

## 10.1. Краткая справка по языку HTML

World Wide Web, или как ее обычно называют, *WWW* – это распределенная компьютерная система, основанная на гипертексте. Информация в ней хранится на компьютерах с соответствующим программным обеспечением (серверах), объединенных в глобальную сеть. Она включает в себя не только текст, но и возможность выполнения определенных действий при выборе специально отмеченных участков текста (так называемый *гипертекст*), а также графику, видео, звук (т. н. средства *мультимедиа*). Эта информация содержится в виде *HTML*-документов, которые могут содержать ссылки на другие документы, хранящиеся как на том же самом, так и на другом сервере. На экране компьютера гиперссылки выглядят как выделенные другим цветом и/или подчеркиванием участки текста или рисунки (графические изображения).

Используя гиперссылки, называемые также гипертекстовыми связями, *пользователь* может автоматически связаться с соответствующим источником информации в сети и получить на экране своего компьютера документ, на который была сделана ссылка. В большинстве случаев выбор гиперсвязей производится при помощи щелчка клавишей мыши на участке текста с гиперсвязью. При этом компьютер посылает через сеть по протоколу *http* запрос серверу, хранящему файл с необходимым документом. Сервер, получив запрос, посылает клиенту этот файл или сообщение об отказе, если требуемый документ по тем или иным причинам недоступен.

Просмотр *HTML*-документов осуществляется с помощью браузеров – программ просмотра *WWW*-документов (*WWW*-browsers). В настоящее время получили распространение десятки таких программ, но наиболее известными и развитыми являются *Microsoft Internet Explorer*, *Mozilla* (в том числе один из его клонов, *Fire Fox*), *Opera*, а также уже сошедший со сцены *Netscape Navigator*.

*WWW*-документ, как уже отмечалось, содержит форматированный текст, графику и гиперсвязи с использованием различных ресурсов *Internet*. Чтобы реализовать все эти возможности, был разработан специальный компьютерный язык – **HyperText Markup Language (HTML)** – язык разметки гипертекста. Гипертекст ("сверхтекст") – это текст, содержащий дополнительные возможности, в частности – гиперссылки. Существует несколько версий языка *HTML*. Самая современная на данный момент версия – *HTML 4.01*, принятая в виде рекомендации консорциума *W3C (World Wide Web Consortium)*, отвечающего за развитие языка *HTML* и других *WWW*-технологий. XML-версия языка *HTML*, называемая *XHTML*, пока не нашла широкого распространения. Наиболее употребляемая при создании простых *WWW*-документов версия – *HTML 3.2*.

Существует большое количество сред, позволяющих интерактивно создавать *HTML*-документы. Тем не менее, даже в этом случае полезно знать основные принципы устройства *HTML*-документов и имеющиеся в этом языке средства разметки.

Документ, написанный на языке *HTML*, представляет собой текстовый файл, содержащий собственно текст, несущий информацию пользователю, и теги разметки (*markup tags*). Теги представляют собой определенные последовательности символов, заключенные между знаками '<' и '>'.

Программа просмотра располагает текст на экране дисплея согласно задаваемой тегами разметке, а также включает в него рисунки, хранящиеся в отдельных графических файлах, и формирует гиперсвязи с другими документами или ресурсами *Internet*. После тега через пробел, вплоть до закрывающего символа '>', может следовать один или несколько параметров.

Файл на языке *HTML* имеет расширения *.html* или *.htm*. Он приобретает облик *WWW*-документа только тогда, когда просматривается в специальной программе просмотра – браузере.

Текст в *HTML* может включать любую последовательность символов, за исключением следующих:

<  
>  
&  
"

Вместо них должны присутствовать комбинации

&lt;  
&gt;  
&amp;  
&quot;

Символы табуляции и перехода на новую строку считаются эквивалентными пробелу, а несколько этих символов и пробелов подряд (в любой комбинации) одному пробелу. Для вставки в текст значимого пробела

используется комбинация

&nbsp;

**Теги** предназначены для форматирования и разметки документа. Теги бывают парные ("контейнеры") и непарные. Действие парного тега начинается с открывающего тега и заканчивается при встрече соответствующего ему закрывающего, признаком которого является символ " / ". Например:

```
<html>Это html документ </html>
```

Непарный тег вызывает единичное действие в том месте, где он встречается. Например, тег `<br>` вызывает перевод текста на новую строку.

Исключением из правила, гласящего об эквивалентности любого числа пробелов, табуляций и переходов на новую строку одному пробелу, является текст внутри контейнера `<pre> </pre>`. Этот текст показывается так же, как он был предварительно отформатирован в обычном текстовом редакторе с использованием моношириного шрифта, и все пробелы и переносы на новую строку являются значимыми. Однако внутри данного контейнера могут действовать другие теги разметки.

Внутри тега кроме его имени могут находиться атрибуты, задающие дополнительные параметры, необходимые для действия тега.

Например, тег `` обеспечивает показ в данном месте текста изображения из файла с именем `MyFile.gif`, а ширина и высота изображения задается 100 на 40 точек (пикселей), соответственно. При этом атрибут `src`, задающий имя файла, является обязательным, а `width` и `height` – необязательные (могут отсутствовать).

Типичный HTML-документ имеет заголовок и тело. Начало документа отмечается тегом `<html>` и заканчивается тегом `</html>`

**Заголовок документа:** `<head>Текст, включающий служебную информацию о документе</head>` Он никак не отображается на экране компьютера при просмотре HTML- файла, за исключением названия документа, заключенного в контейнере `<title>Текст заголовка</title>`, помещаемого между тегами заголовка. Это название обычно выводится как заголовок оконной формы, в которой происходит показ файла.

Важнейшей служебной информацией является кодировка документа, задаваемая следующим образом:

```
<meta http-equiv="content-type" content="text/html; charset=Windows-1251">
```

– русскоязычная кодировка ISO 1251.

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
```

– кодировка UTF-8, и т.д.

**Тело документа** определяет видимую часть документа:

```
<body>
  Это html-документ, содержащий какой-то текст.
</body>
```

На дисплее: Это html-документ, содержащий какой-то текст.

Часть текста или участок изображения в HTML-документах может ссылаться на другой текст внутри того же самого документа, или на другой документ в Internet. Такая связь называется гипертекстовой связью (*hypertext link*), гиперсвязью, гипертекстовой ссылкой или гиперссылкой. Она выделяется подчеркиванием и цветом.

При ссылке на документ, находящийся на другом сервере, необходимо указать адрес (URL – 'Uniform Resources Location') этого документа: сетевой адрес сервера и путь к этому документу на сервере. Если документ находится на том же сервере, но в другой папке, достаточно указать только путь к этой папке.

В подавляющем большинстве случаев документ, на который делается ссылка, находится на том же сервере и в той же папке. В этом случае гипертекстовая ссылка имеет вид:

```
<a href="имя_файла">текст_ссылки</a>
```

Тут `имя_файла` – имя файла (с указанием расширения), содержащего документ, на который делается ссылка, а `текст_ссылки` – якорь, т. е. участок текста, который будет выделен как связанный с гиперсвязью. В общем случае перед именем файла ставится URL-адрес сервера и полный путь к файлу на сервере.

Язык HTML позволяет ссылаться не только на документы целиком, но и на отдельные части конкретного документа. В этом случае та часть документа, на который делается ссылка, называется меткой. То место, куда осуществляется переход называется меткой. Якорь задается в виде

```
<a href="имя_файла#имя_метки">текст</a> ,
```

а *метка* –

```
<a name="имя_метки">участок документа</a>
```

Тут *имя\_метки* – произвольное имя метки, на которую должен быть переход (уникальное для данного документа), а *имя\_файла* – имя файла ( вместе с расширением и путем), содержащего документ, на какое-либо место которого осуществляется ссылка. Надо отметить, что переход производится в начало параграфа, в котором расположена мишень. Поэтому делению текста на параграфы при наличии гиперсвязей внутри документа надо уделять особое внимание.

**Замечание:** Программы просмотра текста имеют специальное средство, уменьшающее загрузку компьютерных сетей: буферизацию принятых сообщений, размер которых не превышает некоторую границу (ее можно менять в параметрах программы просмотра). Этот *буфер* хранится некоторое время (также задаваемое в параметрах программы просмотра – обычно дни или недели), на жестком диске, на котором установлена *программа* просмотра. Те адреса, по которым вы еще "не ходили", показываются одним цветом ( по умолчанию синим), а по которым "ходили" (и документы еще хранятся в буфере) – другим цветом ( по умолчанию малиновым).

**Параграф** – осуществляет показ одной пустой строки и "переводом каретки" перед началом заключенного в контейнере текста. Внутри параграфа возможно выравнивание:

```
<p>без выравнивания текста</p>
<p align=left>выравнивание текста по левому краю</p>
<p align=right>выравнивание текста по правому краю</p>
<p align=center>выравнивание текста по центру</p>
```

Закрывающий тег `</p>` необязателен.

**Горизонтальная разделительная черта** `<hr>`

**Заголовки** – используются для выводов заголовков и подзаголовков (всего 6 уровней). Значение уровня заголовка может от 1 до 6.

```
<h1>Заголовок первого уровня</h1>
<h2>Заголовок второго уровня</h2>
<h3>Заголовок третьего уровня</h3>
<h4>Заголовок четвертого уровня</h4>
<h5>Заголовок пятого уровня</h5>
<h6>Заголовок шестого уровня</h6>
```

**Нумерованный список** – задается в виде:

```
<ol>
<li>...
<li>...
<li>...
...
</ol>
```

В *HTML*-файле:

Курсовые в срок сдали следующие студенты:

```
<ol>
<li>Иванов
<li>Петров
<li>Сидоров
</ol>
```

На дисплее:

Курсовые в срок сдали следующие студенты:

1. Иванов
2. Петров
3. Сидоров

**Ненумерованный список**

В *HTML*-файле задается в схожем с нумерованным списком виде :

Курсовые в срок сдали следующие студенты:

```
<ul>
<li>Иванов
<li>Петров
<li>Сидоров
</ul>
```

На дисплее:

Курсовые в срок сдали следующие студенты:

```
Иванов
Петров
Сидоров
```

В *HTML* существуют следующие основные стили текста:

Таблица 10.1.

<code>&lt;b&gt;Жирный текст (bold)&lt;/b&gt;</code>	<b>Жирный текст (bold)</b>
<code>&lt;i&gt;Наклонный текст (italics)&lt;/i&gt;</code>	<i>Наклонный текст (italics)</i>
<code>&lt;big&gt;Большой размер шрифта&lt;/big&gt;</code>	Большой размер шрифта
<code>&lt;small&gt;Маленький размер шрифта&lt;/small&gt;</code>	Маленький размер шрифта
Нижние индексы <code>&lt;sub&gt;(subscript)&lt;/sub&gt;</code>	Нижниеиндексы( <i>subscript</i> )
Верхние индексы <code>&lt;sup&gt;(superscript)&lt;/sup&gt;</code>	Верхниеиндексы( <i>superscript</i> )

Кроме того, существует показ предварительно отформатированного текста. Текст внутри контейнера `<pre>...</pre>` показывается моноширинным фонтом, все символы имеют одинаковую ширину, все пробелы и переходы на новую строку показываются без игнорирования.

Язык *HTML* позволяет вставлять в текст изображения, хранимые в отдельных графических файлах. Тег вывода изображения имеет следующий вид:

```

```

где **имя\_файла** – имя графического файла (с указанием расширения), содержащего изображение, **ширина** – ширина изображения, **высота** – *высота* изображения, **ширина\_рамки** – ширина рамки вокруг изображения. Все размеры задаются в пикселах (точках экрана). Если реальные размеры изображения не совпадают с заданными в атрибутах **width** и **height**, то при показе оно масштабируется до этих размеров.

Атрибут **border** не обязателен, но желателен, если с картинкой ассоциирована гиперсвязь.

Атрибуты **hspace** и **vspase** задают отступы от картинки по вертикали и горизонтали для текста или других картинок.

Рекомендуется всегда указывать ширину и высоту изображения, в противном случае *программа* просмотра будет вынуждена перед выводом изображения документа на экран загрузить как весь текстовый *файл*, так и все файлы с изображениями, что занимает много времени. Если же атрибуты **width** и **height** указаны, то текст покажется сразу, а изображения будут показываться по мере "подкачивания" по сети. Кроме того, объем текстовых файлов, как правило, намного меньше, чем у графических, и поэтому они получаются гораздо быстрее.

Для того, чтобы изображение служило гиперссылкой, достаточно поместить тег `<img>` внутри тега `<a href="адрес" >`

В *HTML*-документах можно задавать **таблицы**. Каждая *таблица* должна начинаться тегом `<table>`, а если у таблицы требуется внешняя рамка, то с параметром **border** (возможны варианты **border** или **border=ширина\_рамки**): `<table border>` и заканчиваться тегом `</table>`. Таблицы задаются построчно, каждая строка начинается тегом `<tr>` и заканчивается тегом `</tr>`. Каждая графа (т. е. "ячейка", "клетка") в строке с данными должна начинаться тегом `<td>` и заканчиваться тегом `</td>`. При этом ширина столбцов подбирается автоматически по максимальной ширине одной из клеток столбца. В

таблицы так же можно вставлять гипертекстовые ссылки, произвольным образом отформатированный текст, рисунки и т. п. Общий вид таблицы:

```
<table border>
  <tr> <td>...</td> <td>...</td> <td>...</td> <td>...</td> </tr>
  <tr> <td>...</td> <td>...</td> <td>...</td> <td>...</td> </tr>
  ...
</table>
```

Эта *таблица* содержит две строки и четыре графы (столбца). Поскольку в *HTML* перевод на новую строку равнозначен пробелу, а лишние пробелы игнорируются, текст в *HTML*-документах обычно форматируют с помощью переводов на новую строку так, чтобы не было слишком длинных строк. Поэтому приведенная выше *таблица* может быть записана так:

```
<table border>
<tr>
  <td>...</td>
  <td>...</td>
  <td>...</td>
  <td>...</td>
</tr>
<tr>
  <td>...</td>
  <td>...</td>
  <td>...</td>
  <td>...</td>
</tr>
</table>
```

Вид документа при *просмотре файла HTML*-браузером (*browser*), естественно, не изменится.

Если в таблице нужны *заголовки*, они задаются тегами `<th>...</th>`

**Окна подключаемых модулей** (*plug-in*) задаются контейнером `<object>` :

```
<object
  атрибуты
>
  <param name=имя1 value=значение1>
  <param name=имя2 value=значение2>
  ...
  <param name=имяN value=значениеN>
  Альтернативный текст, который будет виден в браузерах, не поддерживающих
  работу с объектами данного типа
</object>
```

В качестве таких объектов могут служить *апплеты Java*, мультимедийные клипы и т.п.

## 10.2. Апплеты

Напомним некоторые вещи, о которых рассказывалось в "[Общие представления о языке Java](#)". *Апплет* – это специализированная *программа Java* с ограниченными возможностями, работающая в окне *WWW*-документа под управлением браузера. Как правило, *апплеты* встраивают в *HTML*-документы (наиболее распространенный вид *WWW*-документов).

Между приложениями (*applications*) и апплетами (*applets*) *Java* имеется принципиальное различие: *приложение* запускается непосредственно с компьютера пользователя и имеет *доступ* ко всем ресурсам компьютера наравне с любыми другими программами. *Апплет* же загружается из *WWW* с постороннего сервера, причем из-за самой идеологии *WWW сайт*, с которого загружен *апплет*, в общем случае не может быть признан надежным. А вот сам *апплет* имеет возможность передавать данные на любой *сервер* в *WWW* – все зависит от алгоритма, заложенного создателем *апплета*. Поэтому для того, чтобы избежать риска утечки конфиденциальной информации с компьютера пользователя или совершения враждебных действий, у апплетов убраны многие возможности, имеющиеся у приложений.

Поддержка работы с апплетами осуществляется стандартной библиотекой классов ( `core library` ), расположенной в пакете `java.applet` для обычных апплетов, а также классом `javax.swing.JApplet` для апплетов, использующих компоненты `Swing` и/или библиотеку `Sun JFC (Java Foundation Classes)`.

Для создания обычного *апплета* требуется задать *класс*, являющийся наследником класса `java.applet.Applet`, который сам является наследником класса `java.awt.Panel`.

В классе *апплета* требуется переопределить ряд методов:

```
public class Applet1 extends java.applet.Applet{

    public void init(){
        //Инициализация перед началом работы.
        //Вызывается один раз после загрузки апплета
        //перед первым выполнением метода start()
    }

    public void start(){
        //Обеспечивает основную функциональность апплета.
        //В первый раз вызывается после загрузки апплета
        //и его инициализации методом init().
        //Затем вызывается каждый раз при заходе пользователя
        //на HTML-страницу с апплетом.
    }

    public void update(java.awt.Graphics g){
        //Форсирование перерисовки апплета с выполнением кода метода
    }

    public void paint(java.awt.Graphics g){
        //Исполняется каждый раз при перерисовке апплета.
        //Обеспечивает всю визуализацию в апплете
    }

    public String getAppletInfo(){
        return "Справочная информация об апплете";
    }

    public void stop(){
        //Приостанавливает выполнение апплета.
        //Исполняется каждый раз сразу после того, когда пользователь
        //покидает HTML-страницу с апплетом.
    }

    public void destroy(){
        //Обычно не требует переопределения.
        //Предназначен для высвобождения ресурсов, захваченных апплетами.
        //Исполняется через негарантированный промежуток времени
        //каждый раз после вызова метода stop()
        //перед разрушением объекта апплета сборщиком мусора.
    }
}
```

Кроме методов, нуждающихся в переопределении, в классе `Applet` имеется некоторое количество методов, позволяющих проверять и задавать его состояние во *время выполнения*:

`getSize()` – возвращает размер *апплета*. Ширину и высоту можно получить как `getSize().width` и `getSize().height`

`showStatus(String s)` – показ в строке статуса браузера сообщения `s`.

`AppletContext getAppletContext()` – получение апплетом информации об документе, из которого был вызван *апплет*, а также о других апплетах того же документа.

`add(Component comp)` – добавление компонента в *апплет*.

`AudioClip getAudioClip(URL url)` – получение апплетом аудиоклипа по заданному WWW-адресу `url`. Создается объект Java, ссылающийся на данный аудиоклип.

`URL getDocumentBase()` – получение апплетом адреса WWW-документа, из которого был вызван *апплет*.

Имеется большое количество других методов для работы с апплетами, большинство которых унаследовано от класса `Panel`.

Ряд примеров апплетов с исходными кодами приведен в *JDK* в папке **demo/applets**.

Пример *апплета*:

```
import java.awt.*;

public class Applet1 extends java.applet.Applet{

    public void paint(java.awt.Graphics g){
        g.setColor(Color.green);
        g.fillRect(0,0,getSize().width - 1, getSize().height - 1);
        g.setColor(Color.black);
        g.drawString("Вас приветствует апплет!",20,20);
        this.showStatus("Это пример апплета");
    }
}
```

Пример *HTML*-документа, в который встроен *апплет*:

```
<html>
<body>
Это пример апплета<br>

<object
  codebase="."
  code="Applet1.class"
  width=200
  height=150
>
```

*Альтернативный текст*, который будет виден в браузерах, не поддерживающих работу с апплетами

```
</object>
```

Если данный *HTML*-документ имеет имя `example.html`, то для запуска *апплета* следует расположить файл `Applet1.class` в той же папке, что и `example.html`. После чего открыть в браузере файл `example.html`. Обычно проще всего это сделать двойным щелчком мыши по имени файла в окне навигации по файлам и папкам.

Если в открывшемся окне браузера *апплет* не будет показан, его можно просмотреть в программе `appletviewer`. Для этого надо перейти в папку с файлами `example.html` и `Applet1.class`, и запустить `appletviewer` с параметром `example.html`. Например, для *Windows®* в командной строке надо набрать

```
appletviewer.exe example.html
```

**Замечание:** на домашнем компьютере с *Windows XP SP1* и браузером *MS Internet Explorer 6.0* автору не удалось запустить ни одного *апплета*, в том числе из примеров *JDK*. Хотя *поддержка Java* была включена, и *автор* пытался менять самые разные установки системы. Но в `appletviewer` все работало. Это является хорошей иллюстрацией того, почему крупные фирмы предпочитают при работе в сетях использовать не *апплеты*, а заниматься обработкой со стороны сервера и отсылать результаты на компьютер клиента в готовом виде. Ведь у клиентов, работающих в *WWW*, могут быть компьютеры с различными версиями *JDK*, операционными системами и браузерами. И гарантировать работоспособность одного и того же *апплета* в таком разнообразном окружении практически невозможно. Если же обработка идет со стороны сервера, все упрощается. Со стороны клиента нужен только *браузер*, работающий со ставшими совершенно стандартными



языками *HTML* и *JavaScript* (после версии *HTML* 4.01 и *JavaScript* 1.3 они перестали изменяться). Браузер передает *запрос* с клиентского компьютера на *сервер*, и там формируется новый *HTML*-документ для клиентского компьютера, при необходимости – со сформированными на сервере графическими файлами.

Такая идеология позволяет при необходимости усовершенствовать систему незаметно для пользователей и с сохранением полной совместимости. Например, новая версия *JVM* устанавливается на сервере, и проводятся все необходимые настройки и тестирования. После чего вместо старой версии системы наружу показывается новая.

### 10.3. Сервлеты

Напомним что *сервлеты* – это приложения *Java*, запускаемые со стороны сервера. Они имеют возможности доступа к файловой системе и другим ресурсам сервера через набор *управляющих конструкций*, предопределенных в рамках пакета *javax.servlet* и технологии *JSP*. Технология *JSP* заключается в наличии дополнительных конструкций в *HTML*- или *XML*-документах, которые позволяют осуществлять вызовы сценариев ("скриптов"), написанных на языке *Java*. В результате удастся очень просто и удобно осуществлять обработку данных или элементов документа, и внедрять в нужные места документа результаты обработки. Сценарии *Java* перед первым выполнением автоматически компилируются на стороне сервера, поэтому выполняемый код выполняется достаточно быстро. Но, конечно, это требует, чтобы была установлена соответствующая *Java*-машина. Для дальнейшей работы требуется, чтобы на компьютере кроме *JDK* был установлен *NetBeans Enterprise Pack* и входящие в состав дистрибутива пакет *j2EE*, а также *Bundled Tomcat Server*.

Рассмотрим пример приложения, работающего с использованием *сервлетов*. Исходный код *сервлета*, выдающего на клиентском компьютере сообщение "Hello!":

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet {

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

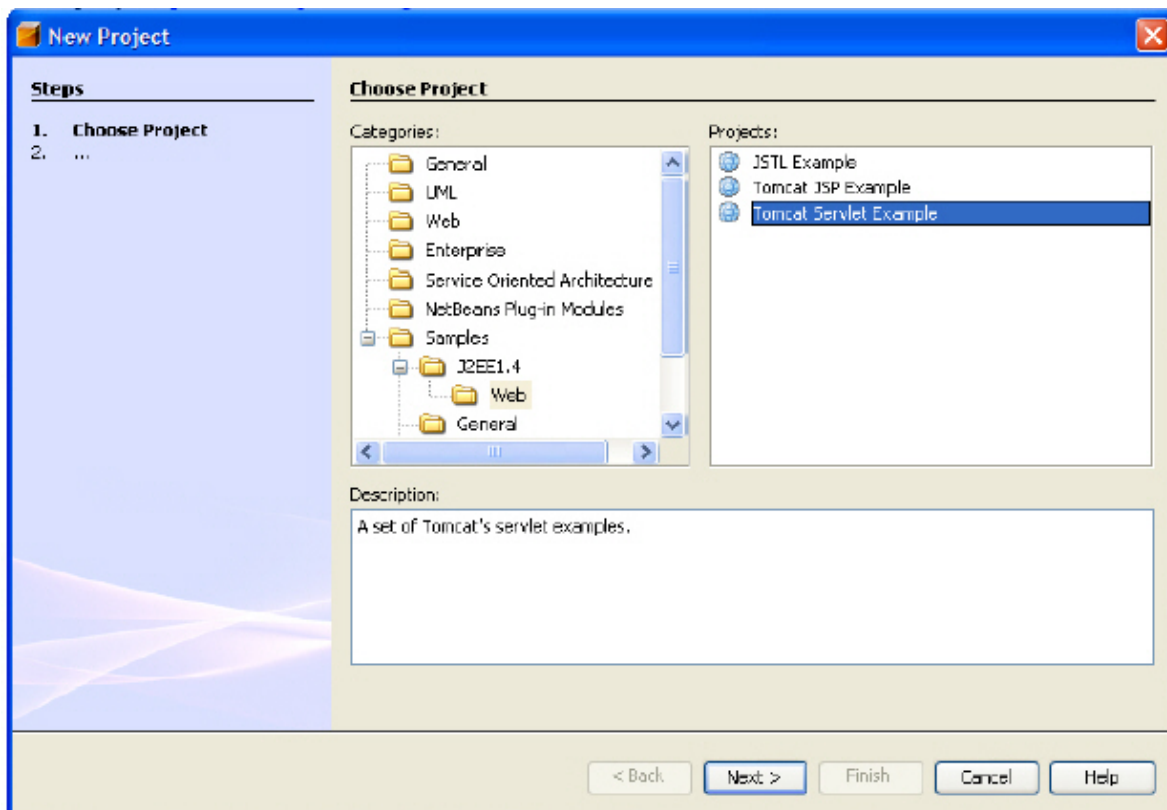
Пример файла, из которого вызывается данный *сервлет*:

```
<html>
<head>
  <title>Servlet example</title>
</head>
<body bgcolor="#FFFFFF">
  <a href="servlet/Hello">Execute servlet</a>
</body>
</html>
```

Работающие варианты *сервлетов* и их исходные коды можно посмотреть, открыв примеры: **File/New Project.../Samples/J2EE1.4/Web /Tomcat Servlet Example**

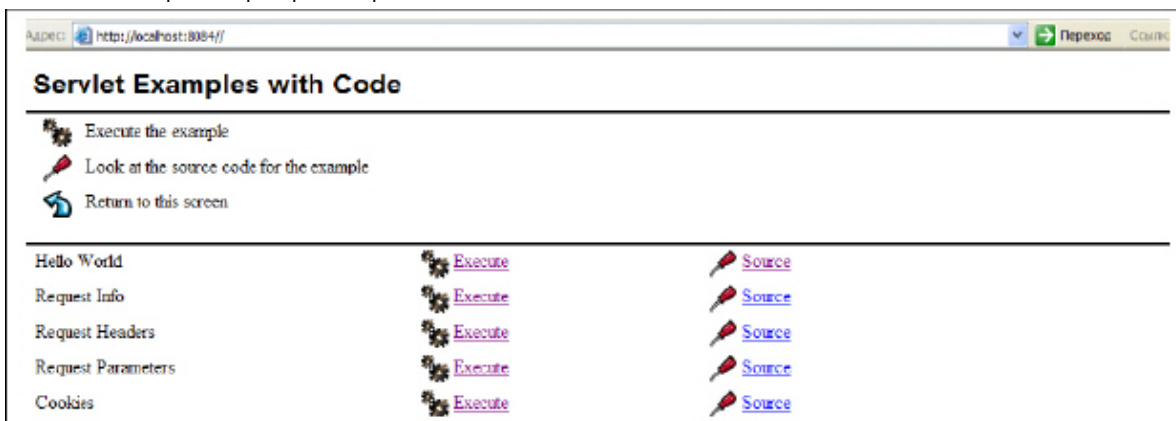


Пример разработан организацией The Apache Software Foundation (<http://www.apache.org/>), и его использование должно соответствовать лицензии, выложенной по адресу <http://www.apache.org/licenses/LICENSE-2.0>.



[увеличить изображение](#)

Рис. 10.1. Открытие примера с сервлетами

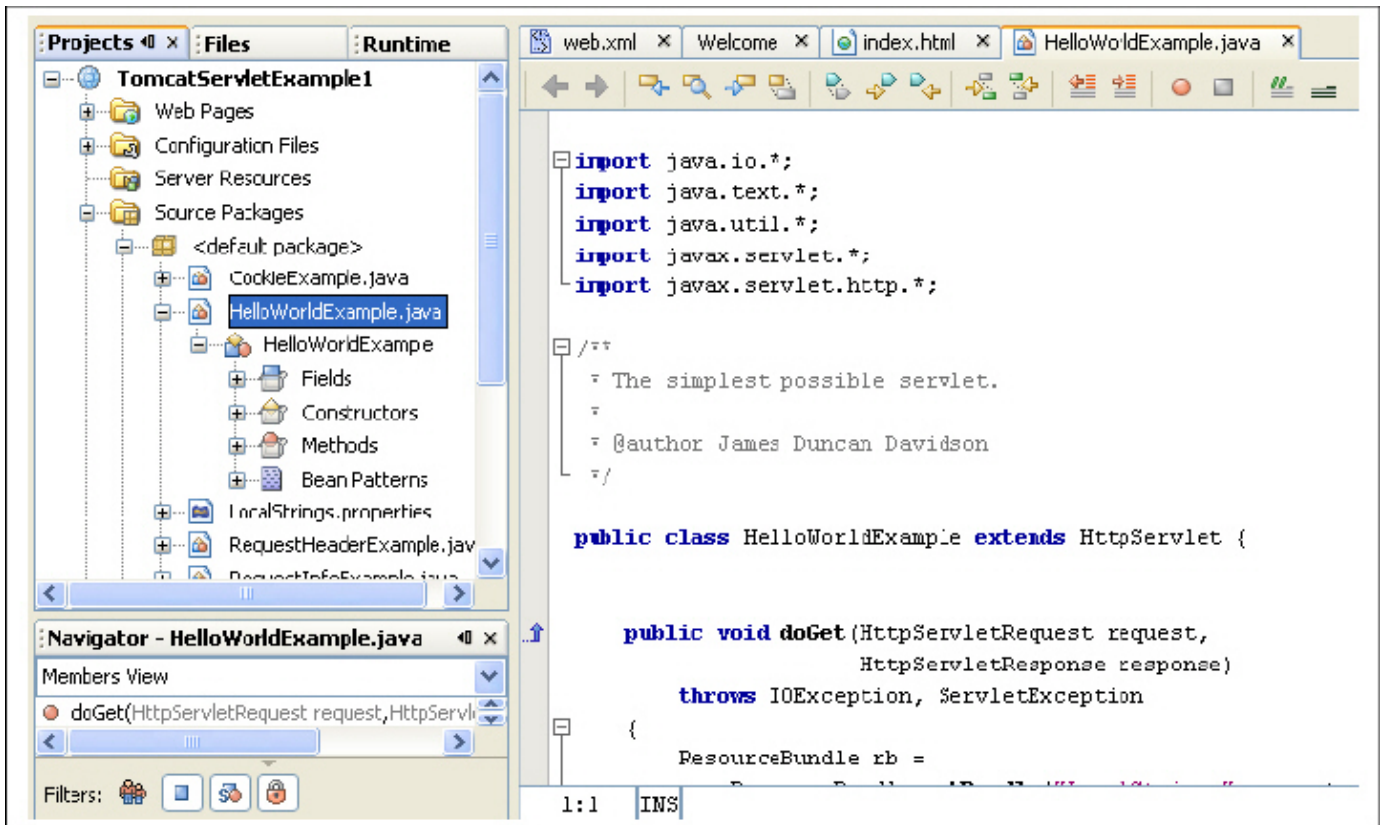


[увеличить изображение](#)

Рис. 10.2. Элементы клиентского экрана запущенного приложения с сервлетами

При нажатии на гиперссылку **Execute** ("Выполнить") соответствующий *сервлет* выполняется, и на экране показывается сформированный им *HTML*-документ. Первым идет пример Hello World. Примерное содержимое исходного кода *сервлета* можно увидеть, перейдя по гиперссылке **Source** в *HTML*-документ с изображением исходного кода. Но это не настоящий код, а лишь *HTML*-документ!— Его исходный код можно увидеть, сделав *двойной щелчок* в окне **Projects...** по узлу **Web Pages/helloworld.html**. А вот настоящий исходный код можно увидеть, сделав *двойной щелчок* в том же окне по узлу **Source Packages/<default package>/HelloWorldExample.java**

То же относится к другим примерам.



[увеличить изображение](#)

Рис. 10.3. Исходный код примера Hello World

## 10.4. Технология JSP – Java Server Pages

Одним из важнейших видов серверных программ являются приложения, использующие технологию *JSP* – *Java Server Pages*. Несмотря на то, что она опирается на использование *сервлетов*, *JSP* является самостоятельной технологией. Идеология *JSP* основана на внедрение в *HTML*-документы специальных конструкций, позволяющих со стороны сервера выполнять программную обработку данных и выводить в соответствующее место документа результат.

В качестве примеров в *NetBeans Enterprise Pack* приведены примеры *Tomcat JSP Example* и *JSTL Example*. Отметим, что *Tomcat* – название программного сервера, варианта сервера *apache*, который автоматически конфигурируется и запускается при выполнении примеров, а *example* означает "пример". Большим достоинством среды *NetBeans* является то, что оригиналы всех примеров, открываемых в *NetBeans*, остаются в неприкосновенности – автоматически создаются копии примеров. Поэтому даже если вы внесете исправления в исходный код проекта с примером, это не повлечет изменений в новом проекте с таким же примером.

Для правильной работы серверных примеров требуется, чтобы на компьютере была установлена работа с *Интернет*. Реально выходить в *Интернет* не надо, идет соединение <http://localhost:8084/>. Но после запуска другого серверного приложения идет соединение по тому же адресу, поэтому документ берется из буфера – и показывается документ, созданный предыдущим приложением. Для показа правильного документа требуется нажать в браузере кнопку "обновить", и в случае автономной работы в появившемся диалоге, предлагающем выбрать режим работы, выбрать "Подключиться". Реального соединения с *Интернет* для адреса <http://localhost:8084/> не будет – все коммуникации станут проходить локально.

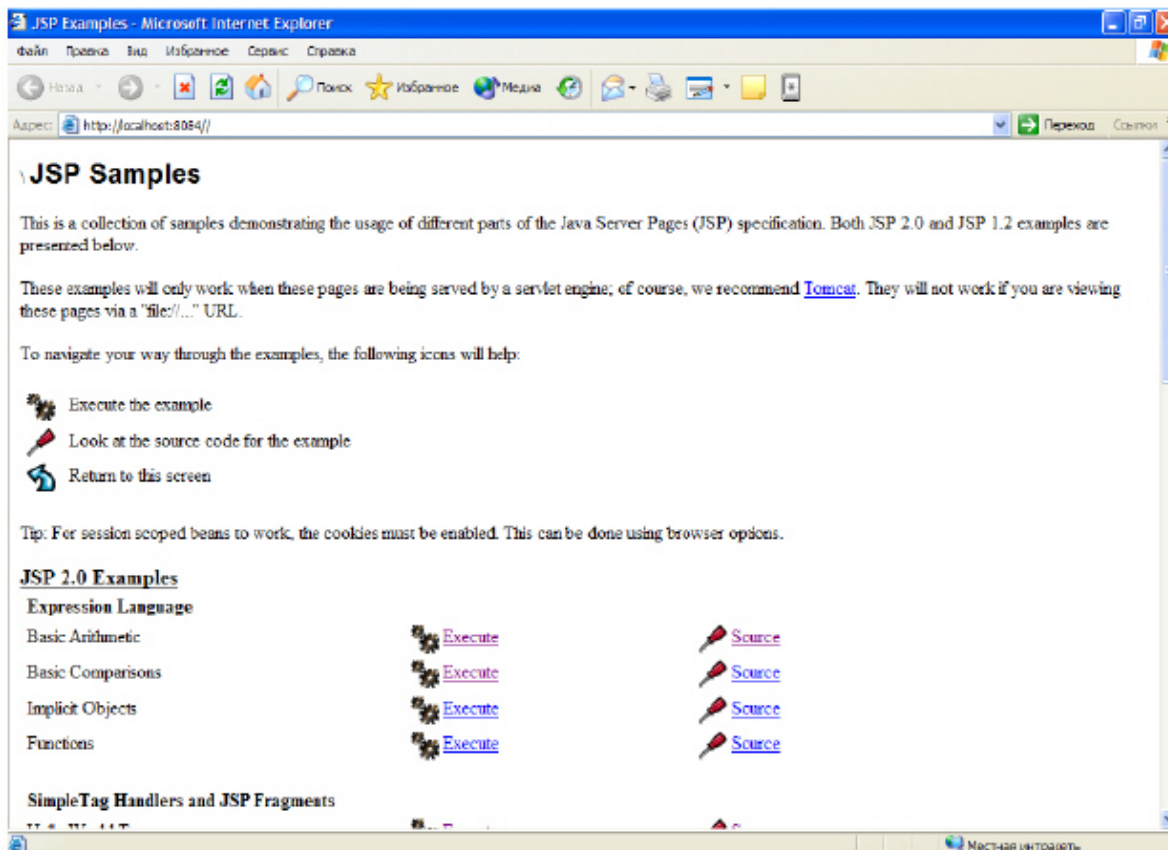
Первый из примеров иллюстрирует базовые конструкции *JSP*, его можно просмотреть, создав проект **File/New Project.../ Samples/J2EE1.4/Web /Tomcat JSP Example**.

Второй пример – надстройка над *JSP*, специальный набор тегов *JSP*, разработанный группой экспертов для облегчения разработки серверных приложений. Пример можно просмотреть, создав проект **File/New Project.../ Samples/J2EE1.4/Web / JSTL Example**.

Оба этих примера также разработаны организацией *The Apache Software Foundation* (<http://www.apache.org/>), и их использование также должно соответствовать лицензии <http://www.apache.org/licenses/LICENSE-2.0>.

Порядок примеров в мастере создания приложений прямо противоположный рассматриваемому нами – сначала предлагается использовать *JSTL* как наиболее развитое средство, затем – *JSP* как средство более низкого уровня, и только затем *Servlet* – как средство еще более низкого уровня. Мы используем при рассмотрении обратный порядок, так как *JSP* использует *сервлеты*, а *JSTL* является надстройкой над *JSP*.

Рассмотрим подробнее первый пример.



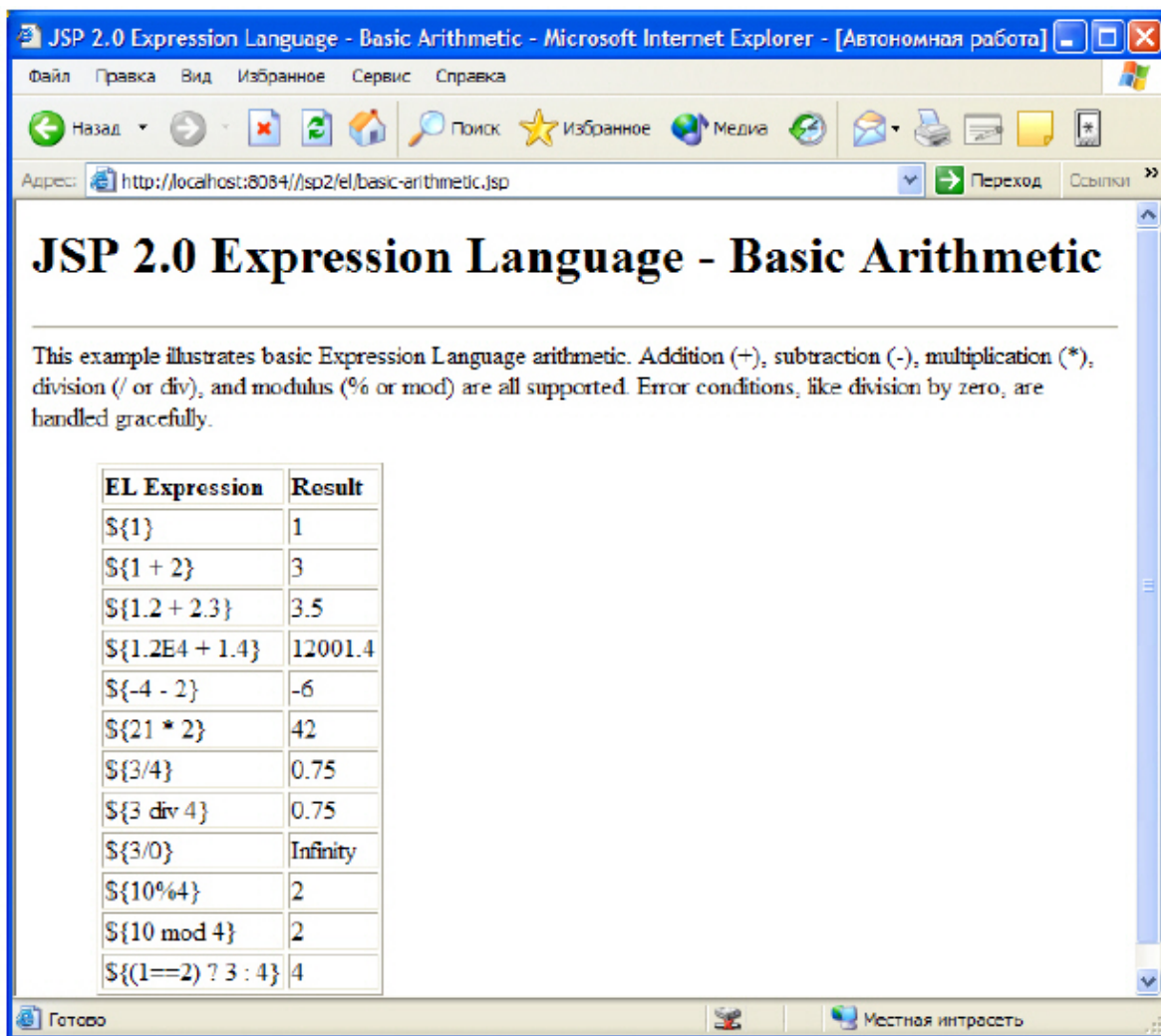
увеличить изображение

Рис. 10.4. Первая страница запущенного примера JSP

Как и в предыдущем случае, при нажатии на гиперссылку **"Execute"** выполняется соответствующий пример – в данном случае запускается страница *JSP*. А при нажатии гиперссылки **Source** показывается *HTML*-страница с примерным видом исходного кода.

Страницы *JSP* представляют обычные *HTML*-документы, но имеющие расширение имени файла *.jsp*, а не *.html* или *.htm*. Это – заготовка *HTML*-документа, который будет показан пользователю-клиенту. При создании клиентского *HTML*-документа в этой заготовке выражения в фигурных скобках после знака доллара вычисляются, а в *HTML*-документ подставляется строковое представление результата.

Например, выражение вида  `$\{1 + 2\}$`  выдаст в соответствующее место документа символ 3. Последовательность `\$` означает, что выражение  `$\{...\}$`  не вычисляется, а рассматривается как строка.



[увеличить изображение](#)

**Рис. 10.5.** Первый пример JSP- вычисление выражений

Имеется ряд встроенных в JSP тегов (объектов Java):

`request` – запрос. Тип объекта – класс, реализующий интерфейс

`javax.servlet.http.HttpServletRequest` .

`response` – ответ на запрос. Тип объекта – класс, реализующий интерфейс

`javax.servlet.http.HttpServletResponse` .

`pageContext` – контекст страницы JSP. Обеспечивает доступ к пространству имен и элементам страницы (тегам, атрибутам). Тип объекта – класс `javax.servlet.jsp.PageContext` .

`session` – сессия (сеанс связи клиента с сервером). Тип объекта – класс, реализующий интерфейс

`javax.servlet.http.HttpSession`

`application` – приложение. Тип объекта – класс, реализующий интерфейс

`javax.servlet.ServletContext`

`out` – выходной поток. Тип объекта – класс `javax.servlet.jsp.JspWriter`

`config` – объект конфигурации сервлета для текущей страницы. Тип объекта – класс, реализующий интерфейс `javax.servlet.ServletConfig`

`page` – объект, обрабатывающий запрос для данной страницы. Тип объекта – класс `Object` .

Данные объекты применяются в виде

```
<%@ имяОбъекта параметр1=значение1 параметр2=значение2 ... %>
```

Пример их использования:

```
<%@ page session=true import="java.util.*" %>
```

Имеется возможность задания сценариев (интерпретируемого кода) с помощью специальных определений вида

```
<%@ код %>
```

Где `код` – сценарий *Java* для работы с документом или вычислений. Например,

```
<%
    for(int i=0; i<table.getEntries().getRows(); i++) {
        cal.Entry entr = table.getEntries().getEntry(i);
    }
%>
```

В частности, разрешается задавать собственные теги вида

```
<%@ имяБиблиотеки prefix="имяОбъекта" uri="путь к библиотеке" %>
```

После чего разрешается использовать теги вида `< имяОбъекта:оператор>`

Так делается в примере использования оператора `out` в *JSTL Example*. Используется *пользовательский объект* с (сокращение от *customer* – "покупатель"), задаваемый как `prefix="c"`, и оператор `out`, заданный в библиотеке `taglib` по адресу `uri="http://java.sun.com/jsp/jstl/core"`:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Отметим, что соответствующие библиотеки уже добавлены в проект. Для примера `out` это пакет `org.apache.taglibs.standard.tag.el.core`.

После запуска данного приложения в появившемся документе в списке *Examples* это первый пример – *General Purpose Tags*. При переходе по данной гиперссылке мы получаем пример

## General-Purpose Tags Examples



Рис. 10.6.

При нажатии на "шестеренки" получаем результат работы программы:

<c:out>		
Richard	no home phone specified	no cell phone specified
Mikita	(320)876-9784	no cell phone specified
Gilbert	no home phone specified	no cell phone specified
Howe	(465)675-0761	(465)879-9802
Sawchuk	no home phone specified	no cell phone specified

Рис. 10.7. Пример использования оператора `c:out`

Соответствующий фрагмент исходного кода этого *JSP*-документа выглядит так:

```
<table border="1">
  <c:forEach var="customer" items="{customers}">
    <tr>
      <td><c:out value="{customer.lastName}"/></td>
      <td><c:out value="{customer.phoneHome}"
        default="no home phone specified"/></td>
      <td>
        <c:out value="{customer.phoneCell}" escapeXml="false">
          <font color="red">no cell phone specified</font>
        </c:out>
      </td>
    </tr>
  </c:forEach>
</table>
```



```
</c:forEach>
</table>
```

В этом примере используется объект `customer` – "покупатель", заданный в файле `Customer.java`, расположенном в пакете `org.apache.taglibs.standard.examples.beans`. То есть самом первом в Source Packages пакете примера. А также объект `customers` – "покупатели", заданный в файле `Customers.java`, расположенном в том же пакете.

В классе `customer` заданы поля `lastName`, `phoneHome`, `phoneCell` и другие. А также ряд методов, которые также можно вызывать в сценарии. С помощью оператора `forEach` (заданного аналогично оператору `out`) осуществляется перебор всех объектов `customer`, агрегированных в объект `customers` – список покупателей. А с помощью тега `c:out` осуществляется вывод необходимой информации в документ.

В JSP имеется огромное количество возможностей. Это тема для отдельной книги. В данном учебном пособии данная технология затронута совсем немного – просто в порядке информирования о ее существовании и некоторых возможностях.

Точно так же, для программирования в локальных и глобальных компьютерных сетях в пакете `java.net` имеется огромное количество средств разного уровня, описание которых требует отдельной книги. Это и Web-адресация (классы `URL`, `URLConnection`, `URI`, `JarURLConnection`, `URLClassLoader`), и IP-адресация (классы `InetAddress`, `InetAddress4`, `InetAddress6`, `NetworkInterface`), и управление соединениями через сокет (классы `Socket`, `SocketAddress`, `InetSocketAddress`, `ServerSocket`, `SocketPermission`). Классы `NetPermission`, `Authenticator` и `PasswordAuthentication` обеспечивают поддержку авторизации (запрос и обработку имени и пароля).

Кроме упомянутых возможностей пакета `java.net` имеется дистрибутив `j2me`, в котором поставляются средства разработки программного обеспечения для "тонких" аппаратных клиентов – то есть таких, которые обладают малыми по сравнению с персональными компьютерами ресурсами. В первую очередь – для сотовых телефонов и неладонных компьютеров. Это также тема для отдельной книги.

Не менее важной и объемной темой является сетевое программирование баз данных.

Так что желающих освоить даже основы сетевых возможностей Java ждет весьма длительная работа.

## Краткие итоги

HTML – язык разметки гипертекста. Гипертекст ("сверхтекст") – это текст, содержащий дополнительные возможности, в частности – гиперссылки. Документ, написанный на языке HTML, представляет собой текстовый файл, содержащий собственно текст, несущий информацию пользователю, и теги разметки (markup tags).

Теги представляют собой определенные последовательности символов, заключенные между знаками '`<`' и '`>`'. Они предназначены для форматирования и разметки документа. Теги бывают парные ("контейнеры") и непарные. Действие парного тега начинается с открывающего тега и заканчивается при встрече соответствующего ему закрывающего, признаком которого является символ '`/`'.

При ссылке на документ, находящийся на другом сервере, необходимо указать адрес (URL – 'Uniform Resources Location') этого документа: сетевой адрес сервера и путь к этому документу на сервере. Если документ находится на том же сервере, но в другой папке, достаточно указать только путь к этой папке. Гипертекстовая ссылка имеет вид `<a href="имя_файла">текст_ссылки</a>`

Для создания обычного апплета требуется задать класс, являющийся наследником класса `java.applet.Applet`, а для апплетов, использующих компоненты `Swing` и/или библиотеку `Sun JFC` (Java Foundation Classes) – наследником класса `javax.swing.JApplet`.

В классе апплета требуется переопределить ряд методов – `init`, `start`, `update`, `paint`, `getAppletInfo`, `stop`, `destroy`.

Сервлеты – это приложения Java, запускаемые со стороны сервера. Они имеют возможности доступа к файловой системе и другим ресурсам сервера через набор управляющих конструкций, определенных в рамках пакета `javax.servlet` и технологии JSP.

Технология JSP заключается в наличии дополнительных конструкций в HTML- или XML-документах, которые позволяют осуществлять вызовы сценариев ("скриптов"), написанных на языке Java. В результате удается очень просто и удобно осуществлять обработку данных или элементов документа, и внедрять в нужные места документа результаты обработки.

## Задания

Написать апплет, рисующий эллипс, и использующий этот апплет HTML-документ. Проверить показ этого HTML-документа в `AppletViewer` и браузере.

Изучить работу и исходный код `TomcatServletExample`. Видоизменить в нем пример `HelloWorldExample`, обеспечив вывод документа с вашими данными: фамилией, именем, отчеством. В конце документа обеспечить вывод даты генерации этого документа. Для этого использовать объект типа `Date` (как преобразовать его в строку?).

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.

---

© Национальный Открытый Университет "ИНТУИТ", 2022 | [www.intuit.ru](http://www.intuit.ru)