Титульный лист материалов по дисциплине

ДИСЦИПЛИНА **Разработка клиент-серверных приложений**

полное название дисциплины без аббревиатуры

ИНСТИТУТ **Информационных технологий**

КАФЕДРА **Инструментального и прикладного программного**

полное название кафедры **обеспечения**

ГРУППА/Ы **ИКБО-01/02/03/12/13/16-18**

номер групп/ы, для которых предназначены материалы

ВИД УЧЕБНОГО МАТЕРИАЛА **Материал к практическим занятиям**

лекция; материал к практическим занятиям; контрольно-измерительные материалы к практическим занятиям; руководство к КР/КП, практикам

ПРЕПОДАВАТЕЛЬ **Строганкова Наталья Владимировна**

фамилия, имя, отчество

СЕМЕСТР **5**

указать номер семестра обучения

ПРАКТИЧЕСКАЯ РАБОТА № 10

1 Цель работы

Целью данной практической работы является ознакомление с технологией MVC, а также разработка веб-сайта с использованием «шаблонизатора» на основе предыдущей практической работы.

2 Теоретическая часть

Про технологии MVC было подробно рассказано в теоретической части учебно-методического пособия. А сейчас мы подробно рассмотрим тему библиотеки FreeMarker.

FreeMaker – это написанная на Java библиотека, предназначенная для реализации механизма шаблонов. Шаблоны работают с текстом, подставляя в текст значения переменных. Цепочка следующая. Есть исходный текст, он представляет собой HTML с особыми тэгами (выражениями – не HTML, а именно шаблонизатора; язык FreeMaker называется FTL, FreeMaker Template Language). Далее этот текст проходит через некоторый процесс, называемый рендерингом. Выходом является текст. Но в нем воспринимаемые шаблонизатором тэги уже заменены на конкретные значения. На этом этапе обычно остается «чистый» HTML – страница в том виде, в каком она будет передана клиенту.

Структура работы библиотеки FreeMarker хорошо видна на рисунке 1.

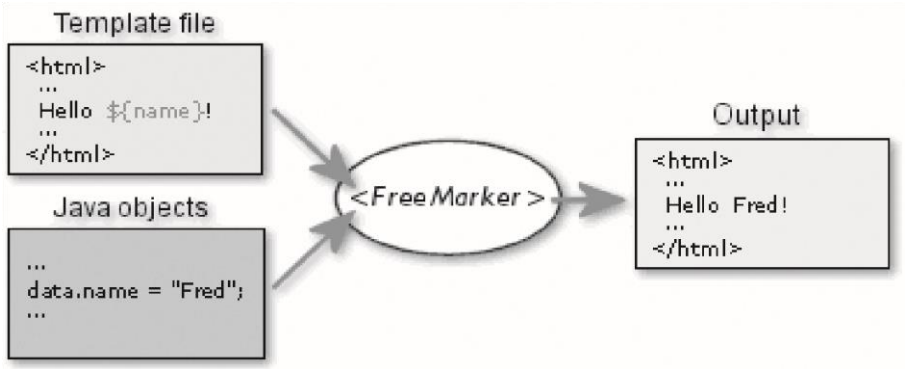

Рисунок 1 – Структура работы библиотеки FreeMarker

Наиболее часто шаблонизатор применяется следующим образом. Создается класс, в котором собирается вся необходимая для отображения информация (изъятая из model MVC). И этот класс используется в качестве *основы* (контекста) для работы шаблонизатора, для получения HTML из заготовки (шаблона).

Рассмотрим следующий пример. Есть шаблон, текстовый файл с именем ourTemplate.html, содержащий следующую строку.

```
<h1>Здравствуйте, ${userName}!</h1>
```

Данный пример позволяет нам не прописывать статично имя пользователя, а задать его через функцию шаблонизатора.

В Java-классе необходимо выполнить следующие действия.

```
1. HashMap renderContext = new HashMap ();
2. renderContext.put ("userName", getCurrentUser ().getName ()); 3.
   Template template = fmConfig.getTemplate ("ourTemplate.html");
3. StringWriter writer = new StringWriter ();
4. template.process (renderContext, writer);
5. String html = writer.toString ();
6. sendToClient (html);
```

Создается словарь (HashMap). В элемент с именем userName помещается ФИО текущего пользователя, полученное вызовом функций getCurrentUser ().getName (). Дальше создается объект-шаблон (template) основанный на ранее сформированной конфигурации (fmConfig) из файла outTemplate.html, и к нему применяем операцию process.

На выходе получаем текстовую строку, которая будет содержать примерно следующее: `<h1>Здравствуйте, Иванов Иван Иванович!</h1>`

Мы *оживили* страницу, заменив шаблон на реальное значение переменной. Такую HTML-строку можно отправлять.

FreeMaker способен заменять шаблоны на реальное значение переменных – это значит сильно упрощать ситуацию. FreeMaker имеет достаточно развитый язык, позволяющий создавать списки, производить условные переходы и т. д. Кроме того, если чего-то не хватило в самом FTL, можно определить и свои собственные директивы (практически это вряд ли понадобится, но такая возможность есть).

Пример списка, сделанного при помощи механизма шаблонов.

Пример шаблона:

```
1. <table>
2. <#list Users as u >
3. <tr>
4. <td>${u.getLastName ()}</td>
5. <td>${u.getFirstName ()}</td>
6. </tr>
7. </#list>
8. </table>
```

Строка в таблице фигурирует один раз. Класс контекста мог бы выглядеть следующим образом:

```
1. HashMap context = new HashMap ();
2. //Собираем всех пользователей, которых мы хотим получить
3. //в таблице на экране – в вектор.
```

```
4. Vector Users = new Vector ();
5. User u = new User ("Иванов", "Иван", "Иванович"); Users.add (u1);
6. User u2 = new User ("Петров", "Петр", "Петрович"); Users.add (u2);
7. User u3 = new User ("Николаев", "Николай", "Николаевич"); Users.add
   (u3);
8. //Теперь в контекст в качестве именованной переменной помещаем сам
   вектор.
9. context.put ("Users", Users);
```

Полученный после рендеринга HTML будет выглядеть примерно следующим образом:

```
1. <table>
2. <tr>
3. <td>Иванов</td>
4. <td>Иван</td>
5. </tr>
6. <tr>
7. <td>Петров</td>
8. <td>Петр</td>
9. </tr>
10. <tr>
11. <td>Николаев</td>
12. <td>Николай</td>
13. </tr>
14. </table>
```

Отметим две вещи:

1) строк в таблице стало столько, сколько элементов в коллекции Users в контексте. Если их будет 100, значит, директива FreeMaker <#list> выполнится 100 раз, и получаем HTML с таблицей, состоящей из 100 строк;

2) в качестве *переменной*, которая заменится шаблонизатором на реальное значение, не обязательно должно выступать именно *значение*. Это может быть и функция, которая вызовется автоматически, и в шаблон будет подставлен результат работы этой функции.

FreeMaker работает с текстом. У него текст на входе и текст на выходе. HTML – это частный случай применения механизма шаблонов. Это может быть и JavaScript, и CSS, и вообще все что угодно. В случае с рендерингом через шаблонизатор JavaScript появляется удобный механизм начального заполнения переменных на клиенте значениями из серверной части.

3 Порядок выполнения работы

Постановка задачи:

1) разобраться с технологией MVC и библиотекой FreeMarker. Используя библиотеку FreeMarker, изменить структуру вашего сайта, результата предыдущей практической работы:

а) отделить одинаковую (статичную) часть от всех страниц (одна из которых обязательно является меню) в отдельный HTML-файл;

б) все созданные ранее страницы изменить в соответствии с требованиями работы с библиотекой FreeeMarker;

2) при организации ответа нужно объединить страницы меню и запрашиваемую страницу с помощью FreeMarker.

4 Защита практических работ

Результаты каждой выполненной практической работы:

1) сформированный проект, выполненный по заданию практической работы;

2) оформленный отчет, содержащий все этапы выполненной работы.

Защита практической работы:

1) продемонстрировать разработанное приложение, выполненное в соответствии с заданием;

2) предоставить отчет по практической работе в печатном виде;

3) давать четкие ответы по выполненной работе (студент должен владеть теоретическими знаниями, свободно комментировать все строчки кода программы и уметь формулировать выводы о проделанной работе).

5 Правила оформления отчета

Отчет по практическим работам должен содержать следующие структурные части:

- титульный лист (1 стр.);
- содержание (1 стр.);
- цели и задачи практической работы (1 стр.);
- формулировки индивидуального задания (если оно предусмотрено);
- перечень библиотек и основных функций, использованных в программе;
- результаты работы программы в распечатанном виде;
- анализ полученных результатов;
- выводы о проделанной работе;
- текст программы с комментариями;
- приложения (при необходимости).

Отчет выполняется в электронном виде с одной стороны листа формата А4 (210х297 мм), шрифтом Times New Roman 14 пт; 1,5 интервалом.