Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра экономической информатики

А. В. Лепеш, В. Н. Комличенко, В. В. Онищук, А. А. Пасовец.

Лабораторный практикум по курсу «Разработка информационных систем в WWW» для студентов экономических специальностей БГУИР. В 2 ч. Ч. 1

Минск 2002

УДК 681.324 (075.8)
ББК 32.97 я 73
Л12

Авторы:

А. В. Лепеш, В. Н. Комличенко, В. В. Онищук, А. А. Пасовец.

Лабораторный практикум по курсу «Разработка информационных
Л12 систем в WWW» для студентов экономических специальностей БГУИР. В 2 ч. Ч. 1 / А. В. Лепеш, В. Н. Комличенко, В. В. Онищук, А. А. Пасовец. — Мн.: БГУИР, 2002. — 56 с.: ил.

ISBN 985-444-077-X (ч. 1)

В лабораторном практикуме представлен курс из четырех лабораторных работ, даны краткие теоретические сведения, необходимые для их выполнения, а также примеры.

УДК 681.324 (075.8)
ББК 32.97 я 73

ISBN 985-444-077-X (ч. 1)
ISBN 985-444-078-8

СОДЕРЖАНИЕ

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Основы DHTML (JavaScript, VBScript)

1.1. Hyper Text Markup Language

Для создания Web-страниц используется Hyper Text Markup Language (HTML). Документ, созданный на языке HTML, содержит обычный текст и дескрипторы, предназначенные для *разметки* текста. С помощью дескрипторов можно описать внешний вид текста (например, указать, что некоторый фрагмент должен отображаться красным цветом или выделяться полужирным шрифтом) или его расположение на странице (например, задать размещение данных в третьей и четвёртой строках таблицы). Однако чаще всего дескрипторы описывают содержимое документа (например, определённый фрагмент текста является заголовком) и предоставляют браузеру решать, как именно следует разместить текст на странице. Основные установки для данного браузера, такие, как начертание, размер и цвет символов, задаются конкретным пользователем. Характеристики отображения, явно не указанные пользователем, определяет сам браузер. Следует помнить, что попытки явно задать внешний вид документов, которые должны отображаться различными браузерами, как правило, оканчиваются неудачей. В первой лабораторной работе студенты познакомятся с каскадными таблицами стилей, которые предоставляют авторам дополнительные возможности по управлению отображением Web-страниц. Однако даже в этом случае создатель Web-страниц не имеет полного контроля над внешним видом своих документов. При разработке Web-страниц могут возникнут дополнительные трудности, связанные с тем, что браузеры – не единственный тип программ, используемый для воспроизведения HTML-документов. Различные приложения могут отображать, выводить на печать, модифицировать, индексировать HTML-документы (например, MS WORD, MS FRONTPAGE) и даже синтезировать речь на основе их содержимого. Однако в курсе лабораторного практикума мы ограничимся лишь отображением документов посредством WWW-браузеров.

Помимо проблем, связанных с отображением в различных браузерах, авторы Web-страниц должны принимать во внимание изменения в спецификации HTML. Так, до января 1997 г. широко применялся стандарт HTML 2.0, разработанный W3C (World Wide Web Consortium). Спецификация HTML 2.0 описывала возможности, типичные для браузеров, созданных в середине 1994 г. Ещё до того, как официально была опубликована HTML 2.0, проводились работы по созданию спецификации нового поколения, которая сначала получила название HTML+, а затем HTML 3.0. Однако основные производители браузеров не стали поддерживать эту спецификацию. В качестве промежуточного варианта при переходе к новому стандарту была предложена спецификация HTML 3.2 (в ней были учтены не все средства HTML 3.0).

В декабре 1997 г. появился стандарт HTML 4.0. Следует понимать, что современные браузеры не поддерживают HTML 4.0 в полном объёме. Наиболее соответствуют данной спецификации Netscape 4.XX – 6.XX; Internet Explorer 4.XX – 6.XX. После принятия в 1997 г. HTML 4.0 в качестве стандарта была предложена спецификация HTML 4.01, в которой были исправлены незначительные ошибки предыдущей спецификации и добавлены некоторые атрибуты.

HTML 4.01 была принята W3C 24 декабря 1999 г. HTML 4.01 важна потому, что на ней базируется новая спецификация XHTML 1.0, предназначенная для поддержки XML в Web-страницах. В этом случае HTML 4.01 используется для интерпретации HTML-дескрипторов. XML позволяет определять форматы данных для электронной коммерции, векторной графики и многих других применений. Более подробно язык XML будет обсуждаться в четвёртой лабораторной работе.

Спецификация XHTML 1.0 была официально принята 26 января 2000 г.

1.2. Структура HTML-документа

Как уже говорилось, документ, созданный на языке HTML, содержит обычный текст и дескрипторы, предназначенные для *разметки* текста. *HTML-элементы* определяются посредством *дескрипторов*, заключённых в угловые скобки. Дескриптор иногда называют «*тэгом*». Некоторые (но не все) языковые элементы представляют собой пары дескрипторов **<ИМЯ> и </ИМЯ>** и являются *контейнерами*. Например, **<ИМЯ>** представляет собой *начальный* или *открывающий*, дескриптор элемента **ИМЯ**, а **</ИМЯ>** является *закрывающим* или *конечным*.

Если браузер не может распознать HTML-элементы или атрибуты, то он их игнорирует. Текст документа всегда обрабатывается, даже если он содержится между недопустимыми начальными и конечными дескрипторами.

Дополнительная информация об HTML-элементах задаётся посредством *атрибутов*, которые записываются в форме *атрибут-значение*. Например, в дескрипторе **<IMG SRC=”images/sample.gif”>** последовательность **images/sample.gif** является значением атрибута **SRC** элемента **IMG. E**сли в составе значения атрибута присутствуют символы, отличные от букв и цифр, это значение должно быть заключено в кавычки. Поэтому имеет смысл представлять в кавычках все значения атрибутов, за исключением целых чисел. Все имена HTML- элементов и атрибутов не зависят от регистра символов, однако это не относится к значениям атрибутов. Лишние пробелы, присутствующие в тексте документа, а также рядом с именами элементов атрибутов, игнорируются, но в составе значений атрибутов пробелы учитываются. Для сравнения, в спецификации XHTML 1.0 имена элементов должны быть представлены символами нижнего регистра, а значения атрибутов, содержащие цифры, заключаются в кавычки. Это соответствует правилам XML (см. далее в четвёртой лабораторной работе).

Комментарии помещаются между наборами символов “<!--” и “-->”, могут располагаться на нескольких строках, но не должны включать символы “-“.

HTML-документ начинается с декларации **DOCTYPE**, в которой задаётся версия HTML. Информация о версии используется браузером или программой проверки. После **DOCTYPE** следует элемент HTML, содержащий элементы **HEAD** и **BODY**. Элемент **HEAD** содержит элемент **TITLE**. Элемент **BODY** непосредственно следует за элементом **HEAD**.

Со следующего шаблона удобно начинать создание большинства WEB-страниц, соответствующих спецификации HTML 4.0:

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01 Transitional//EN”>
<HTML>
<HEAD>
<TITLE> Ваш Заголовок </TITLE>
</HEAD>

<BODY>
<H1>Main Heading</H1>

<!— Остальная часть Вашей WEB-страницы -->
</BODY>
</HTML>
```

В зависимости от содержимого HTML-документа могут использоваться три варианта декларации DOCTYPE:

- строгая **<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01//EN”>**;
- промежуточная **<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01 Transitional//EN”>**;
- для работы с фреймами **<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01 Frameset//EN”>**.

1.3. Назначение сценариев

Как языки сценариев, используемые в Web, соотносятся с полноценными языками программирования типа Java? Как правило, языки сценариев используются небольшими частями для выполнения определенных задач. Сценарии имеют узкую сферу применения, например, для таких задач, как расчет размеров ссуды. По данным простым задачам можно судить об основных функциях языка сценариев. Чаще всего их используют для:

- создания HTML-кода в процессе загрузки WEB-страницы;
- включения динамических элементов в состав документа;
- обработки событий, связанных с действиями пользователя;
- оживления Web-страниц;
- связывания вместе объектов страницы;
- проверки данных HTML-формы;
- обработки cookie;
- работы с фреймами;

- обеспечения взаимодействия Java и JavaScript (для JavaScript).

1.4. JavaScript

Несмотря на сходство имён, JavaScript существенно отличается от Java. JavaScript – это язык сценариев; фрагменты программ на JavaScript встраиваются на Web-страницы и выполняются при загрузке документов. Java – это язык программирования общего назначения, который можно использовать для написания приложений, программ, выполняющихся на стороне сервера, и апплетов, работающих в среде браузера. JavaScript-сценарию доступны практически любые сведения об HTML-документе, он может выполнять различные действия с элементами, которые содержатся в этом документе. Апплет, включённый в документ, изолирован от остальных элементов Web-страницы. JavaScript-сценарий не может пользоваться графической библиотекой, создавать запросы или поддерживать сетевые соединения. Программисту, пишущему Java-программы, доступны мощные графические средства (AWT, Swing, Java 2D), классы для поддержки потоков и сетевого обмена, включая сокеты (гнёзда), RMI и JDBC.

На момент написания этих строк существуют шесть версий JavaScript (от 1.0 до 1.5) и седьмая версия JavaScript 2.0, предложенная Mozilla Organization. Следует понимать, что основные языковые средства JavaScript с момента создания версии 1.2 не претерпели существенных изменений и поддерживаются практически всеми популярными браузерами.

JavaScript как язык сценариев достаточно прост, он не компилируется (а интерпретируется браузером) и даже полезен в малых порциях. Это и отличает его от Java и других языков, которые используются в Интернете. Программы, написанные на них, как правило, должны компилироваться, а освоить их непрограммисту очень трудно. Синтаксис JavaScript чем-то напоминает язык C или Java, кроме того, в нем имеются операции для работы с выражениями, которые есть в языке Perl, и черты языка объектно-ориентированного программирования.

Пример того, как программа на JavaScript может использоваться для проверки содержимого формы перед отправкой их в серверную программу, представлен ниже.

```
<HTML>
<HEAD>
<TITLE>Пример JavaScript</TITLE>
<SCRIPT TYPE="JavaScript">
<!—-
function validate ()
{
if (regform.name.value == "")
{alert ("Вы должны указать свое имя");
return;
}
else
regform. submit ()
```

```
//-->
</SCRIPT>
</HEAD>
<BODY>
<H1 ALIGN="CENTER">Регистрация</H1>
<HR>
<FORM ACTION="mailto:CafedraEI@bsuir.unibel.by" METHOD="POST"
NAME="regform">
Имя: <INPUT NAME-"name" TYPE="TEXT" SIZE="20" MAXLENGTH="40"> <BR><BR>
<INPUT TYPE="BUTTON" VALUE="Зарегистрировать" onClick-="validate ()">
</FORM>
</BODY>
</HTML>
```

1.5. VBScript

Язык Visual Basic Scripting Edition, сокращенно VBScript, является подмножеством популярного языка Visual Basic. Данный язык может использоваться для выполнения тех же самых операций, что и JavaScript, он так же как и JavaScript, способен работать с различными объектами, из которых состоит Web-страница (этот набор объектов называется объектной моделью документа). Как кросс-платформенный язык сценариев VBScript использовать нельзя. Но в более управляемой среде, например в интрасетях, язык VBScript в сочетании с элементами ActiveX может предоставить все, что необходимо разработчику, ориентирующемуся на использование решений компании Microsoft. В сочетании с элементами ActiveX язык VBScript может сделать даже больше, чем JavaScript. Ниже представлен пример программы на VBScript (данный пример выполняет ту же операцию, что и пример на JavaScript):

```
<HTML>
<HEAD>
<TITLE>Пример VBScript</TITLE>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub Reg_OnClick
Dim TheForm
Set TheForm = Document.regform
If (TheForm.name.Value) = "" Then
MsgBox "Вы должны указать свое имя"
Else
TheForm.submit
End If
End Sub
—->
</SCRIPT> </HEAD>
<body>
<H1 ALIGN="CENTEK">Регистрация</H1>
<HR>
<FORM ACTIOH="mailto: CafedraEI@bsuir.unibel.by " METHOD-POST
NAME="regform">
Имя: <IMPUT NAME="name" TYPE="TEXT" SIZE="20" MAXLEHGTH»"40">
```

```
<br><br>
<INPUT  TYPE="BUTTON" VALUE=Register" NAME="Reg">
</FORM>
</body>
</HTML>
```

1.6. Вставка сценариев в HTML-документ

Как вы уже поняли из предыдущих примеров, вставить сценарий, написанный на любом языке, в страницы можно прежде всего с помощью элемента <SCRIPT>. Элемент <SCRIPT> служит для выделения кода сценария, все, что находится внутри него, рассматривается браузером как код, а не как HTML-разметка. На эту фразу разработчикам необходимо обратить особое внимание. В коде сценария символы перевода строки и регистр букв играют важную роль, в то время как в HTML — нет.

Хотя содержимое элемента <SCRIPT> может быть очень сложным для понимания, синтаксис самого элемента достаточно прост. У элемента <SCRIPT> есть только три важных атрибута, которые представлены в табл. 1.1.

Сам код выполняемого сценария должен размещаться между тегами **<SCRIPT> и </SCRIPT>,** например:

<SCRIPT LANGUAGE="JavaScript">Здесь помещается код </SCRIPT>

Таблица 1.1. Атрибуты элемента <SCRIPT>

Название атрибута	**Значения**	**Описание**
LANGUAGE	JavaScript, JScript, VBS, VBScript	Значение данного атрибута указывает, какой язык используется в сценарии. Имеется два возможных варианта: JavaScript или VBScript
SRC	URL	Данный атрибут указывает на URL файла, в котором содержится код внешнего загружаемого сценария
TYPE	Application/x-Javascript*, text/JavaScript/ text/VBscript	Данный атрибут указывает на MIME-тип запускаемого сценария
* Значение атрибута TYPE application/x-Javascript использовать не рекомендуется, хотя он часто встречается в старых браузерах, поддерживающих JavaScript		

Элемент <SCRIPT> может встречаться несколько раз в документе и может располагаться как в элементе <HEAD>, так и в элементе <BODY>. Так как HTML-страницы читаются браузером последовательно, лучше всего располагать код сценариев в заголовочном разделе документа. Некоторые из сценариев могут являться так называемыми отложенными сценариями, так как браузер их считывает и исполняет не сразу. Данный отложенный код может быть вызван позднее другим сценарием или сценариями, реагирующими на действия пользователя, расположенными внутри тела HTML-документа. Сценарий, помещаемый в элементе <HEAD> документа, напоминает объявление функции или процедуры.

Элемент <SCRIPT> может также располагаться и в теле документа. В данном случае он, как правило, выполняется сразу же, как только браузер его прочитает.

Помимо использования элемента <SCRIPT>, код сценариев можно вставлять прямо в HTML-теги. Обычно это делают для того, чтобы обработать действия пользователя. У HTML-элемента, как правило, есть специальный атрибут, на основе которого создается обработчик событий.

Сочетая отложенные сценарии с событиями, можно создавать динамические документы. Следующий пример демонстрирует кнопку, которая при нажатии открывает небольшое диалоговое окно, созданное с помощью JavaScript.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!—
function AlertTest( ) {
alert("Внимание запущен сценарий на JavaScript! ");
}
//-->
</SCRIPT>
</HEAD>
<BODY><DIV ALIGN="CENTER">
<FORM> <INPUT TYPE="BUTTON" NAME="TestButton"
VALUE="Не нажимай меня!" onClick="AlertTest()">
</FORM>
</DIV></BODY>
</HTML>
```

Существует еще два способа добавления сценариев в HTML- документ. В первом способе используется псевдопротокол URL. Браузеры Netscape ввели эту новую форму URL вместе с ключевым словом JavaScript:, которое теперь можно использовать в ссылках, например:

<A HREF='Javascript:alert("Внимание запущен сценарий на JavaScript!")'>Запусти меня!</A>

И, наконец, очень необычный способ добавления кода на JavaScript в Web-страницу заключается в использовании символьных примитивов. Код на JavaScript можно вставить в специальный примитив, который выглядит так: &{код на JavaScript};. Код на JavaScript должен обязательно располагаться в фигурных скобках, внутри же может размещаться вызов функции и даже несколько команд. Данный примитив может выступать только в роли значения атрибута. Можно, например, в заголовке документа определить массу характеристик для цвета и стиля шрифта, а затем вызвать их по имени позднее в коде страницы, например:

```
<HTML>
<HEAD>
<TITLE> Код в  примитиве</TITLE>
<SCRIPT LANGUAGE="JavaScript">
textColor='green';
</SCRIPT>
```

</HEAD>
<BODY>
<FONT COLOR=&{textColor;};>Этот текст должен быть зеленым</FONT>
</BODY>
</HTML>

1.7. Внешние сценарии

Код сценария можно поместить и в отдельный файл, а затем сослаться на него с помощью атрибута SRC, которому присваивается URL этого файла, например:

<SCRIPT SRC="http://www.bsuir.unibel.by/scripts/myscript.js"> </SCRIPT>

Данный фрагмент загружает сценарий из файла myscript.js, URL которого присвоен атрибуту SRC. Если в странице подключается внешний сценарий с помощью атрибута SRC, нет необходимости использовать атрибут LANGUAGE. Хоть браузер и предположит, что сценарий написан на языке JavaScript, Web-сервер, судя по расширению файла, сам сообщит о том, какой язык будет использоваться. Web-сервер должен связать расширение с соответствующим MIME-типом, в данном случае application/x-JavaScript, чтобы браузер, получив файл, понял, что с ним надо делать. Для старых серверов может понадобиться особая настройка MIME-типа, чтобы он мог работать с внешними файлами сценариев.

1.8. Обработчики событий и HTML

Код сценариев в HTML-документы можно подключить к специальным атрибутам, называемым событиями. Что такое событие? Событие происходит в результате какого-либо действия, пользователя или какого-либо внешнего события, например, загрузки страницы. Примером события может служить нажатие пользователем кнопки, клавиши, перемещение окна или даже просто перемещение мыши по экрану. Язык HTML позволяет связать сценарий с определенным событием через соответствующий ему атрибут. Название атрибута составляется из названия события, перед которым ставится приставка on, например onclick. В представленном ниже фрагменте кода показано, как атрибут-событие связывается со сценарием, реагирующим на нажатие кнопки:

<form><BUTTON onclick='alert("Эй, это JavaScript!")' VALUE="Нажми меня"></form>

Согласно спецификации HTML 4.0, обработчики событий можно добавить к достаточно большому количеству HTML-элементов. Как можно судить по табл. 1.2, в спецификации HTML 4.0 определен широкий набор событий практически для каждого элемента.

Базовая модель событий, согласно спецификации HTML 4.0, включает в себя следующие события: onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover и onmouseup. Эти базовые события существуют практически у всех HTML-элементов, которые выводятся на экран. Хоть в HTML 4.0 и определено множество событий, в

браузерах Netscape и Internet Explorer их еще больше, т.е существуе расширенная модель событий, которая здесь не приводится.

Таблица 1.2. Набор атрибутов-событий для элементов согласно HTML 4.0

Атрибут-событие	**Описание события**	**Элементы, имеющие данный атрибут**
Onblur	Элемент перестает быть активным, то есть когда пользователь переходит в другое поле формы либо щелчком мыши, либо с помощью клавиши [Tab]	<A>. <AREA>, <BUTTON>, <INPUT>, <LABEL>, <SELECT>, <TEXTAREA>
Onchange	Элемент формы перестает быть активный и при этом его значение было изменено	<INPUT>, <SELECT>, <TEXTAREA>
Onclick	По элементу щелкают мышью	Большинство элементов
Ondblchick	По элементу дважды щелкнули мышью, т. е. сделано два быстрых щелчка	Большинство элементов
Onfocus	Элемент формы становится активным, а именно — он был выбран пользователем, чтобы произвести определенные действия или ввести данные	<A>, <AREA>, <BUTTON>, <INPUT>, <LABEL>, <SELECT>, <TEXTAREA>
Onkeydown	Нажимается клавиша	Большинство элементов
Onkeypress	Клавиша нажимается, а затем отпускается	Большинство элементов
Onkeyup	Нажатая клавиша отпускается	Большинство элементов
Onload	Заканчивается загрузка документа в окно или в набор кадров	<BODY>. <FRAMESET>
Onmousedown	Нажимается кнопка мыши	Большинство элементов
Onmousemove	Перемещается курсор мыши	Большинство элементов
Onmouseout	Курсор мыши уходит от элемента	Большинство элементов
Onmouseover	Курсор мыши располагается над элементом	Большинство элементов
Onmouseup	Отпускается нажатая кнопка мыши	Большинство элементов
Onreset	Все данные в форме сбрасываются нажатием кнопки Reset (Сброс)	<FORM>
Onselect	Пользователь выделяет текст	<INPUT>,
Onsubmit	Форма отправляется на обработку нажатием кнопки Submit (Отправить)	<FORM>
Onunload	Из браузера, из окна или кадра выгружается текущий документ	<BODY>, <FRAMESET>

1.9. HTML и сценарии

Объектная модель документа определяет модель для всех объектов и HTML-элементов, составляющих Web-страницу, тем не менее, для того чтобы сценариям было проще считывать и управлять объектами, необходимо придерживаться определенных правил при присвоении имен элементам. Основной способ, которым согласно HTML 4.0 каждому элементу можно присвоить уникальное имя, является атрибут ID. Этот атрибут присутствует практически в каждом элементе, это подтверждается табл. 1.3.

Назначение атрибута ID состоит в том, чтобы присвоить элементу уникальный идентификатор. Например, чтобы присвоить определенному тексту название «сверхважный», можно набрать следующий код:

<B ID="SuperImportant">Это очень важный текст</B>

Таблица 1.3. Поддержка атрибута ID в элементах HTML 4.0

Элементы, в которых атрибут ID есть	**Элементы, в которых атрибута ID нет**
<A>, <ACRONYMN>, <ADDRESS>, <APPLET>, <AREA>, <B>, <BASEFONT>, <BDO>, <BIG>, <BLOCKQUOTE>, <BODY>, , <BUTTON>, <CAPTION>, <CENTER>, <CITE>, «CODE>, <COL>, <COLGORUP>. <DD>; <DEL>, <DFN>, <DIR>, <DIV>, <DL>, <DT>, <EM>, <FIELDSET>, <FONT>, <FORM>, <FRAME>, <FRAMESET> <H1><H2>, <H3>, <H4>, <H5>. <H6>. <HR>, <(>, <IFRAME>, <IMG>, <INPUT>, <INS>, <ISINDEX>, <KBD>, <LABEL>, <LEGEND>, <Lf>, <SCR1PT>, <STYLE>, <LINK>, <MAP>, <MENU>, <NOFRAMES>, <NOSCRIPT>, <OBJECT>, <OL>, <OPTGROUP>, <OPTION>. <P>, <PARAM>, <РЯЕ>, <Q>, <S>, <SAMP>, <SELECT>, <SMALL>, <SPAN>, <STRIKE>. <STRONG>, <D>, <SUP>, <TABLE>, <TBODY>, <TD>, <TEXTAREA>, <TFOOT>, <TH>, <THEAD>, <TR>, <TT>,.<D>, <UL>, <VAR>.	<BASE>, <HEAD>, <HTML>, <META>, <SCR1PT>, <STYLE>, <TITLE>

Стиль выбранного имени играет большую роль. Авторам Web-страниц рекомендуется придерживаться постоянных правил при присвоении имен и избегать использования запутанных названий, в которые включаются названия самих HTML-элементов. Например, имя «button» не совсем удачно, кроме того, оно может помешать обращению к элементу из сценария. Чтобы обеспечить уникальность имен, воспользуйтесь символом подчеркивания, например: _myButton.

1.10. Взаимодействие сценариев со списками стилей

И в Netscape, и в Internet Explorer возможен доступ из сценариев к спискам стилей. Netscape поначалу делала особый акцент на своих списках стилей, доступ к которым осуществлялся с помощью языка JavaScript (JASS — JavaScript-accessible stylesheets). В JASS можно было с помощью сценария на JavaScript задать стиль страницы. Однако после прочтения страницы браузером стиль уже изменить было нельзя. Microsoft продемонстрировала, как с помощью сценариев можно менять стили. Например, следующий фрагмент кода показывает, как к событию можно привязать сценарий, который меняет стиль оформления текста: когда мышь располагается над ним, текст становится красным.

```
<SPAN onmouseover='this.style.color="#FFOOOO"'
onmouseout='this.style.color="#000000"'>Цветная кнопка</SPAN>
```

Используя специальное ключевое слово «this», можно сослаться на текущий элемент, но можно воспользоваться и атрибутом ID, например:

```
<SPAN ID="testtext"
onmouseover='testtext.style.color="#FFOOOO";
testtext.style.fontSize="larger"'
onmouseout='testtext.style.color="#000000";
testtext.style.fontSize="smaller"'>
Цветная кнопка
</SPAN>
```

2. CSS (Каскадные таблицы стилей)

2.1. Назначение CSS

Дизайн Web-узлов - это точное размещение компонентов HTML-страниц относительно друг друга в рабочей области окна браузера.

Неточность данного определения Web-дизайна очевидна. В нем не учтены ни цвет, ни форма, ни другие свойства компонентов HTML-страниц. Главное в этом определении - показать ограниченность возможностей HTML-разметки. Позиционирование компонентов на странице является одним из самых слабых мест в HTML.

Компонентами страницы являются: блоки текста, графика и встроенные в страницу приложения. Размер и границы каждого из этих компонентов в рамках HTML-разметки задаются с разной степенью точности. Размер графики и приложений можно задать с точностью до пикселя. Размеры текстовых блоков в HTML задать нельзя. Они вычисляются браузером на основе относительного размера шрифта умолчания.

Автор страницы не может заранее определить настройки браузера пользователя, что существенно ограничивает число вариантов представления информации на странице.

Другой способ управления настройками браузера - программирование на JavaScript. Бурное развитие этого языка заставляет говорить о возможности полного контроля над процессом отображения HTML-страниц. Недостаток JavaScript - отказ от декларативного характера разметки и относительно большой объем кода для переопределения свойств элементов разметки.

Спецификация **CSS**(Cascading Style Sheets) позволяет остаться в рамках декларативного характера разметки страницы и дает полный контроль над формой представления элеметов HTML-разметки.

Каскадные таблицы стилей призваны разрешить противоречие между точностью определения размеров картинок и приложений с одной стороны, и точностью определения размеров блоков текста и его начертания с другой.

Кроме размера компонентов, таблицы стилей позволяют определить цвет и начертание текстового фрагмента, изменять эти параметры внутри текстового блока, выполнять выравнивание текстового блока относительно других блоков и компонентов страницы.

Наличие всех этих возможностей позволяет говорить о CSS как о средстве разделения логической структуры документа и формы его представления. Логическая структура документа определяется элементами HTML-разметки, в то время как форма представления каждого из этих элементов задается CSS-описателем элемента.

CSS позволяет полностью переопределить форму представления элемента разметки по умолчанию. Например, ... определяет отображение текста курсивом:

Отобразим текст курсивом

Отобразим текст курсивом

А теперь переопределим стиль отображения для элемента разметки i:

****
Отобразим текст курсивом
****

Отобразим текст курсивом

Этот пример показывает, что привычный стиль отображения элементов может быть полностью изменен при помощи CSS. В данной технологии HTML-разметка носит чисто декларативный характер.

Практическая значимость CSS для Web-инжиниринга (совокупности технологий разработки и сопровождения Web-узлов) заключена в том, что процесс создания узла можно формализовать и представить в виде следующей последовательности действий:

- Сначала нужно определиться с номенклатурой страниц, т.е. все страницы проектируемого Web-узла разбить на типы, например,: домашняя страница, навигационные страницы, информационные страницы, коммуникационные страницы и т.п.. У каждого узла этот перечень может быть своим.
- Для каждого из типов страниц разрабатывается определенная логическая структура (стандартный набор компонентов страницы).
- После этого разрабатывается навигационная карта узла и форма ее реализации на страницах.
- Для каждого стандартного компонента страницы разрабатывается стиль его отображения (CSS-описатель).
- Теперь остается только рисовать картинки, создавать анимацию, писать программы, вручную вводить текст и графику или генерировать содержание страниц автоматически во время обращения к ним.

Объяснив таким образом роль и назначение CSS среди многообразия Web-технологий, мы перейдем непосредственно к обсуждению практики применения каскадных таблиц стилей.

2.2. Способы применения CSS

Под способами применением CSS мы в данном разделе понимаем форму декларирования стиля на HTML-странице и форму связывания описания стиля отображения элемента разметки с самим этим элементом. Речь идет о том, где и в какой форме автор страницы (или дизайнер) описывает стиль, и как и в какой форме на него ссылается.

Итак, различают четыре способа применения стилей:

- переопределение стиля в элементе разметки;
- размещение описания стиля в заголовке документа в элементе STYLE;
- размещение ссылки на внешнее описание через элемент LINK;
- импорт описания стиля в документ.

2.3. Переопределение стиля

Под переопределением стиля в элементе разметки мы понимаем применение атрибута STYLE у данного элемента разметки:

```
<h1 style="font-weight:normal;
font-style:italic;
font-size:10pt;">
Заголовок первого уровня
</h1>
```

Атрибут style можно применить внутри любого элемента разметки. Например, мы можем через style определить ширину и выравнивание элемента hr(горизонтальное отчеркивание):

<hr style="width:100px;">

Очевидно, что не всякие параметры стиля можно установить для конкретного элемента разметки.

Здесь же следует отметить следующее: стили разработаны в первую очередь для управления отображением текста. Не следует увлекаться стилями при управлении отображением нетекстовых элементов HTML-разметки.

2.4. Элемент STYLE

Применение элемента STYLE - это основной способ внедрения каскадных таблиц стилей в ткань HTML-документа. Кроме управления отображением элементов разметки элемент STYLE позволяет описывать стилевые свойства элементов, которые можно изменять при программировании на JavaScript.

Элемент STYLE позволяет определить стиль отображения для:

- стандартных элементов HTML-разметки;
- произвольных классов (селектор class);
- HTML-объектов (селектор id).

К сожалению, работа с селекторами в браузерах различных производителей может преподнести различного рода сюрпризы. Особенно это касается работы с селектором ID. Будем считать правильной интерпретацию Microsoft как держателя патента на спецификацию CSS.

Стандартные элементы разметки описываются в элементе STYLE следующим способом:

```
<head>
<style> p {color:darkred;text-align:justify;font-size:8pt;} </style>
</head>
<body>
...7444
<p>
```

Этот параграф мы используем в качестве примера применения описания стиля для стандартного элемента HTML-разметки.

</p>

...

</body>

Теперь все параграфы документа будут отображаться стилем из элемента STYLE, если только стиль не будет переопределен каким-либо способом. В STYLE можно определить стиль любого элемента разметки.

2.5. Ссылка на внешнее описание

Ссылка на описание стиля, расположенное за пределами документа, осуществляется при помощи элемента LINK, который размещают в элементе HEAD. Внешнее описание может представлять из себя файл, содержание которого - описание стилей. Описание стилей в этом файле будет по синтаксису в точности совпадать с содержанием элемента STYLE.

Ниже приведен пример ссылки на внешнее описание стилей:

```
<link type="text/css" rel="stylesheet" href="http://kuku.ru/my_css.css">
```

Важными здесь являются значения атрибутов rel и type. Rel обязан иметь значение "stylesheet". Type может принимать значения: "text/css" или "text/Javascript". Второй тип описания стилей введен Netscape. Его мы в данном учебном курсе не обсуждаем.

Атрибут href задает универсальный локатор ресурса (URL) для внешнего файла описания стилей. Это может быть ссылка на файл с любым именем, а не только на файл с расширением *.css.

2.6. Импорт описания стилей

Импорт описателей стилей - это в некотором смысле конкурент описанному выше указанию на внешний описатель стиля.

Импортировать стиль можно либо внутрь элемента STYLE, либо внутрь внешнего файла, который представляет собой описатель стиля. Оператор импорта стиля должен предшествовать всем прочим описателям стилей:

```
<style>
@import:url(http://kuku.ru/style.css)
a {color:cyan;text-decoration:underline;}
</style>
```

Импортируемый стиль можно переопределить либо через описатель элемента в STYLE, либо через атрибут элемента style.

3. Основы программирования в Java

Первая версия языка Java была выпущена компанией Sun Microsystem Inc. в 1991 г. Первоначально язык назывался “Oak” (Дуб). Java был задуман как независимый от платформы язык, с целью внедрения в электронные устройства различных потребителей. Позднее открылась и другая его более значительная особенность – пригодность для использования в WWW. Публичное признание язык получил в 1995 году под именем “Java” (Ява).

Простой, объектно-ориентированный, интерпретируемый язык Java был создан с использованием концепций, заимствованных из других языков, таких,

как C, C++, smalltalk, objective и других. С использованием Java можно создавать два типа приложений:

Апплеты – мини-приложения, выполняемые в среде Java совместимого браузера, например Netcscape, Microsoft Exploer, Hot Java и др.

Приложения – программы, выполняющиеся на локальном компьютере с помощью его операционной системы, подобно приложениям на других языках программирования.

Java – надежный, универсальный, защищенный, независимый от платформы объектно-ориентированный язык. Он позаимствовал синтаксис языка C++ и его лучшие качества. Кроме того, стандартные библиотеки Java значительно расширены и включают мощные функции поддержки сетевых соединений. Это позволяет легко создавать средствами Java приложения для Internet.

Исходный текст на Java компилируется в переносимые байт-коды, для выполнения которых необходим **интерпретатор** (**виртуальная машина JAVA**). Для апплетов это браузер. Приложения командной строки запускаются из строки приглашения MS-DOS или командного процессора UNIX, подобно обычным командам этих операционных систем. Для выполнения приложений (приложения командной строки JAVA имеют расширение **.class)** необходима интерпретирующая программа (виртуальная машина с именем Java.exe, имеющаяся в каталоге BiN, например, пакета SDK). Для запуска приложения, например приложения MyApp.class, необходимо в режиме командной строки ввести: Java MyApp. Расширение .class в команде запуска не указывается.

3.1. Установка Java и переменных окружения

Программа установки Java (пакет Java SDK, известный также под названием JDK) представляет собой самораспаковывающийся архив, при разархивировании которого выдаются инструкции по установке. Не следует разархивировать файлы classes.zip, они предназначены для использования в среде выполнения Java.

Java SDK включает обычно 6 подкаталогов:

bin – содержит выполняемые модули и утилиты JDK;

demo – включает множество апплетов, а также примеры текстов программ на Java;

docs – содержит документацию по Java;

include – включает заголовочные файлы C и C++, используемые для построения среды Java;

lib – библиотеки и архивы, используемые в Java;

src – исходные коды библиотек, созданных компанией Sun.

При работе Java использует переменную окружения **CLASSPATH**, которая должна содержать путь к каталогу с библиотеками lib/classes.zip.

Посмотреть содержимое этой переменной можно вводом в командной строке приглашения MS DOS команды SET без параметров.

На компьютерах, работающих под управлением Windows 95/98, переменную CLASSPATH можно ввести непосредственно редактированием файла autoexec.bat.

На компьютерах, работающих под системой Widows NT, установка этой переменной выполняется через панель управления.

Для этого необходимо:

- В меню start выбрать команду Seting->Control Panel (Настройка-> Панель управления).
- В появившемся диалоговом окне открыть (двойным щелчком) пиктограмму System (Система).
- В диалоговом окне System Properties (Свойства: система) выбрать закладку Environment (Окружение).
- В верхнем списке System Variables выбрать переменную CLASSPATH. После щелчка на ней ее текущее значение появится в текстовом поле в нижней части окна.
- Добавить в переменную CLASSPATH путь к каталогу, содержащему библиотеки.

Рабочий каталог для своих собственных программ можно разместить в любом месте, поэтому, чтобы не приходилось каждый раз при запуске указывать путь к утилитам JDK, необходимо указать в переменной **PATH** путь доступа к утилитам (к файлу bin).

3.2. Основные утилиты Java

Наиболее важными в Java являются следующие утилиты:

javac – компилятор Java, компилирует разработанный текст программы в файл с именем компилируемого текста и расширением .class, который можно запустить при помощи утилиты Java (просмотреть работу апплета можно с помощью утилиты appletviewer);

java – утилита-интерпретатор (вернее программа запуска приложений Java);

appletviewer –программа просмотра апплетов;

jdb – утилита тестирования программ, написанных на Java. Для проверки корректности установки данного отладчика необходимо ввести в коммандной строке команду jdb. В ответ должно появиться сообщение Initializing jdb, завершающееся правой угловой скобкой (>). Чтобы просмотреть перечень доступных команд, введите help. Для запуска отладчика вызовите его с указанием имени файла с интересующим вас классом (файла с расширением .class);

javadoc – утилита для создания документации. Собирает в виде HTML-страницы комментарии из исходного текста программы Java, ограниченные комментирующими скобками /** … */.

3.3. Разработка апплетов

Апплеты встраиваются в HTML - страницу и пересылаются на машину клиента, где выполняются при поддержке браузера. Для внедрения апплетов Java в среду браузера необходимо создать HTML-ter, вызывающий апплет. Обычно HTML - документ, вызывающий апплет, и сам апплет располагаются на сервере.

Загрузка апплета выполняется следующим образом:

- загружается HTML – файл;
- обнаруживается ter <APPLET>;
- загружается файл класса, указанного в APPLET, с сервера;
- распознаются и загружаются классы, на которые ссылается класс APPLET;
- класс APPLET вызывает методы init ( ) start ( );
- при нормальном их завершении интерфейсная часть апплета отображается в окне браузера (или вне его, если апплет использует собственный кадр);

Синтаксис <APPLET> Ter должен обязательно содержать атрибуты **CODE, WIDTH, HELGHT**. Остальные атрибуты тега не являются обязательными.

Основные пакеты:

java Java.applet – классы, необходимые для создания Java-апплетов.

java-awt – для создания аппаратно-независимого графического интерфейса приложений.

java.io – классы для ввода/вывода (кдассы потоков данных).

java.lang – основные классы Java. Пакет импортируется неявно.

java.net – классы для реализации сетевых приложений (чтение, запись и т.д. в сети)

java.util – вспомогательные утилиты и структуры данных.

Рассмотрим пример апплета, в котором прокомментирован жизненный цикл его существования.

Пример:

```
import Java.applet.Applet;  //импорт конкретного класса Applet
import Java.awt.*;    //импорт пакета AWT
public class LifeCycle extends Applet   {//наследование от класса //Applet
     public LifeCycle() {//ограничение public необходимо,
                    //чтобы вызвать апплет извне.
   System.out.println("Запущен конструктор.\n");   //вывод
                    //сообщения на системную консоль (в окно MS DOS)
 }
 public void init() {
   System.out.println("Работает метод init .\n");
 }
 public void start() {
   System.out.println("Апплет запущен .\n");
```

```
  }
  public void paint(Graphics theGraphics) {
                theGraphics.drawString("Hello, Word",0,50);  //вывод в
                        //графическое окно апплета, графика поставляется
                 //классом Graphics из пакета awt.
          System.out.println("Апплет вывел строку .\n");
  }
        public void stop() {
    System.out.println("Апплет остановлен  .\n");
  }
        public void destroy() {
    System.out.println("Апплет уничтожен  .\n");
  }
}
```

Рассмотренный апплет сам сообщает о последовательности состояний своего жизненного цикла.

Интервал между start и stop предназначается для размещения методов реализации действий апплета.

Представленный текст необходимо набрать в любом ASCII-редакторе, сохранить в файле с именем класса и расширением Java. При этом нужно следить за тем, чтобы имя файла и имя создаваемого класса апплета в точности совпадали (**Java чувствительный к строчным и прописным символам и LifeCycle и Lifecycle – разные имена**). Набранный исходный текст следует оттранслировать в байт-коды при помощи утилиты Javac, введя в коммандной строке следующее:

javac LifeCycle.Java

Выходной файл трансляции будет создан с именем LifeCycle.class.

Для выполнения апплета нужно подготовить простейшую страничку HTML, с именем LifeCycle.html, в которой разместить вызов апплета, например, следующим образом:

```
<HTML>
<TITLE> Жизненный цикл апплета </TITLE>
<APPLET CODE="LifeCycle.class" WIDTH=200 HEIGHT=200></APPLET>
</HTML>
```

Просмотреть выполнение апплета можно либо при помощи браузера, либо при помощи программы просмотра, введя в командной строке следующее:

appletviewer LifeCycle.html

Кроме обязательных атрибутов для тега <APPLET> могут использоваться:

CODEBASE=URL - допустимая URL-ссылка на каталог, в котором располагается файл класса этого апплета, если HTML-файл и файл каталога располагаются в разных классах.

Alt=строка – определение альтернативного текста для случая, когда Java – совместный браузер не может выполнить апплет.

NAME – частное имя апплета, по которому другие апплеты, расположенные на той же HTML – странице, могут обращаться к нему.

ALIGN- выравнивание апплета: допустимые значения: left, right, texttop, middle и др.

VSPACE- пустые места ( ) – сверху и снизу от апплета.

HSPASE- пустые места ( ) – справа и слева от апплета.

PARAM – параметры, передаваемые апплету.

Любой HTML текст, располагающийся вне тела <APPLET>, за исключением атрибута <PARAM>, игнорируется Java–совместимыми браузерами, а другими распознается. Поэтому вне тела можно разместить альтернативный текст типа «извините наш браузер не работает с Java».

Апплету доступны методы суперклассов, наследником которых он является. Фрагмент иерархии классов, предшествующих апплету представляется:

Object -> Component -> Container -> Panel -> Applet

3.4. Разработка приложения Java

Приложения на Java не являются полностью самостоятельными. Запустить такую программу можно при помощи специального интерпретатора в JDK-- это Java.exe.

Пример «самостоятельной программы»

```
class test
{public static void main (string args [ ])
{system.out.println("My test Java");} }
```

Данный текст класса необходимо записать в файл с именем test.Java и откомпилировать командой Javac test.Java . Компилятор создаст файл test.class, который можно запустить, используя интерпретатор Java, командой: **java test** (в команде запуска указывается только имя файла без расширения).

Самостоятельная программа не запускается из браузера.

Замечание JDK использует длинные имена файлов.

Ограничения языка, принятые для апплетов, с целью уменьшения опасности распространения вирусов не распространяются на самостоятельные программы, следовательно, **не все операции, допустимые для самостоятельной программы, может выполнить апплет**.

3.5. Организация низкоуровневого сетевого взаимодействия с использованием сокетов

Множество приложений Интернет используют для обмена информацией протоколы TCP/IP (Transmission Control Protocol/Internet Protocol). Протокол построен на основе сетевых соединений – соединение сначала должно быть установлено, затем осуществляется передача данных, после чего соединение разрывается (действия, похожие на звонок по телефонной линии). Другой тип сетевого протокола – UDP(User DataGram Protocol) работает без установки сетевого соединения и больше напоминает процесс отправки обычной телеграммы.

Язык Java развивается в настоящее время как объектно-ориентированный язык профессионального программирования, предназначенный в первую очередь для использования в Интернет, и является одним из самых широко

применяемых языков для разработки приложений, основанных на сетевом взаимодействии, главным достоинством которого является распределение процесса обработки данных между несколькими компьютерами. Организация такого взаимодействия базируется на процессах передачи данных и программ между приложениями, работающими на разных узлах сети. Передача данных по сети - сложный процесс, включающий в себя определение пути доставки данных, организацию взаимодействия, алгоритмы синхронизации, обработки сбойных ситуаций и т.п. Программное обслуживание такого процесса сложное. Для упрощения введено **понятие сокета (гнездо)** как конечной точки коммуникации. Каждый из сокетов определяется типом и ассоциированным с ним процессом. Реально для передачи организуются определенные дескрипторы TCP-соединения, так называемого гнезда (socket): гнездо сервера и гнездо клиента, которые в Internet-домене включают в себя **IP-адреса сервера и клиента и номера портов**, через которые они взаимодействуют. Сервер обычно имеет закрепленный и постоянный во взаимодействии номер порта, а клиенту, обращающемуся по этому номеру для связи к серверу, назначается некоторый другой (эфемерный) номер порта после установления соединения с сервером на сеанс их взаимодействия и таким образом основной порт освобождается для установления последующих связей (номер порта выбирается сервером из числа незанятых номеров портов в диапазоне от **1024 до 65535**). Эта комбинация (**IP-адрес и номера порта)** однозначно определят **отдельные сетевые процессы** в сети Internet (номера портов до 1024, как правило, резервируются для широко известных приложений, например, 80 – для связывания с серверами Web по протоколу HTTP).

Сокеты для работы в сети можно создать 2-х типов:

- Потоковые для TCP-соединения;
- Датаграммные.

TCP могут передавать данные только между 2-мя приложениями, так как они предполагают наличие логического канала между этими приложениями. Датаграммные позволяют передавать всем узлам сети.

Для датаграмм не нужно создавать канал, данные посылаются приложению с использованием адреса, состоящего из сокета и номера порта. (В датаграммах не гарантируется доставка и корректность последовательности передачи пакетов). Для передачи датаграмм не нужны ни механизмы подтверждения связи, ни механизмы управления потоком данных.

Создавая серверное и клиентское приложение (здесь используется термин приложения, в смысле приложения Java) через сокеты, мы фактически разрабатываем примитивный браузер и сервер сети. *С помощью клиентской части можно обращаться не только к разработанной нами серверной, но и к любому WEB серверу сети, передавая сообщение в порт, который прослушивается данным WEB-сервером.*

3.6. TCP-соединение

Чтобы установить **TCP-соединение**, необходимо:

Создать гнездо сервера, указав порт, по которому гнездо должно ожидать обращения и перевести его в режим ожидания запроса на связь от клиента.

Создать гнездо клиента и обратиться с его помощью (с помощью методов соответствующего класса) к гнезду сервера. После установления контакта ОС присвоит гнезду клиента номер порта, не используемый другими программами.

Связать с обоими гнездами входной и выходной потоки.

Передать данные в выходной поток, связанный с сокетом на локальном компьютере, после чего они будут доступны через входной поток на удаленном компьютере.

После передачи всех данных удалить гнездо (закрыть соединение).

Java позволяет организовать гнездо и связать с ним входной и выходной потоки данных. Для организации гнезда с именем «**theConnection**» на клиенте используется объект класса Socket, который может быть создан оператором, подобным следующему:

Socket theConnection=new Socket(адрес_сервера, номер порта);

Для организации гнезда с именем «**httpServer»** на сервере можно воспользоваться серверным классом ServerSocket следующим образом:

ServerSocket httpServer=new ServerSocket(port);

Где «port» - переменная целого типа в диапазоне от **1024 до 65535**, которая определяет номер незанятого порта для прослушивания сервером. Если интерпретатор Java не может связать программу с указанным портом, например, он уже занят другой программой, то генерируется исключительная ситуация (IOException). Номер порта может быть передан в командной строке запуска программы или его значение может быть присвоено переменной port каким-либо другим способом, предусмотренным в программе.

После связывания программы с портом можно вызвать метод accept() для объекта httpServer, имеющего тип ServerSocket, и организовать прослушивание сети на обращение клиента по указанному порту, и далее на работу по приему передачи данных указанного порта:

```
Socket reg=httpServer.accept();  //Прослушивание (ожидание
 // запроса на соединения, если запрос появился, то)
// Прием содержания соединения клиента (см. далее организация //потоков и ввод вывод байтов).
….
//Отправка клиенту сообщения
…
//Разрыв соединения
```

В представленном плане после установки соединения с клиентом метод accept() разблокируется и возвращает объект класса Socket (в данном случае reg), с помощью которого можно создавать и использовать байтовые и символьные потоки для обмена данными с клиентами. Для этого с гнездом связываются входной и выходной потоки, которые реализуются с помощью классов **InputStream и OutputStream**. При работе с Java нет необходимости программировать потоки ввода-вывода и связывать их с гнездом. Они создаются автоматически при создании гнезда и можно воспользоваться их

методами, чтобы получить информацию об имеющихся в распоряжении потоках:

InputStream inputstream = reg.getInputStream();

OutputStream outputstream=reg.getOutputStream().

Получив ссылку на объекты, реализующие потоки, можно воспользоваться предоставляемыми соответствующими классами потоков методами, чтобы организовать взаимодействие по сети. Например, организовать чтение байта из входного потока можно при помощи метода read: **int c= inputstream.read();,** а запись байта в поток – с использованием метода write : **outputstream. write(c);**

```
fo.close();
byte d[]=new byte[2];  //объявление массива байт длиною 2 символа
(new String("OK\n")).getBytes(0,2,d,0);  // запись OK в d
outp.write(d);}    //запись в поток двух символов фразы «OK»
```

Заметим, что символ, отправленный в выходной поток сокета клиента, становиться доступным для ввода через сокет сервера и наоборот.

В процессе работы программы могут возникнуть исключительные ситуации (ситуации, не позволяющие продолжить нормальный ход работы программы). По умолчанию в таких случаях интерпретатор Java выводит соответствующее сообщение и завершает работу. Чтобы обработать такие ситуации, необходимо фрагменты кода, которые могут вызвать исключительные ситуации, поместить в блок **try,** после которого должен размещаться хотя бы один блок **catch**

```
Try {
//Программный блок, который может вызвать возникновение
//исключительной ситуации
}
catch (описание исключения) { //… блок операторов для  обработки
// исключения …}
```

При нормальном завершении блока try блок catch пропускается. Для каждой исключительной ситуации предусмотрен специальный класс, объект которого объявляется в скобках после ключевого слова catch (смотри пример сокета в прил. 2).

3.7. Соединение с использованием Datagram

При организации соединения при помощи датаграмм необходимо воспользоваться 2-мя классами: DatagramPacket и DatagramSocket и инкапсулированными в них методами.

4. Основы XML, XSL

Язык разметки документов - это набор специальных инструкций, называемых тэгами, предназначенных для формирования в документах какой-либо структуры и определения отношений между различными элементами этой структуры. Тэги языка, или, как их иногда называют, управляющие дескрипторы, в таких документах каким-то образом кодируются, выделяются относительно основного содержимого документа и служат в качестве

инструкций для программы, производящей показ содержимого документа на стороне клиента. В самых первых системах для обозначения этих команд использовались символы “<” и “>”, внутри которых помещались названия инструкций и их параметры. Сейчас такой способ обозначения тэгов является стандартным.

Использование гипертекстовой разбивки текстового документа в современных информационных системах во многом связано с тем, что гипертекст позволяет создавать механизм нелинейного просмотра информации. В таких системах данные представляются не в виде непрерывного потока текстовой информации, а набором взаимосвязанных компонентов, переход по которым осуществляется при помощи гиперссылок.

Самый популярный на сегодняшний день язык гипертекстовой разметки – HTML, был создан специально для организации информации, распределенной в сети Интернет, и является одной из ключевых составляющих технологии WWW. С использованием гипертекстовой модели документа способ представления разнообразных информационных ресурсов в сети стал более упорядочен, а пользователи получили удобный механизм поиска и просмотра нужной информации.

HTML является упрощенной версией стандартного общего языка разметки – SGML (Standart Generalised Markup Language), который был утвержден ISO в качестве стандарта еще в 80-х годах. Этот язык предназначен для создания других языков разметки, он определяет допустимый набор тэгов, их атрибуты и внутреннюю структуру документа. Контроль за правильностью использования дескрипторов осуществляется при помощи специального набора правил, называемых DTD – описаниями(более подробно о DTD мы поговорим чуть позже), которые используются программой клиента при разборе документа. Для каждого класса документов определяется свой набор правил, описывающих грамматику соответствующего языка разметки. С помощью SGML можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате. Но ввиду некоторой сложности SGML использовался в основном для описания синтаксиса других языков(наиболее известным из которых является HTML) и немногие приложения работали с SGML-документами напрямую.

Однако современные приложения нуждаются не только в языке представления данных на экране клиента, но и в механизме, позволяющем определять структуру документа, описывать содержащиеся в нем элементы. HTML обладает несложным набором команд и вполне успешно справляется с задачей описания текстовой информации и отображением ее на экране программы просмотра- браузера. Однако сами отображаемые данные никак не связаны с теми тэгами, которые используются для форматирования, поэтому у программ-анализаторов нет возможности использовать тэги HTML для поиска нужных нам фрагментов документа. То есть, встретив, например, такое описание rose, программа просмотра будет знать, каким цветом отобразить текст, содержащийся внутри тэгов и,

вероятно, отобразит его правильно, но ей абсолютно безразлично, в каком месте документа встретился этот тэг, в какие другие тэги заключен текущий фрагмент, существуют ли вложенные в него фрагменты, правильно ли построены отношения между объектами. Такое "безразличие" к структуре документа приводит к тому, что поиск или анализ информации внутри него ничем не будет отличаться от работы со сплошным, не разбитым на элементы текстовым файлом. А это, как известно, не самый эффективный способ работы с информацией.

Другим существенным недостатком HTML можно назвать ограниченность набора его тэгов. DTD- правила для HTML определяют фиксированный набор дескрипторов и поэтому у разработчика нет возможности вводить собственные, специальные тэги. Хотя время от времени появляются новые расширения языка(на сегодняшний день последней версией HTML является HTML 4.0), но долгий путь их стандартизации, сопровождаемый постоянными разногласиями между основными производителями браузеров, делают практически невозможной быструю адаптацию языка, его использование для отображения специализированной информации(например, мультимедийной, математических, химических формул и т.д.).

XML (*Extensible Markup Language*) - это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Этот язык используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. То есть сам по себе XML не содержит никаких тэгов, предназначенных для разметки, он просто определяет порядок их создания. Таким образом, если, например, мы считаем, что для обозначения элемента *rose* в документе необходимо использовать тэг <*flower*>, то XML позволяет свободно использовать определяемый нами тэг и мы можем включать в документ фрагменты, подобные следующему:

```
<flower>rose</flower>
```

Набор тэгов может быть легко расширен. Если, предположим, мы хотим также указать, что описание цветка должно по смыслу идти внутри описания оранжереи, в которой он цветет, то просто задаем новые тэги и выбираем порядок их следования

```
<conservatory>
<flower>rose</flower>
</conservatory>
```

Если мы хотим посадить туда еще несколько цветков, то должны внести следующие изменения:

```
<conservatory>
<flower>rose</flower>
<flower>tulip</flower>
<flower>cactus</flower>
</conservatory>
```

Как видно, сам процесс создания XML документа очень прост и требует от нас лишь базовых знаний HTML и понимания тех задач, которые мы хотим выполнить, используя XML в качестве языка разметки. Таким образом, у разработчиков появляется уникальная возможность определять собственные

команды, позволяющие им наиболее эффективно определять данные, содержащиеся в документе. Автор документа создает его структуру, строит необходимые связи между элементами, используя те команды, которые удовлетворяют его требованиям и добивается такого типа разметки, который необходим ему для выполнения операций просмотра, поиска, анализа документа.

Еще одним из очевидных достоинств XML является возможность использования его в качестве универсального языка запросов к хранилищам информации. Сегодня в глубинах W3C находится на рассмотрении рабочий вариант стандарта XML-QL(или XQL), который, возможно, в будущем составит серьезную конкуренцию SQL. Кроме того, XML-документы могут выступать в качестве уникального способа хранения данных, который включает в себя одновременно средства для разбора информации и представления ее на стороне клиента. В этой области одним из перспективных направлений является интеграция Java- и XML-технологий, позволяющая использовать мощь обеих технологий при построении машинно-независимых приложений, использующих, кроме того, универсальный формат данных при обмене информации.

XML позволяет также осуществлять контроль за корректностью данных, хранящихся в документах, производить проверки иерархических соотношений внутри документа и устанавливать единый стандарт на структуру документов, содержимым которых могут быть самые различные данные. Это означает, что его можно использовать при построении сложных информационных систем, в которых очень важным является вопрос обмена информацией между различными приложениями, работающими в одной системе. Создавая структуру механизма обмена информации в самом начале работы над проектом, менеджер может избавить себя в будущем от многих проблем, связанных с несовместимостью используемых различными компонентами системы форматов данных.

Если вы знакомы с HTML, изучение XML не потребует от вас особых усилий. Хотя XML, безусловно, сильно отличается по своим возможностям и предназначению от языка гипертекстовой разметки, оба эти языка являются подмножествами SGML, и, следовательно, наследуют его базовые принципы.

Простейший XML- документ может выглядеть так, как это показано ниже:

```
<?xml version="1.0"?>
<list_of_items>
<item id="1"><first/>Первый</item>
<item id="2">Второй <sub_item>подпункт 1</sub_item></item>
<item id="3">Третий</item>
<item id="4"><last/>Последний</item>
</list_of_items>
```

Обратите внимание на то, что этот документ очень похож на обычную HTML-страницу. Так же как и в HTML, инструкции, заключенные в угловые скобки, называются тэгами и служат для разметки основного текста документа. В XML существуют открывающие, закрывающие и пустые тэги (в HTML

понятие пустого тэга тоже существует, но специального его обозначения не требуется).

Тело документа XML состоит из элементов разметки (markup) и непосредственно содержимого документа - данных (content). XML-тэги предназначены для определения элементов документа, их атрибутов и других конструкций языка. Более подробно о типах применяемой в документах разметки мы поговорим чуть позже.

Любой XML-документ должен всегда начинаться с инструкции <?xml?>, внутри которой также можно задавать номер версии языка, номер кодовой страницы и другие параметры, необходимые программе-анализатору в процессе разбора документа

В общем случае XML-документы должны удовлетворять следующим требованиям:

- В заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация.
- Каждый открывающий тэг, определяющий некоторую область данных в документе, обязательно должен иметь своего закрывающего "напарника", то есть в отличие от HTML нельзя опускать закрывающие тэги.
- В XML учитывается регистр символов.
- Все значения атрибутов, используемых в определении тэгов, должны быть заключены в кавычки.
- Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов.
- Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные и поэтому учитываются все символы форматирования (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML).

Если XML-документ не нарушает приведенные правила, то он называется *формально-правильным* и все анализаторы, предназначенные для разбора XML-документов, смогут работать с ним корректно.

Однако кроме проверки на формальное соответствие грамматике языка, в документе могут присутствовать средства контроля над содержанием документа, за соблюдением правил, определяющих необходимые соотношений между элементами и формирующих структуру документа. Например, следующий текст, являясь вполне правильным XML- документом, будет абсолютно бессмысленным:

```
<country><title>Russia</title><city><title>Novosibirsk</country>
</title></city>
```

Для того чтобы обеспечить проверку корректности XML-документов, необходимо использовать анализаторы, производящие такую проверку и называемые верифицирующими.

На сегодняшний день существует два способа контроля правильности XML-документа: DTD-определения(Document Type Definition) и схемы

данных(Semantic Schema). Более подробно об использовании DTD и схемах мы поговорим в следующих разделах. В отличие от SGML определение DTD-правил в XML не является необходимостью, и это обстоятельство позволяет нам создавать любые XML- документы, не ломая пока голову над весьма непростым синтаксисом DTD.

4.1. Конструкции языка

Содержимое XML-документа представляет собой набор элементов, секций CDATA, директив анализатора, комментариев, спецсимволов, текстовых данных. Рассмотрим каждый из них подробней.

4.2. Элементы данных

Элемент - это структурная единица XML- документа. Заключая слово rose в в тэги <flower> </flower> , мы определяем непустой элемент, называемый <flower>, содержимым которого является *rose*. В общем случае в качестве содержимого элементов может выступать как просто какой-то текст, так и другие, вложенные, элементы документа, секции CDATA, инструкции по обработке, комментарии, - т.е. практически любые части XML- документа.

Любой непустой элемент должен состоять из начального, конечного тэгов и данных, между ними заключенных. Например, следующие фрагменты будут являться элементами:

```
<flower>rose</flower>
<city>Novosibirsk</city>
```

А эти - нет:

```
<rose>
<flower>
rose
```

Набором всех элементов, содержащихся в документе, задается его структура и определяются все иерархическое соотношения. Плоская модель данных превращается с использованием элементов в сложную иерархическую систему со множеством возможных связей между элементами. В следующем примере мы описываем месторасположение новосибирских университетов (указываем, что новосибирский университет расположен в городе Новосибирске, который, в свою очередь, находится в России), используя для этого вложенность элементов XML :

```
<country id="Russia">
 <cities-list>
<city>
<title>Новосибирск</title>
<state>Siberia</state>
<universities-list>
<university id="2">
<title>Новосибирский Государственный Технический Университет</title>
<noprivate/>
<address URL="www.nstu.ru"/>
<description>очень хороший институт</description>
```

```
</university>
<university id="2">
<title>Новосибирский Государственный Университет</title>
<noprivate/>
<address URL="www.nsu.ru"/>
<description>тоже не плохой</description>
</university>
</universities-list>
</city>
</cities-list>
</country>
```

Производя впоследствии поиск в этом документе, программа клиента будет опираться на информацию, заложенную в его структуру - используя элементы документа. То есть если, например, требуется найти нужный университет в нужном городе, используя приведенный фрагмент документа, то необходимо будет просмотреть содержимое конкретного элемента <university>, находящегося внутри конкретного элемента <city>. Поиск при этом, естественно, будет гораздо более эффективен, чем нахождение нужной последовательности по всему документу.

В XML-документе, как правило, определяется хотя бы один элемент, называемый корневым и с него программы-анализаторы начинают просмотр документа. В приведенном примере этим элементом является <country>

В некоторых случаях тэги могут изменять и уточнять семантику тех или иных фрагментов документа, по-разному определяя одну и ту же информацию и тем самым предоставляя приложению-анализатору этого документа сведения о контексте использования описываемых данных. Например, прочитав фрагмент **<city>**Holliwood**</city>,** мы можем догадаться, что речь в этой части документа идет о городе, а вот во фрагменте **<restaurant>**Holliwood**</restaurant>** - о забегаловке.

В случае, если элемент не имеет содержимого, т.е. нет данных, которые он должен определять, он называется пустым. Примером пустых элементов в HTML могут служить такие тэги HTML, как **
, <hr>, <img>**;. Необходимо только помнить, что начальный и конечные тэги пустого элемента как бы объединяются в один, и надо обязательно ставить косую черту перед закрывающей угловой скобкой (например, **<empty/>**;).

4.3. Комментарии

Комментариями является любая область данных, заключенная между последовательностями символов <!-- и -->. Комментарии пропускаются анализатором и поэтому при разборе структуры документа в качестве значащей информации не рассматриваются.

4.4. Атрибуты

Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то имеется возможность

использовать атрибуты элемента. Атрибут - это пара "название" = "значение", которую надо задавать при определении элемента в начальном тэге. Пример:

```
<color RGB="true">#ff08ff</color>
<color RGB="false">white</color>
```

или

```
<author id=0>Ivan Petrov</author>
```

4.5. Специальные символы

Для того чтобы включить в документ символ, используемый для определения каких-либо конструкций языка (например, символ угловой скобки), и не вызвать при этом ошибок в процессе разбора такого документа, нужно использовать его специальный символьный либо числовой идентификатор. Например, < , > " или $(десятичная форма записи), &#x1a (шестнадцатеричная) и т.д. Строковые обозначения спецсимволов могут определяться в XML-документе при помощи компонентов (entity), о чем мы еще поговорим немного позже.

4.6. Директивы анализатора

Инструкции, предназначенные для анализаторов языка, описываются в XML документе при помощи специальных тэгов - <*?* и *?*>;. Программа клиента использует эти инструкции для управления процессом разбора документа. Наиболее часто инструкции используются при определении типа документа (например, <? Xml version=”1.0”?>) или создании пространства имен[11].

4.7. CDATA

Чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы, но в отличие от комментариев иметь возможность использовать их в приложении, необходимо использовать тэги <![CDATA] и]]>. Внутри этого блока можно помещать любую информацию, которая может понадобиться программе-клиенту для выполнения каких-либо действий (в область CDATA можно помещать, например, инструкции JavaScript). Естественно, надо следить за тем, чтобы в области, ограниченной этими тэгами, не было последовательности символов]].

Как уже отмечалось, в отличие от HTML XML никак не определяет способ отображения и использования описываемых с его помощью элементов документа, т.е. программе-анализатору предоставляется возможность самой выбирать нужное оформление. Этого требует принцип независимости определения внутренней структуры документа от способов представления этой информации. Например, задавая в документе элемент *<flower>роза</flower>*, мы лишь указываем, что *rose* в данном случае является цветком, но информации о том, как должен выглядеть данный элемент документа на экране клиента и должен ли он отображаться вообще, в таком определении нет.

Для того чтобы использовать данные, определяемые элементами XML, например, отображать их на экране пользователя, необходимо написать программу-анализатор, которая бы выполняла эти действия. Уже сегодня таких программ появилось достаточное количество и у разработчиков существует возможность выбора наиболее подходящей из них для решения конкретных проблем

На сегодняшний день существует два подхода к их построению: событийный - *Simple API for XML*, SAX и объектно-ориентированный - DOM(Document Object Model). Рассмотрим их использование на конкретных примерах.

4.8. Объектная модель документа DOM

Одним из самых мощных интерфейсов доступа к содержимому XML документов является Document Object Model - DOM.

Объектная модель XML документов является представлением его внутренней структуры в виде совокупности определенных объектов. Для удобства эти объекты организуются в некоторую древообразную структуру данных - каждый элемент документа может быть отнесен к отдельной ветви, а все его содержимое в виде набора вложенных элементов, комментариев, секций CDATA и т.д. представляется в этой структуре поддеревьями. Так как в любом правильно составленном XML-документе обязательно определен главный элемент, то все содержимое можно рассматривать как поддеревья этого основного элемента, называемого в таком случае корнем дерева документа. Для следующего фрагмента XML документа:

```
<tree-node>  <node-level1>
   <node-level2/>
<node-level2>text</node-level2>   <node-level2/>
  </node-level1>
<node-level1>
   <node-level2>text</node-level2>
<node-level1>
   <node-level2/>
   <node-level2><node-level3/></node-level2>
</node-level1> </tree-node>
```

дерево элементов выглядит так:

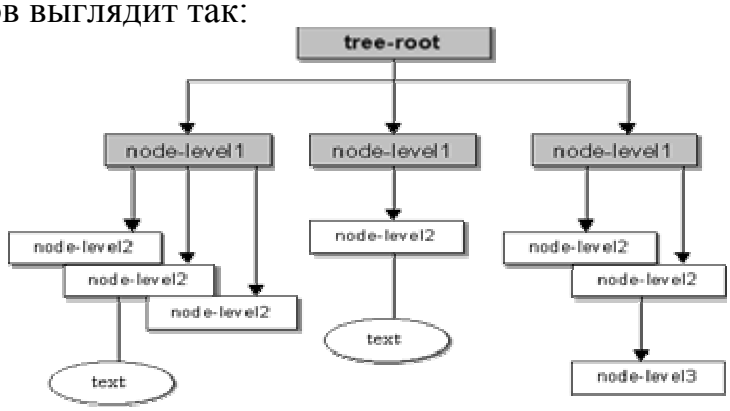


Рис.4.1. Дерево элементов

Объектное представление структуры документа не является чем-то новым для разработчиков. Для доступа к содержимому HTML-страницы в сценариях давно используется объектно-ориентированный подход, - доступные для JavaScript или VBScript элементы HTML-документа могли создаваться, модифицироваться и просматриваться при помощи соответствующих объектов. Но их список и набор методов постоянно изменяется и зависит от типа браузера и версии языка. Для того чтобы обеспечить независимый от конкретного языка программирования и типа документа интерфейс доступа к содержимому структурированного документа в рамках W3 консорциума, была разработана и официально утверждена спецификация объектной модели DOM Level 1.

DOM - это спецификация универсального платформо- и программно-независимого доступа к содержимому документов, которая является просто своеобразным API для их обработчиков. DOM является стандартным способом построения объектной модели любого HTML- или XML- документа, при помощи которой можно производить поиск нужных фрагментов, создавать, удалять и модифицировать его элементы.

Для описания интерфейсов доступа к содержимому XML-документов в спецификации DOM применяется платформо-независимый язык IDL и для использования их необходимо "перевести" на какой-то конкретный язык программирования. Однако этим занимаются создатели самих анализаторов, нам можно ничего не знать о способе реализации интерфейсов - с точки зрения разработчиков прикладных программ DOM выглядит как набор объектов с определенными методами и свойствами.

4.9. SAX

Вторым стандартным интерфейсом для большинства универсальных XML-анализаторов является событийно-ориентированное API SAX - *Simple API for XML*.

Термин «событийно-ориентированный» является ключевым в этом определении и объясняет способ использования SAX. Каждый раз, когда при разборе XML документа анализатор оказывается в каком-то новом состоянии - обнаруживает какую-либо синтаксическую конструкцию XML-документа (элемент, символ, шаблон, и т.д.), фиксирует начало, конец объявлений элементов документа, просматривает DTD-правила или находит ошибку, он воспринимает его как произошедшее событие и вызывает внешнюю процедуру – обработчик этого события. Информация о содержимом текущей конструкции документа передается ему в качестве параметров функции. Обработчик события – это какой-то объект приложения, который выполняет необходимые для обработки полученной из XML информации действия и осуществляет таким образом непосредственный разбор содержимого. После завершения этой функции управление опять передается XML-анализатору и процесс разбора продолжается.

Реализацией этого механизма в Java SAX 1.0 является библиотека классов org.xml.sax. Наследуя классы SAX-совместимого анализатора, мы получаем

универсальный доступ к XML-документу при помощи классов, содержимое и механизм использование которых приведено в соответствующем описании.

Последовательный разбор XML-документа SAX-обработчиком обычно производится по следующей схеме (более подробное описание приведено ниже):

- загрузить документ, установить обработчики событий, начать просмотр его содержимого (если есть DTD-описания, то - их разбор);
- найдено начало документа (его корневой, самый первый элемент) - вызвать виртуальную функцию- обработчик события **startDocument**;
- каждый раз, когда при разборе будет найден открывающий тэг элемента, вызывается обработчик-функция **startElement**. В качестве параметров ей передаются название элемента и список его атрибутов;
- найдено содержимое элемента - передать его соответствующему обработчику - **characters**, **ignorableWhitespace**,**processingInstruction** и т.д.;
- если внутри текущего элемента есть подэлементы, то эта процедура повторяется;
- найден закрывающий тэг элемента - обработать событие **endElement**();
- найден закрывающий тэг корневого элемента -обработать событие **endDocument**;
- если в процессе обработки были обнаружены ошибки, то анализатором вызываются обработчики предупреждений (warning), ошибок (error) и критических ошибок обработчика (fatalError).

Ссылка на объект класса обработчика событий может передаваться объекту XML-анализатора при помощи следующих функций:

- parser.**setDocumentHandler**(event_class); // - обработчик событий документа;
- parser.**setEntityResolver**(event_class); // - обработчик событий загрузки DTD-описаний;
- parser.**setDTDHandler**(event_class); // - обработчик событий при анализе DTD-описаний;
- parser.**setErrorHandler**(event_class); // - обработчик чрезвычайных ситуаций.

Здесь event_class - объект созданного нами ранее класса.

Часто возникает необходимость преобразовать существующий XML-документ в другой документ (например, в HTML-документ или в XML-документ с другой структурой). Для решения даной задачи был предложен язык для описания необходимых преобразований. Этот язык называется XSL. Он базируется на XML, однако имеет уже заранее предопределенные теги. Рассмотрим некоторые из них.

Любой xsl-файл начинается следующим заголовком

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Каждая схема преобразований находится в пределах тега <xsl:template match="…">. В атрибуте match указывается узел, данные которого будут подвергнуты трансформации.

Циклический проход по некоторым узлам можно произвести, используя тег <xsl:for-each select="…">. В атрибуте select указывается имя узлов, которые будут выбираться из xml-документа.

Для выполнения тех или иных действий в зависимости от значения узла предназначен тег <xsl:if test="…">. В атрибуте test задается условие, при котором будут выполняться действия, указанные внутри данного тега.

Для выбора данных из какого-либо узла используется тег <xsl:value-of select="…"/>. Атрибут select указывает, из какого узла будут выбираться данные.

В Java можно работать с xml- и xsl- файлами, используя специальные библиотеки. Наибольшее распространение получили Xerces для работы с xml и Xalan для работы с xsl.

Следующий код загружает xml-файл и трансформирует его используя xsl-template.

```
DOMParser parser=new DOMParser();
parser.parse(xmlFile);
XMLParserLiaison xmlProcessorLiaison=null;
try{
  String parserLiaisonClassName="org.apache.xalan.xpath.xdom.XercesLiaison";
  Class parserLiaisonClass=Class.forName(parserLiaisonClassName);
  Constructor parserLiaisonCtor=parserLiaisonClass.getConstructor(null);
  xmlProcessorLiaison=(XMLParserLiaison)parserLiaisonCtor.newInstance(null);
}catch (Exception e){
  e.printStackTrace();
}
XSLTProcessor
processor=XSLTProcessorFactory.getProcessor(xmlProcessorLiaison);
StylesheetRoot stylesheet=null;
String xslFileName=xslFile;
stylesheet=processor.processStylesheet(xslFileName);
Document doc=parser.getDocument();
XSLTInputSource source=new XSLTInputSource(doc);
XSLTResultTarget dest=new XSLTResultTarget(new
org.apache.xerces.dom.DocumentImpl());
stylesheet.process(source,dest);
m_root=dest.getNode().getChildNodes().item(0);
```

В строке 1 создается парсер xml-файла, который представляет xml-файл в виде DOM-структуры.

В строке 2 xml-файл (xmlFile – имя файла) парсируется.

В строках 4-12 создается xsl-процессор, который мы будем использовать для трансформации.

В строках 13-15 загружается xsl-файл, который описывает преобразование.

В строках 16-19 производится преобразование.

И в строке 20 мы получаем корневой узел полученного xml-документа.

5. Технологии создания динамических Web-приложений (сервлеты и JavaServer Pages)

5.1. Сервлеты Java

Сервлеты – это программы, написанные на Java, которые выполняются на Web-сервере, действуя в качестве посредника между запросом, поступающим от Web-браузера или другого клиента HTTP, и базами данных или приложениями на HTTP-сервере.

Они выполняют следующие действия:

1. *Читают любые данные, переданные пользователем*. Данные обычно вводятся в форму на HTML-странице, но также могут поступить от апплета Java или настраиваемой HTTP-программы клиента.

2. *Просматривают всю информацию о запросе, которая встроена в HTTP-запрос*. Информация включает сведения о возможностях браузера, cookies, имени хоста клиента, делающего запрос, и т.д.

3. *Генерируют результаты*. Этот процесс может потребовать общения с базой данных, выполнения вызова RMI или CORBA, активизации существующего приложения или непосредственного вычисления ответа.

4. *Форматируют результаты внутри документа*. В большинстве случаев этот процесс предполагает встраивание информации на HTML-страницу.

5. *Устанавливают соответствующие параметры HTTP-ответа*. Это означает, что браузеру поступает сообщение о типе возвращаемого документа (например HTML), установке cookies, кэширования параметров и других подобных действий.

6. *Возвращают документ клиенту*. Документ может быть возвращен клиенту в текстовом формате (HTML), двоичном (GIF-изображение) или даже в сжатом формате, например gzip, который помещается на верхний уровень какого-либо другого формата.

5.2. Базовая структура сервлетов

Чтобы быть сервлетом, класс должен наследовать класс HttpServlet и переопределить метод doGet или doPost в зависимости от того, передаются данные методом GET или POST протокола HTTP.

Оба метода имеют два аргумента: объекты классов HttpServletRequest и HttpServletResponse. В классе HttpServletRequest имеются методы, которые позволяют получить сведения о поступающей информации: о данных формы, заголовках HTTP-запроса, имени хоста клиента. Класс HttpServletResponse позволяет задать выходную информацию: коды состояния HTTP, заголовки ответа, и, что самое главное, позволяет получить объект класса PrintWriter, который используется для передачи содержимого документа клиенту.

В листинге 4.1 приведен простейший сервлет, который генерирует обычный текст.

Листинг 4.1. SampleServlet.Java.

```
import Java.io.*;
import Java.servlet.*;
import Java. servlet.http.*;
public class SampleServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response )
        throws ServletException, IOException {
    printWriter out = response.getWriter();
out.println("Hello, world!");
  }
}
```

5.3. Жизненный цикл сервлета

Когда сервлет создается в первый раз, активизируется его метод init, поэтому именно в него необходимо включать код начальной установки (setup), который выполняется один раз.

После этого в результате выполнения каждого запроса пользователя создается отдельный поток, который вызывает метод service ранее созданного экземпляра. Появление нескольких параллельных запросов обычно приводит к тому, что несколько потоков одновременно вызывают метод service, хотя сервлет может реализовать специальный интерфейс, который будет предусматривать запуск только одного потока за один раз.

Затем метод service вызывает метод doGet, doPost или doXXX в зависимости от типа полученного HTTP-запроса.

Наконец, когда сервер решает выгрузить сервлет, он вызывает метод destroy этого сервлета.

5.4. Страницы Java Server Pages

Технология серверных страниц JSP (Java Server Pages) предоставляет возможность смешивать обычные, статические HTML-страницы с динамически генерированным содержимым, полученным из сервлетов. Страница JSP состоит из обычного HTML-кода и частей динамического кода на Java, заключенных в специальные теги, большинство из которых начинается с символов <%, а заканчивается символами %>.

Сразу следует отметить, что страницы JSP не предоставляют таких возможностей, которые в принципе не могли бы быть выполнены сервлетами. Более того, на практике страница JSP автоматически преобразуется в обычный сервлет, при этом ее статический HTML просто направляется в выходной поток, связанный с методом service сервлета. Подобное преобразование, как правило, производится, когда страница запрашивается в первый раз.

Самое большое преимущество использования JSP состоит в том, что эта технология помогает отделить представление от содержания, что позволяет эффективно разделять задачи между разными людьми. Эксперты по дизайну Web-страниц могут создавать HTML, используя привычный инструментарий и

оставляя место для программистов сервлетов, которые вставят впоследствии на страницу динамическое содержимое.

Кроме обычного HTML-кода существуют три основных типа JSP-конструкций, которые обычно встраиваются в страницу: элементы сценария, директивы и действия.

Элементы сценария JSP позволяют вставить код в сервлет, который будет генерироваться из JSP-страницы. Существуют три вида элементов:

Выражения (expressions) вида <%=expression%>, которые вычисляются, а полученные значения вставляются в страницу HTML, генерируемую сервлетом.

Скриплеты (scriplets) вида <% code %>, которые вставляются в метод _jspService сервлета (вызываемый методом service).

Объявления (declarations) вида <%! Code %>, которые вставляются в тело класса сервлета вне каких-либо его существующих методов.

Для упрощения кода в выражениях JSP и скриплетах можно использовать восемь автоматически определенных переменных, иногда называемых неявными объектами (request, response, session, out, application, config, pageContext, page). Поскольку JSP-объявления дают код, который появляется вне метода _jspService, к этим переменным нет доступа из объявлений. Ниже приводятся сведения о каждой переменной:

- **request** – переменная класса HttpServletRequest, предоставляющая доступ к текущему HTTP-запросу;
- **response** – переменная класса HttpServletResponse, предоставляющая доступ к формируемому HTTP-ответу;
- **session** – переменная класса HttpSession, предоставляющая доступ к объекту “сеанс”, ассоциированному с текущим запросом (если только эта возможность не отключена соответствующей установкой значения атрибута session в объявлении page);
- **out** – переменная класса PrintWriter (буферизированная версия класса называется JspWriter), используется для отправки сформированного выходного документа клиенту;
- **application** – эта переменная является объектом класса ServletContext данного сервлета (сервлеты и страницы JSP могут хранить постоянные данные в объекте ServletContext, который совместно используется всеми сервлетами в Web-приложении);
- **config –** является объектом класса ServletConfig для данной страницы;
- **pageContext** – данная переменная содержит ссылку на объект класса PageContext, связанного с текущей страницей (класс PageContext специфичен для страниц JSP, он задает единственую точку доступа ко многим атрибутам страницы и предоставляет удобное место для хранения совместно используемых данных);
- **page** –синоним this в языке программирования Java.

Директивы страницы JSP влияют на общую структуру сервлета, в который транслируется страница JSP. Используется три директивы: page, include и taglib.

Действия позволяют задать существующие компоненты, которые будут использоваться и другим способом контролировать поведение процессора JSP.

5.5. Интегрирование сервлетов и JSP

Итак, сервлеты представляют собой очень мощное средство для программирования. Однако генерация HTML-кода с помощью сервлетов может оказаться трудоемкой и привести к результату, который трудно модифицировать. Тогда приходит черед JSP-страниц, использование которых позволяет отделить представление от динамического содержимого.

Однако следует отметить, что страница JSP имеет относительно фиксированный внешний вид. В сложных же приложениях, которым может потребоваться несколько совершенно различных представлений, оптимальным является решение, когда сервлет будет обрабатывать исходный запрос и настраивать bean-компоненты, а затем в зависимости от обстоятельств передавать результаты одной из JSP-страниц.

Таким образом, наибольшую выгоду от этих технологий мы получим, если будем использовать JSP и сервлеты совместно, применяя JSP для реализации уровня представления и сервлеты для объемной обработки данных. Такой подход к проектированию Web-приложений получил в спецификации Java название JSP Model 2 architecture.

Перенаправление запросов сервлетами и включение внешнего содержимого обеспечивается специальным объектом класса RequestDispatcher. В сервлетах для получения ссылки на этот объект следует вызвать метод getRequestDispatcher объекта класса ServletContext, задавая URL относительно корневого каталога сервера. Затем можно использовать методы forward для полной передачи управления ресурсу, ассоциированному с соответствующим адресом URL, и include для вывода содержимого, соответствующего этому URL. В обоих случаях в качестве параметров передаются объекты классов HttpServletRequest и HttpServletResponse. Пример использования:

```
String adr = “/Laba/index.jsp”;
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher(adr);
dispatcher.forward(request, response);
```

Для перенаправления запросов со страниц JSP можно также пользоваться классом RequestDispatcher, встраивая показанный выше код в скриплет, однако существует специальный более простой синтаксис:

```
<jsp:forward page=”/Laba/LabaServlet”>
```

5.6. Установка и настройка программного обеспечения для поддержки сервлетов и JSP

Перед началом работы следует установить необходимое программное обеспечение и сконфигурировать систему.

Данная лабораторная работа ориентирована на использование серверов Apache и Tomcat. Tomcat является официальной справочной реализацией

спецификаций Servlet 2.2 и JSP 1.1. Его можно использовать как небольшой автономный сервер для тестирования сервлетов и страниц JSP или интегрировать его в Apache Web-сервер.

Конфигурирование сервера

Перед запуском сервера необходимо корректно установить некоторые параметры, которые полностью зависят от типа сервера. В случае Tomcat это:

- номер порта – по умолчанию используется порт 8080;
- переменная JAVA_HOME – должна указывать на каталог установки JDK.

Каталог Web-приложения

Для хранения файлов (jsp и html) Web-приложения необходимо создать каталог в директории <tomcat_install_dir>/webapps. Использовать страницы JSP очень просто: если есть файл с расширением .jsp, то его надо только поместить в этот каталог и обращаться, используя указатель URL следующего вида: http://localhost/<webapp_dir>/<file_name>.jsp. Не требуется никакой компиляции и настроек, однако сервер должен быть сконфигурирован так, чтобы иметь доступ к файлам классов сервлетов и JSP и компилятору Java.

Замечание: Существует возможность просмотреть исходный код сервлетов, сгенерированных из страниц JSP. Его можно найти в директории <tomcat_install_dir>/work/localhost/<webapp_dir>.

Компиляция и установка сервлетов

Скомпилированные классы сервлетов должны быть помещены в специальный директорий, местоположение которого зависит от используемого сервера. В Tomcat это каталог <webapp_dir>/WEB-INF/classes.

Замечание: При каждом изменении уже существующего сервлета следует перезапустить сервер для того, чтобы эти изменения вступили в силу.

В директории <webapp_dir>/WEB-INF также должен находиться файл web.xml, в котором прописывается сервлет. Например, следующим образом надо прописать сервлет тестового примера (раздел 1.4.5) из приложения:

```
<servlet>
    <servlet-name>
     LabaServlet
    </servlet-name>
    <servlet-class>
LabaServlet
    </servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>
     LabaServlet
    </servlet-name>
    <url-pattern>
     /LabaServlet
    </url-pattern>
```

</servlet-mapping>

Активизировать сервлет теперь можно, используя указатель URL следующего вида: http://localhost/<webapp_dir>/LabaServlet.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

1. Лабораторная работа №1

Тема: Введение в технологию HTML

Цели работы:

1. Изучить основы HTML, CSS, закрепить знания написанием интерактивных страничек.
2. Освоить элементы VBScript, JScript, COM, DOM, научиться использовать их совместно с HTML и создавать динамические страницы (DHTML), закрепить знания написанием динамических страниц.

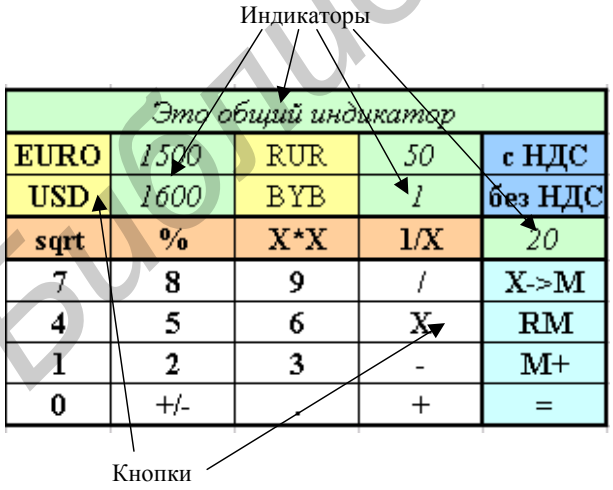
Варианты индивидуальных заданий:

1. Калькулятор индивидуального предпринимателя:

отличается от стандартного наличием следующих функций: мультивалютный пересчёт, расчёт суммы с НДС и без НДС (рис 1.).

2. Калькулятор программиста:

отличается от стандартного наличием следующих функций: перевод текущего числа в различные системы счислений (посредством специальных кнопок), а также в формат даты или времени, перевод цветов из RGB в CMYK и обратно.



Это общий индикатор				
EURO	*1500*	RUR	*50*	с НДС
USD	*1600*	BYB	*1*	без НДС
sqrt	%	X*X	1/X	*20*
7	8	9	/	X->M
4	5	6	X	RM
1	2	3	-	M+
0	+/-	.	+	=

Это общий индикатор				
BIN	HEX	RND	Data	/
OCT	DEC	1/X	Time	:
sqrt	%	X*X	RGB	CMYK
7	8	9	/	X->M
4	5	6	X	RM
1	2	3	-	M+
0	+/-	.	+	=

Рис. 1. Калькулятор ИП *Рис. 2. Калькулятор программиста*

3. Калькулятор электротехника.
4. Ёлочная гирлянда: используя 6 ярких цветов (без картинок, при помощи только изменения цвета объектов), изобразить гирлянду на 60 лампочек. Обязательно предусмотреть следующие варианты работы гирлянды:

бегущие огни, случайное мигание, переливание разными цветами. Предусмотреть кнопку выключения, а также возможность задания скорости.

5. Змейка: набираем предложение в поле ввода, нажимаем кнопку «пуск», далее оно по буквам перемещается по экрану в различных направлениях (как по вертикали, так и по диагонали). Предусмотреть кнопки «пауза» и «стоп».

6. Гостевая книга: вверху страницы - стандартные поля ввода (ФИО, email, тема и текст сообщения, кнопки добавить и очистить), после нажатия кнопки «добавить новое сообщение» оно появляется в нижней части страницы первой по порядку. Каждое добавленное сообщение помнит дату и время добавления.

7. Телефонный справочник: предусмотреть возможность добавления нового телефонного номера и поиска, (поля: телефон, ФИО, улица, дом, корпус, квартира).

8. Расписания занятий студентов БГУИР: предусмотреть 2 режима работы – режим редактирования и режим просмотра.

9. Мастер Гистограмм.

10. Игра «Поймай предмет».

11. Крестики-нолики – 5x5.

12. Шахматы.

13. Шашки.

2. Лабораторная работа №2

Тема: Технологии организации сетевых соединений. Сокеты

Цели работы:

1. Изучить теоретические основы организации сетевых соединений.
2. Получить практические навыки в области разработки программ организации сетевого взаимодействия на языке Java.

Для выполнения данной лабораторной работы требуется знание основ программирования на C++ на уровне разработки несложных объектно-ориентированных программ либо языка Java, а также принципов организации компьютерной сети, теоретических сведений по основам передачи данных и взаимодействии узлов в сети Интернет.

В теоретической части лабораторной работы рассмотрены основные возможности языка Java и принципы организации сетевого соединения с использованием протокола TCP и сокетов. Цель лабораторной работы – освоение программирования простейших приложений на Java, в том числе и приложений для организации сетевого взаимодействия через сокеты. Решаемые задачи – получение навыков разработки приложений на Java и более глубокое понимание форм и методов организации взаимодействия компьютеров в сети.

Задание к лабораторной работе: Разработать приложение, позволяющее осуществлять взаимодействие клиента и сервера по совместному решению задач обработки информации.

Приложение должно располагать:

1. «Минимальным» пользовательским интерфейсом, определяющим возможности приложения;
2. Возможностью передачи и модифицирования получаемых (передаваемых) данных.
3. Средствами диагностики и обработки исключительных ситуаций, а также средствами поддержки деятельности пользователя.

Варианты индивидуальных заданий:

1. Разработать интерфейс для обмена сообщениями между пользователями различных узлов сети.
2. Разработать программу организации простых расчетов на сервере для клиентских задач.
3. Разработать апплет, который высылается на клиентскую машину и обеспечивает интерфейс для взаимодействий с сервером (следует учесть, что апплет может работать только с тем сервером, с которого он загружен, и настройки соответствующего браузера).
4. Организовать взаимодействие, реализующее рассылку сообщений от одного клиента группе клиентов.
5. Организовать пересылку журнала репликаций БД между клиентами Remote-установки через сокеты.

6…10 В пп. 6-10 реализовать соответствующие взаимодействия (пп. 1-5) с использованием Датаграмм.

Примеры реализации взаимодействияя через сокеты см. в прил. 1.

3. Лабораторная работа №3

Тема: Основы XML, XSL

Цели работы:

1. Изучить теоретические основы использования языка разметки XML.
2. Получить практические навыки в области создания простейшего Java-приложения для разбора xml-файла.

Варианты индивидуальных заданий:

1. Создать xml-файл, описывающий структуру вуза (факультет, группа, студент). Написать xsl-шаблон для выбора студентов по факультетам. Результат вывести в консоль.
2. Создать xml-файл, описывающий структуру вуза (факультет, группа, студент). Написать xsl-шаблон для выбора студентов по факультетам в HTML-файл.

3. Создать xml-файл, хранящий языки и денежные знаки для стран. Написать xsl-шаблон для выбора отдельно языков по странам и денежных знаков по странам. Результат вывести в консоль.

4. Создать xml-файл, хранящий языки и денежные знаки для стран. Написать xsl-шаблон для выбора отдельно языков по странам и денежных знаков по странам в HTML-файл.

5. Создать xml-файл, описывающий структуру вуза (факультет, группа, студент). Написать xsl-шаблон для выбора студентов по факультетам. Результат вывести в xsl-файл.

6. Создать xml-файл, описывающий структуру комплектующих готового изделия. Изделие состоит из сборочных единиц и комплектующих. Сборочные единицы состоят из комплектующих. Написать xsl-шаблон для выбора комплектующих по готовым изделиям. Результат вывести в консоль.

7. Создать xml-файл, описывающий структуру комплектующих готового изделия. Изделие состоит из сборочных единиц и комплектующих. Сборочные единицы состоят из комплектующих. Написать xsl-шаблон для выбора комплектующих по готовым изделиям в HTML-файл.

8. Создать xml-файл, описывающий следующую структуру: аэропорт-рейс-экипаж. Написать xsl-шаблон для выбора любого члена экипажа (командир, бортинженер, второй пилот и т.д.) по аэропортам. Результат вывести в консоль.

9. Создать xml-файл, описывающий следующую структуру: аэропорт-рейс-экипаж. Написать xsl-шаблон для выбора любого члена экипажа (командирб бортинженер, второй пилот и т.д.) по аэропортам в HTML-файл.

10. Создать xml-файл, описывающий следующую структуру: аэропорт-рейс-экипаж. Написать xsl-шаблон для выбора любого члена экипажа (командирб бортинженер, второй пилот и т.д.) по аэропортам. Результат вывести в консоль.

4. Лабораторная работа №4

Тема: Технологии создания динамических Web-приложений (сервлеты и JavaServer Pages).

Цель работы:

1. Изучить теоретические основы динамического проектирования Web-страниц.
2. Получить практические навыки в области создания Web-приложений с использованием Java-технологий (сервлетов и Java Server Pages).

Задание к лабораторной работе: Разработать Web-приложение, позволяющее осуществлять взаимодействие с базой данных в соответствии с темой индивидуального задания. В приложении должны быть реализованы следующие функции:

1. Просмотр, добавление, удаление и редактирование записей (по аналогии с тестовым примером).

2. Возможность вывода информации из базы данных в отсортированном виде по кажому полю в зависимости от выбора пользователя (обновление должно происходить при «клике» мышью на соответствующем столбце в заголовке таблицы).

3. Поиск по каждому полю.

Варианты индивидуальных заданий: Приложение должно быть основано на базе данных по учету:

1. Поступлений компакт-дисков в музыкальный магазин.
2. Товаров на складе мебельного магазина.
3. Продаж телевизоров в магазине техники.
4. Информации о студентах.
5. Информации о сотрудниках кафедры.
6. Автомобилей в ГАИ.
7. Комплектующих в компьютерной фирме.
8. Поступлений цветов в цветочный магазин.
9. Наличия медикаментов в аптеке.
10. Кредитных операций в банке.
11. Поступления товаров в обувном магазине.
12. Продажи билетов в кинотеатр.

ЛИТЕРАТУРА

1. Холл. М. Сервлеты и JavaServer Pages. СПб. Питер, 2001.
2. Homer. A. XML in IE5. Programmer’s Reference, Wrox Press Ltd., 1999.
3. Морган. М. Java 2. Руководство разработчика. М., 1999.
4. http://www.citforum.ru.
5. http://www.Javable.com.
6. http://msdn.microsoft.com.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1. Пример программы организации соединений по протоколу TCP/IP.

Следующий пример демонстрирует работу в архитектуре клиент-сервер с использованием TCP – протокола и сокетов.

Пусть программа манипулирования с файлами данных располагается на одном компьютере, а программы, которым нужен этот файл для обработки - на другом компьютере. Очевидно, что в данной ситуации должны участвовать две программы: клиент на стороне прикладной программы, обрабатывающей файлы, и сервер на узле, манипулирующем файлами. Программа клиент может посылать для хранения на сервер (в заголовке переылаемого сообщения пересылается слово SEND) или читать с сервера (в заголовке RESEIVE не анализируется. Считается, что если в заголовке не SEND, то значит, это RESEIVE-запрос на получение файла с сервера). Вслед за командой пересылается имя файла, к которому при сохранении на сервере добавляется расширение .day, а при чтении с сервера разыскивается файл с прочитанным из пришедшего на сервер сообщения имени файла и расширения .day. Сервер должен обеспечить прием и хранение в файле или передачу данных из затребованного файла.

Запуск приложений осуществляется из командной строки. В приложении номер порта, прослушиваемого сервером (в серверной части программы), задается фиксированный, хотя можно организовать передачу его и через параметры командной строки, как в клиентской программе, где предусмотрена либо посылка сообщения по номеру порта 1234 по умолчанию, либо передача номера порта через параметры командной строки при запуске клиентского приложения.

Создадим программу серверной части для сформулированной задачи.

```
import Java.lang.*;
import Java.net.*;
import Java.io.*;
class OrganizerServer {     //объявляем класс
public static void main(String args[]) { //стандартный заголовок приложения
int c,n;
ServerSocket httpServer;  //объект класса ServerSocket
FileOutputStream fo;     //объявление объект класса
FileInputStream fi;
Socket reg;
int port=1234;//переменная, используемая далее для задания
        //номера порта, через который сервер будет прослушивать
        //сеть в ожидании обращения (в данном случае порт 1234
String header, fname;
int nbytes, i;
trym {     //блок, в котором может возникнуть исключительная ситуация
httpServer=new ServerSocket(port); //httpServer сокет сервера,
                                   //прослушивающий порт за номером 1234
for(;;){       // заголовок "вечного" цикла, сервер работает постоянно,
```

```
                        // ожидая соединения со стороны клиента
reg=httpServer.accept(); //reg - гнездо, через которое
                                //установлено соединение с клиентом
System.out.println("есть клиент");
InputStream inp=reg.getInputStream(); // inp - входной поток
OutputStream outp=reg.getOutputStream(); //outp - выходной поток
header=new String(); //резервирование строки для заголовка
      //пока не конец первой строки
while (((c=inp.read()) !='\n') && (c !=-1))  header +=(char)c;
   // пока не конец файла - читать байт и
       //дописывать его  в формируемую строку командного слова
System.out.println("заголовок прочитан");
if (header.startsWith("SEND"))    //чтение прчитанной команды
{System.out.println("Polucheno SEND");  // если send, то пропустить это
  //слово (SEND)и пробелы (trim()), затем выделить имя файла
header=header.substring(new String("SEND").length()).trim();
// длиной 8 символов и дополнить его расширением (в данном случае .day)
fname=header.substring(0,8).trim().concat(".day"); //в fname имя файла для
      //хранения данных на сервере
header=header.substring(8).trim(); // "урезать" строку на 8 символов и далее
      //определить число оставшихся переданных байт для счетчика
nbytes=(new Integer(header)).intValue();
      //связать выходной поток с файлом
fo=new FileOutputStream(fname); //выходной поток ассоциируется с файлом
      //вывод тела запроса в файл
while(nbytes-->0)  //цикл с уменьшением счетчика на 1, пока его значение >0
{c=inp.read(); //читать символ из входного потока
fo.write(c);}  //писать символ в выходной поток
fo.close();    //закрыть выходной поток, ассоциированный с файлом
byte d[]=new byte[2]; //зарезервировать массив длиной 2 байта
(new String("OK\n")).getBytes(0,2,d,0); //записать в массив OK
outp.write(d);}//вывод в выходной поток сервера формального ответа "OK"
else   //если команда SEND не прочитана
{System.out.println("Polucheno RESEIVE"); // считается, что
       //получено RESEIVE, т.е. клиент запрашивает у сервера
  //содержимое одного из файлов.
header=header.substring(new String("RECEIVE").length()).trim();
        // "урезаем" файл на длину слова RECEIVE
        //выделяем имя запрашиваемого файла(8 символов)
        // и добавляем расширение .day
fname=header.substring(0,8).trim().concat(".day");
if ((new File(fname)).exists())  //существует ли файл?
{     //файл существует
fi=new FileInputStream(fname); //ассоциируем файл со входным потоком
String tmpStr="OK"+fi.available() +"\n"; //определяем строку длиной ОК + число
         //байт в файле + символ конца строки, т.е. описание данных
byte d[]=new byte[tmpStr.length()];//резервируем массив байт,
                //соответствующей длины для описания
tmpStr.getBytes(0, tmpStr.length(),d,0);//выбираем символы описания в
                        //байтовый массив
outp.write(d);    //записываем во входной поток
while(fi.available()>0)
```

```
{
c=fi.read();// посимвольно читаем
outp.write(c);  // и пишем содержимое файла в выходной поток

}
}
else     //если
{     //файл отсутствует
byte d[]=new byte[4];  //резервируем массив байт
(new String("OK 0\n")).getBytes(0,4,d,0); //формируем формальное сообщение о
                                          //нуле  пересылаемых байт
outp.write(d);          //выводим сообщение в выходной поток
}}
outp.close();  //закрываем выходной, входной потоки и гнездо
inp.close();
reg.close();
}     //hhtpServer.close
      //ServerSoket  не удаляется и работает постоянно
}
catch(Exception e)
{System.out.println(e.toString());  //здесь должна быть предусмотрена
      //обработка исключительных ситуаций
}}}
```

В приложении создается гнездо сокета, и сервер ожидает обращения (метод accept()) через нестандартный порт 1234.

Текст возможной реализации TCP-соединения для клиентской части.

```
import Java.net.*;
import Java.io.*;
class MyTelnet
{public static void main(String argv[])
{Socket sock;
InputStream inp;
OutputStream outp;
String url;
int port=1234; //номер порта "по умолчанию" соответствует номеру, который
               //прослушивается сервером
int c=0;
if(argv.length==0)    //в командной строке не заданы параметры
{System.out.println("Parametri otsutstvyut"); //тогда сообщение
return;       //завершение работы клиентского приложения
}
if(argv.length>1)      //заданы 2 параметра
{
port=(Integer.valueOf(argv[1],10)).intValue();  //второй параметр - номер порта
}
try
{sock=new Socket(argv[0],port);  //Организация гнезда на клиенте
                                 //(первый параметр IP-адрес сервера)
inp=sock.getInputStream();    //Получить объекты потоков:  входного
outp=sock.getOutputStream();    //и выходного
FileInputStream        frequest=new
```

```
FileInputStream("toserv.reg");   //ассоциировать со входным потоком
                //файл toserv.reg из текущего каталога
while((c=frequest.read()) !=-1)// читать очередной символ из файла
{ outp.write(c);  //и писатьего в выходной поток, пока не конец файла

System.out.println("simvol v potoc"); //вывод сообщения (это отладочный
}         //оператор), его следует убрать из программы
frequest.close();         //закрыть входной поток
        //создать файл fromserv.rpl и ассоциировать его с выходным потоком
        freply
FileOutputStream  freply=newFileOutputStream("fromserv.rpl");
while((c=inp.read())!=-1)    //читать символ  ответа сервера из
{freply.write(c);        // входного потока и  писать в файл
System.out.println("vivod iz potoca");  //отладочная печать, выбросить
}
                //закрыть все потоки и гнездо
freply.close();
inp.close();
outp.close();
sock.close();
}
catch(UnknownHostException ex)  //если указанный Host "ex" не найден
{
System.out.println(ex.toString());
System.out.println("neizvestnii host");
}
catch(IOException ex)       //обработка исключительных ситуаций
{System.out.println(ex.toString());// ввода-вывода
System.out.println("oshibcaIO");
}
}}
```

Запуск серверного приложения осуществляется

Работа с приложением:

Откомпилировать серверный и клиентский модули командами:

Java OrganizerServer.Java

Java MyTelnet.Java

Создаются файлы классов: UDPServer.class, и UDPClient.class

Запустить на выполнение следующим образом:

Запуск серверного приложения осуществляется командой

Java OrganizerServer

Запуск клиентского приложения – командой

Java MyTelnet 127.0.0.01 1234

либо например:

Java MyTelnet pc252 1234, где параметр строки запуска (127.0.0.01)- IP-адрес компьютера-сервера (pc252 – доменный адрес компьютера) . В данном случае адрес 127.0.0.01 является собственным адресом (петля через Интернет), именно этот адрес используется в том случае, если и сервер и клиент находятся на одном и том же компьютере. В качестве этого параметра можно использовать либо IP-адрес компьютера, либо его имя в домене. Порт 1234 –

«зашит» в программе сервера и вводится в команде запуска для программы клиента.

Для работы примера необходимо создать файл с именем “toserv.reg” в текущем каталоге, в качестве содержимого которого задается команда, определяющая тип взаимодействия клиента и сервера, например: RECEIVE 12122001, где RESEIVE – требование переслать с сервера, а 12122001 – определяет имя файла на сервере с расширением .day (т.е. файла 12122001.day, размещаемого в текущем каталоге сервера , где находится текст пересылаемой информации). В режиме SEND команда и пересылаемый текст располагаются в одном файле toserv.reg, например: SEND 12122001 привет из Минска, наконец работает!

Здесь:

SEND –команда;

12122001 – имя файла, который дополнится расширением .day и будет создан в текущем каталоге сервера для хранения пересылаемой последующей информациией. Все команды и исходные файлы с текстами должны заканчиваться символом «конец строки/возврат каретки», т.е. в конце текста файла необходимо нажать клавишу «Enter».

ПРИЛОЖЕНИЕ 2. Пример программы, демонстрирующей использование технологий проектирования динамических Web-приложений.

В качестве простого примера, демонстрирущего возможности доступа к базе данных, расположенной на сервере, при помощи технологий JSP и сервлетов Java, представлено Web-приложение для учета книг в библиотеке.

Приложение состоит из одной страницы JSP и одного сервлета, исходный код которых следует ниже.

Страница index.jsp отображает информацию о книгах из таблицы book базы данных.

При нажатии на одну из кнопок, имеющихся на странице, происходит обращение к функциям сервлета LabaServlet, которые позволяют добавлять, удалять и редактировать соответствующие записи в базе данных.

Листинг 1. index.jsp

```
<%@ page contentType="text/html; charset=windows-1251" import="Java.sql.*"%>
<html>

<head>
<title>Лабораторная работа №4</title>
<script language="JavaScript" type="text/Javascript">
var sel
function onRowMove() {
if (event.srcElement.parentElement.id!=sel)
      event.srcElement.parentElement.style.setAttribute("background","#ccffff");
 }
 function onRowOut() {
   if (event.srcElement.parentElement.id!=sel)
      event.srcElement.parentElement.style.setAttribute("background", "white");
 }
```

```
        function onRowClick() {
          document.all(sel).style.setAttribute("background", "white");
          sel = event.srcElement.parentElement.id;
          document.all(sel).style.setAttribute("background", "Aqua");
        document.all.edtAuthor.value=event.srcElement.parentElement.children.tags('TD').item(0).title;
  document.all.edtName.value=event.srcElement.parentElement.children.tags('TD').item(1).title;
  document.all.edtYear.value=event.srcElement.parentElement.children.tags('TD').item(2).title;
  document.all.edtStyle.value=event.srcElement.parentElement.children.tags('TD').item(3).title;
        document.all.book_id.value=event.srcElement.parentElement.id;
        }
        function onButtonClick() {
          if (event.srcElement.name=="btnUpdate") document.all.cmd.value="update";
          if (event.srcElement.name=="btnInsert") document.all.cmd.value="insert";
          if (event.srcElement.name=="btnDelete") document.all.cmd.value="delete";
          document.all.mainForm.submit();
        }
        </script>
        </head>

        <body bgcolor="#006699" align="center">
        <div align="center">
        <font color="white" size="+3"><b>Лабораторная работа №4</b></font><br><br>
        <% Connection conn;
         try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          conn = DriverManager.getConnection("jdbc:odbc:BOOKS", "dba", "sql");
        %>
        <table border="1" cellpadding="3" cellspacing="0" width="500" align="center">
         <thead bgcolor="silver">
          <th width="100">Автор</th>
          <th width="150">Название</th>
          <th width="100">Год издания</th>
          <th width="*">Жанр</th>
         </thead>
        </table>
        <div style="width:500; height:191; overflow-y:scroll" align="center">
        <table border="1" cellpadding="3" cellspacing="0" width="484" id="mainTable" align="center">
         <colgroup align="center">
          <col width="100">
          <col width="150">
          <col width="100">
          <col width="*">
         </colgroup>
        <%      Statement stmt = conn.createStatement();
              ResultSet rs = stmt.executeQuery ("select * from book");
            String author, name, year, style, book_id;
```

```
            int i=0;
            while (rs.next()) {
              i++;
              book_id = rs.getString("book_id");
              author = rs.getString("author");
              name = rs.getString("name");
              year = rs.getString("year");
              style = rs.getString("style");
        %>
          <TR bgcolor="White" id='<%=book_id%>' onclick="onRowClick()"
onmousemove="onRowMove()" onmouseout="onRowOut()">
            <TD title='<%=author%>'><%=author%></TD>
            <TD title='<%=name%>'><%=name%></TD>
            <TD title='<%=year%>'><%=year%></TD>
            <TD title='<%=style%>'><%=style%></TD>
          </TR>
        <%      if (i==1) {
        %>
                <script language="JavaScript" type="text/Javascript">
                  sel = '<%=book_id%>';
                </script>
        <%      }
           }
        %>
        </table>
        </div>

        <br>
        <form name="mainForm" action="LabaServlet" method="get">
        <input type="hidden" name="cmd">
        <input type="hidden" name="book_id">
          <input type="input" name="edtAuthor" style="width:108">
          <input type="input" name="edtName" style="width:152">
          <input type="input" name="edtYear" style="width:108">
          <input type="input" name="edtStyle" style="width:120">
        <br><br>
          <input type="button" value="Добавить" name="btnInsert" style="width:80"
onclick="onButtonClick()">
        <%    if (i!=0) {
        %>
          <input type="button" value="Изменить" name="btnUpdate" style="width:80"
onclick="onButtonClick()">
          <input type="button" value="Удалить" name="btnDelete" style="width:80"
onclick="onButtonClick()">
            <script language="JavaScript" type="text/Javascript">
              document.all(sel).style.setAttribute("background", "Aqua");
          document.all.edtAuthor.value=document.all(sel).children.tags('TD').item(0).title;
  document.all.edtName.value=document.all(sel).children.tags('TD').item(1).title;
  document.all.edtYear.value=document.all(sel).children.tags('TD').item(2).title;
  document.all.edtStyle.value=document.all(sel).children.tags('TD').item(3).title;
              document.all.book_id.value=document.all(sel).id;
            </script>
```

```
<%      }
%>
</form>
<%
}
  catch(Exception exc) {
    exc.printStackTrace();
%>
<br><br><i>Не удалось установить соединение с базой данных.</i>
<% }
%>
</div>
</body>
</html>
```

Листинг 2. JavaServlet.Java

```
import Javax.servlet.*;
import Javax.servlet.http.*;
import Java.sql.*;
import Java.io.*;

public class LabaServlet extends HttpServlet implements SingleThreadModel {
  public void service(HttpServletRequest request, HttpServletResponse response) {
    String cmd = (String)request.getParameter("cmd");
      String book_id = (String)request.getParameter("book_id");
        String author = (String)request.getParameter("edtAuthor");
        String name = (String)request.getParameter("edtName");
        String year = (String)request.getParameter("edtYear");
      String style = (String)request.getParameter("edtStyle");
    if (cmd!=null)
    try {
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection  conn = DriverManager.getConnection("jdbc:odbc:BOOKS",  "dba",
"sql");
        Statement stmt = conn.createStatement();

        if (cmd.equalsIgnoreCase("update")) {
        int res = stmt.executeUpdate ("update book set author='"+author+"',
name='"+name+"', year="+year+", style='"+style+"' where book_id="+book_id);
        }

        if (cmd.equalsIgnoreCase("insert")) {
        int res = stmt.executeUpdate ("insert into book values
(null,'"+author+"','"+name+"',"+year+",'"+style+"'"+")");
        }

        if (cmd.equalsIgnoreCase("delete")) {
        int res = stmt.executeUpdate ("delete from book where book_id="+book_id);
        }
    }
    catch(Exception exc) {
      exc.printStackTrace();
    }
```

```
            finally {
        try                                                                    {
getServletConfig().getServletContext().getRequestDispatcher("/index.jsp").forward(request
, response); }
                catch(Exception exc) { exc.printStackTrace(); }
              }
            }

            public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {
                service(request, response);
            }

            public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {
                service(request, response);
            }
```

Св. план 2002, поз. 109

Учебное издание

Лепеш Анастасия Владимировна,
Онищук Вениамин Викентьевич,
Комличенко Виталий Николаевич,
Пасовец Александр Александрович

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по курсу
«Разработка информационных систем в WWW»
для студентов экономических специальностей БГУИР
В 2-х частях
Часть 1

Редактор Е. Н. Батурчик

Подписано в печать 30.09.2002.			Формат 60x84 1/16.
Бумага офсетная.	Печать ризографическая	Гарнитура Таймс	Усл.-печ. л. 3,49
Уч.-изд. л. 2,7	Тираж 150 экз.		Заказ

Издатель и полиграфическое исполнение:
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия ЛП № 156 от 05.02.2001.
Лицензия ЛВ № 509 от 03.08.2001.
220013, Минск, П. Бровки,6.