класса сервлета, вне существующих методов.
Все они в отдельности будут подробно рассмотрены далее.

Выражения JSP

Выражения JSP применяются для того чтобы вставить значения Java непосредственно в вывод. Для этого используется следующая форма:

```
<%= Выражение на Java %>
```

Выражения Java вычисляются, конвертируются в строку и вставляются в страницу. Эти вычисления происходит во время выполнения (то есть при при запросе страницы), а потому существует полный доступ к информации о самом запросе. Например, следующий код служит для отображения даты и времени запроса данной страницы:

```
Текущее время: <%= new java.util.Date() %>
```

Для того чтобы упростить эти выражения существует несколько заранее определенных переменных, которые вы можете использовать. Более детально они будут рассмотрены ниже, но тем не менее, рассматривая их применение при использовании выражений, приведем несколько наиболее важных:

- request, HttpServletRequest;
- response, HttpServletResponse;
- session, HttpSession ассоциируется с запросом (если таковой имеется);
- out, PrintWriter (буферизированный вариант типа JspWriter) используется для отсылки выводимых данных клиенту.

Пример:

```
Имя вашего хоста: <%= request.getRemoteHost() %>
```

И, наконец, авторы, использующие XML, могут применить альтернативный синтаксис для выражений JSP:

```
<jsp:expression>
Выражения на Java
</jsp:expression>
```

Помните, что элементы XML в отличие от HTML чувствительны к регистру. Поэтому убедитесь в том, что используете строчные символы.

Скриплеты JSP

Если вы хотите сделать что-то большее чем вставка простых выражений, скриплеты JSP дадут вам возможность вставить любой код в метод сервлета, который будет создан при обработке данной страницы. Скриплеты имеют следующий вид:

```
<% Код на Java %>
```

Скриплеты также имеют доступ к тем же автоматически определенным переменным, что и выражения. Поэтому, например, если вы хотите вывести что-нибудь на страницу, вы должны воспользоваться переменной out.

```
<%
String queryData = request.getQueryString();
out.println("Дополнительные данные запроса: " + queryData);
%>
```

Обратите внимание на то, что код внутри скриплета вставляется в том виде, как он записан, и весь статический HTML (текст шаблона) до или после скриплета конвертируется при помощи оператора print. Это означает что скриплеты не обязательно должны содержать завершенные фрагменты на Java, и что оставленные открытыми блоки могут оказать влияние на статический HTML вне скриплета. Например, следующий фрагмент JSP содержит смешанный текст шаблона и скриплеты:

```
<% if (Math.random() < 0.5) { %>
<B>Удачного</B> Вам дня!
<% } else { %>
<B>Не удачного</B> Вам дня!
<% } %>
```

после преобразования приведет к чему-то вроде:

```
if (Math.random() < 0.5) {
  out.println("<B>Удачного</B> вам дня!");
} else {
  out.println("<B>Не удачного</B> вам дня!");
}
```

Если вы хотите использовать последовательность символов "%>" внутри скриплета, вместо нее используйте "%\>". Эквивалентом <% Код %> для XML является:

```
<jsp:scriptlet>
Код
</jsp:scriptlet>
```

Объявления JSP

Объявления JSP позволят вам задать методы или поля, для вставки в тело класса сервлета (вне метода service, обрабатывающего запрос). Они имеют следующую форму:

```
<%! Код на Java %>
```

Поскольку объявления не осуществляют вывода, обычно они используются совместно с JSP выражениями или скриплетами. В приведенном в качестве примера фрагменте JSP отображается количество запросов к данной странице с момента загрузки сервера (или с момента последнего изменения и перезагрузки сервлета):

```
<%! private int accessCount = 0; %>
Количество обращений к странице с момента загрузки сервера:
<%= ++accessCount %>
```

Также как и в скриплетах, если вам необходимо использовать последовательность символов "%>", используйте для этого последовательность "%\>". XML эквивалентом <%! Код %> является

```
<jsp:declaration>
Код
</jsp:declaration>
```

Директивы JSP

Директивы JSP воздействуют на всю структуру класса сервлета. Обычно они имеют следующую форму:

```
<%@ директива атрибут="значение" %>
```

Вы также можете объединить установку нескольких атрибутов для одной директивы:

```
<%@ директива атрибут1="значение1"
              атрибут2="значение2"
              ...
              атрибутN="значениеN" %>
```

Существуют два основных типа директив: page, которая позволяет вам совершать такие операции, как импорт классов, изменение суперкласса сервлета, и т.п.; и include, которая дает вам возможность вставить файл в класс сервлета при трансляции JSP файла в сервлет. Также следует упомянуть директиву taglib, которая не поддерживается в JSP версии 1.0, но позволяет авторам JSP задавать свои собственные тэги. Предполагается что эта директива станет основной особенностью JSP 1.1.

Пример использования элементов скриптов и директив

На этом простейшем примере показано использование различных конструкций JSP: выражений, скриплетов, объявлений и директив. Также вы можете скачать исходный файл или вызвать этот скриплет из Интернет.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Использование JavaServer Pages</TITLE>
```

```
<META NAME="author" CONTENT="Marty Hall -- hall@apl.jhu.edu">
<META NAME="keywords"
      CONTENT="JSP,JavaServer Pages,servlets">
<META NAME="description"
      CONTENT="Быстрый пример четырех основных тэгов JSP.">
<LINK REL=STYLESHEET
      HREF="My-Style-Sheet.css"
      TYPE="text/css">
</HEAD>

<BODY BGCOLOR="#FDF5E6" TEXT="#000000" LINK="#0000EE"
      VLINK="#551A8B" ALINK="#FF0000">

<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
  <TR><TH CLASS="TITLE">
      Использование JavaServer Pages</TABLE>
</CENTER>
<P>

Некое динамическое содержание созданное с использованием различных механизмов JSP:
<UL>
  <LI><B>Выражение.</B><BR>
      Имя вашего хоста: <%= request.getRemoteHost() %>.
  <LI><B>Scriptlet.</B><BR>
      <% out.println("Дополнительные данные запроса: " +
                     request.getQueryString()); %>
  <LI><B>Объявление (совместно с выражением).</B><BR>
      <%! private int accessCount = 0; %>
      Количество обращений к странице с момента загрузки сервера: <%= ++accessCount
%>
  <LI><B>Директива (совместно с выражением).</B><BR>
      <%@ page import = "java.util.*" %>
      Текущая дата: <%= new Date() %>
</UL>


</BODY>
</HTML>
```