# **Модуль 1 «Основы пользовательского интерфейса веб-приложений»**

## **Лекция 1.1. Введение в курс**

Строго говоря, Internet-приложением можно считать любой программный продукт, так или иначе использующий Internet. Но программ, подпадающих под это определение, слишком много! Необходимо классифицировать все Internet-приложения.

Можно выделить четыре типа Internet-приложений:

* [Web-приложения](#веб_приложения), которые работают на сервере, передавая через Internet данные на клиентские машины. Для их применения требуются Web-браузеры, такие как MS IE, Netscape Navigator, Opera и другие;
* Web-сервисы, которые позволяют приложениям обрабатывать их данные на сервере. При этом передача подлежащих обработке данных на сервер и возврат результатов осуществляется через Internet;
* приложения с поддержкой Internet – автономные программы со встроенными механизмами, позволяющими их пользователям регистрироваться, получать обновления, а также предоставляющими доступ к справочной системе и другим вспомогательным службам через Internet.
* одноранговые приложения – автономные программы, использующие Internet для взаимодействия с другими программными продуктами этого же типа.

***Принцип работы Web-приложений***

Web-приложения используют архитектуру “клиент-сервер”. Собственно, Wеb-приложение находится на сервере и обрабатывает запросы, которые передают ему через Internet многочисленные клиенты (Рисунок 1-1).

На стороне клиента Web-приложение работает в браузере. Пользовательский интерфейс Web-приложения передается на клиентскую машину в виде страниц на языке HTML (Hypertext Markup Language), где браузер интерпретирует и отображает их.

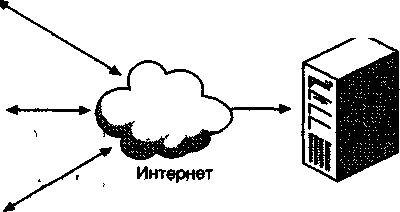


Рисунок 1-1. Архитектура Web-приложений

На стороне сервера Web-приложение работает под управлением IIS (Internet Information Services). IIS управляет работой приложения, передает ему клиентские запросы и возвращает клиентам результаты исполнения их запросов. Запросы и результаты их исполнения передаются через Internet по протоколу HTTP (Hypertext Transport Protocol). Протокол – это набор правил, регламентирующих взаимодействие двух и более сущностей, которое реализуется через среду, такую, как Internet. На Рисунке 1-2 показано, как клиент и сервер взаимодействуют через Internet.

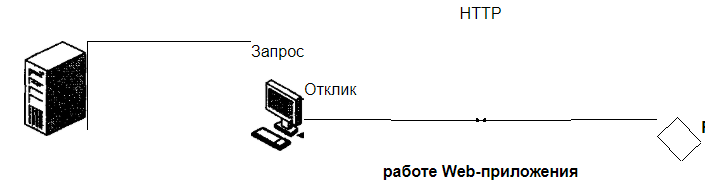


Рисунок 1-2. Взаимодействие клиента и сервера при работе Web-приложения

Web-приложение компонует отклик из серверных ресурсов, к которым относятся исполняемый код, работающий на сервере (то, что традиционно считается «приложением» в Windows-программировании), Web-формы, HTML-страницы, графические файлы и иное содержимое, составляющее информационное наполнение приложений.

Web-приложения во многом напоминают традиционные Web-сайты, но в отличие от них отображают пользователю динамическое содержимое, генерируемое исполняемым кодом приложения, а не статические страницы, хранящиеся на сервере в готовом виде.

Исполняемая часть Web-приложения способна делать многое из того, чего не могут статические Web-сайты, а именно:

* принимать данные от пользователя и сохранять их на сервере;
* выполнять для пользователя различные действия: размещать заказы, делать сложные вычисления и извлекать информацию из баз данных (БД);
* опознавать пользователя и отображать интерфейс, настроенный в соответствии с его предпочтениями;
* отображать постоянно меняющееся содержимое.

Этот перечень далеко не полон. В принципе, Web-приложения способны решить любые вообразимые задачи, доступные и клиент-серверному приложению. Особенность Web-приложений в том, что взаимодействие между клиентом и сервером осуществляется через Internet.

## **Лекция 1.2. Основы реализации фронтенда в веб-приложениях**

Фронтенд — все, что браузер может читать, выводить на экран и / или запускать. То есть это [HTML](#html), [CSS](#CSS) и [JavaScript](#js).

[HTML](#HTML) (HyperText Markup Language) говорит браузеру, каково содержание страницы, например, «заголовок», «параграф», «список», «элемент списка».

CSS (Cascading Style Sheets) говорит браузеру, как отображать элементы, например, «после первого параграфа отступ в 20 пикселей» или «весь текст в элементе bodyдолжен быть темно-серым и написан шрифтом Verdana».

JavaScript говорит браузеру, как реагировать на некоторые взаимодействия, используя легкий язык программирования. Большинство сайтов на самом деле не используют много JavaScript, но, если вы нажмете на что-то и содержимое страницы поменяется без белого мигания экрана, значит, где-то использовался JavaScript.

Бэкенд — все, что работает на сервере, то есть «не в браузере» или «на компьютере, подсоединенном к сети (обычно к Интернету), который отвечает на сообщения от других компьютеров».

Для бэкенда можно использовать любые инструменты, доступные на сервере (который, по сути, является просто компьютером, настроенным для ответов на сообщения). Это означает, что можно использовать любой универсальный язык программирования: Ruby, PHP, Python, Java, JavaScript / Node, bash. Это также означает, что можно использовать системы управления базами данных, такие как MySQL, PostgreSQL, MongoDB, Cassandra, Redis, Memcached.

***Структура взаимодействия*** [***бэкенда***](#бэкенд) ***и*** [***фронтенда***](#фронтенд)

Сегодня существует несколько основных архитектур, определяющих, как будут взаимодействовать ваши бэкенд и фронтенд.

1. ***Серверные приложения***

В этом случае HTTP-запросы отправляются напрямую на сервер приложения, а сервер отвечает HTML-страницей.

Между получением запроса и ответом сервер обычно ищет по запросу информацию в базе данных и встраивает ее в шаблон (ERB, Blade, EJS, Handlebars).

Когда страница загружена в браузере, HTML определяет, что будет показано, CSS — как это будет выглядеть, а JS — всякие особые взаимодействия.

1. ***Связь с использованием*** [***AJAX***](#AJAX)

Другой тип архитектуры использует для связи [AJAX](#AJAX) (Asynchronous JavaScript and XML). Это означает, что JavaScript, загруженный в браузере, отправляет HTTP-запрос (XHR, XML HTTP Request) изнутри страницы и (так сложилось исторически) получает XML-ответ. Сейчас для ответов также можно использовать формат JSON.

Это значит, что у сервера должна быть конечная точка, которая отвечает на запросы JSON- или XML-кодом. Два примера протоколов, используемых для этого — REST и SOAP.

1. ***Клиентские (одностраничные) приложения***

AJAX позволяет загружать данные без обновления страницы. Больше всего это используется в таких фреймворках, как Angular и Ember. После сборки такие приложения отправляются в браузер, и любой последующий рендеринг выполняется на стороне клиента (в браузере).

Такой фронтенд общается с бэкендом через HTTP, используя JSON- или XML-ответы.

1. ***Универсальные/изоморфные приложения***

Некоторые библиотеки и фреймворки, например, React и Ember, позволяют исполнять приложения как на сервере, так и в клиенте.

В этом случае для связи [фронтенда](#фронтенд) с [бэкендом](#бэкенд) приложение использует и AJAX, и обрабатываемый на сервере HTML.

***Вне фронтенда и бэкенда***

1. ***Автономный фронтенд***

Веб-приложениям, которые будут созданы, подключение к Сети будет требоваться всё меньше и меньше.

Прогрессивные веб-приложения загружаются лишь один раз и работают (почти) всегда. Можно хранить базу данных в браузере. В некоторых случаях вашим приложениям нужен бэкенд только при первой загрузке, а затем лишь для синхронизации / защиты данных. Такой уровень постоянства означает, что большая часть логики приложения находится непосредственно в клиенте.

1. ***Легкий бэкенд***

Бэкенд, в свою очередь, становится легче и легче. Такие технологии, как хранилища документов и графовые базы данных, приводят к сокращению количества обращений к бэкенду для повторного агрегирования данных. Задача клиента — уточнить, какие данные ему нужны (базы данных графов), или извлечь все различные фрагменты данных, которые ему нужны (REST API).

Сейчас можно создавать бэкенд-сервисы, которые работают не постоянно, а только тогда, когда они нужны, благодаря бессерверным архитектурам, таким как AWS Lambda.

1. ***Размытые границы***

Вычислительные задачи теперь можно перемещать между фронтендом и бэкендом. В зависимости от вида приложения можно сделать так, чтобы вычисления производились либо в клиенте, либо на сервере.

Каждый из вариантов имеет свои плюсы и минусы. Сервер — среда более стабильная, имеет меньше неизвестных, но ему постоянно нужно подключение к Сети. Некоторые пользователи используют последние версии браузеров, и им выгоднее использовать клиентские приложения, которые и делают большую часть работы, и могут похвастаться красивым интерфейсом, но тогда вы оттолкнёте пользователей, которые не используют новейшие браузеры и высокоскоростное подключение к Интернету.

В любом случае, хорошо, что есть, из чего выбирать. Главное — выбирать именно то, что лучше всего подходит для конкретной задачи.

## **Лекция 1.3. Язык гипертекстовой разметки HTML**

***Структура документа HTML***

Документ HTML 4 состоит из трех частей:

* строка, содержащая информацию о версии HTML,
* объявляющий раздел header/"шапка" (ограниченный элементом HEAD),
* тело, содержащее собственно сам документ.

Тело может содержаться в элементах BODY или FRAMESET. Пробельные символы (пробелы, символы новой строки, символы табуляции и комментарии) могут появляться до или после этого раздела.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

<HTML>

<HEAD>

<TITLE> Simple page </TITLE>

</HEAD>

<BODY>

<H1>Hello world!</H1>

</BODY>

</HTML>

Документ начинается с элемента типа документа, или doctype. Он описывает, какой тип HTML будет использован - чтобы клиентское приложение пользователя могло определить, как интерпретировать документ, и решить, следует ли он тем правилам, которым собирался следовать по своему заявлению.

После этого можно видеть открывающий тег элемента html. Это оболочка вокруг всего документа. Закрывающий тег html является последним объектом в любом документе HTML.

Внутри элемента html имеется элемент head. Он содержит информацию о документе (метаданные). Внутри head находится элемент title, который определяет заголовок "Simple page" в панели меню.

После элемента head следует элемент body, который является оболочкой, содержащей реальное содержимое страницы - в данном случае только элемент заголовка первого уровня (h1), который содержит текст "Hello world!" .

Элементы часто содержат другие элементы. Тело документа всегда будет содержать множество вложенных друг в друга элементов.

Разделы страницы создают общую структуру документа, и могут содержать подразделы. Они также могут содержать заголовки, параграфы, списки и т.д. Параграфы могут содержать элементы, которые создают ссылки на другие элементы, цитаты, выделения и т.д.

***Синтаксис элементов HTML***

Базовый элемент в HTML состоит из двух тэгов вокруг блока текста. Существуют элементы, которые не являются оболочкой для текста, и почти в каждом случае элементы могут содержать подэлементы (как html содержит head и body в примере выше).

Элементы могут также иметь атрибуты, которые могут модифицировать поведение элемента и вводить дополнительное значение.

<div id="masthead">

<h1>Основы

<abbr title="Hypertext Markup Language">HTML</abbr>

</h1>

</div>

В этом примере элемент div (раздел страницы, способ разбиения документов на логические блоки) имеет добавленный атрибут id, для которого задано значение masthead. Элемент div содержит элемент h1 (заголовок первого, или самого важного уровня), который в свою очередь содержит некоторый текст. Часть этого текста упакована в элемент abbr (который используется для определения расширения сокращений), который имеет атрибут title, значение которого задано как Hypertext Markup Language.

Многие атрибуты в HTML являются общими для всех элементов, но некоторые являются специфическими для данного элемента или элементов. Все они имеют форму:

ключевое\_слово="значение"

Значение должно быть помещено в одиночные или двойные кавычки (в некоторых ситуациях кавычки могут отсутствовать, но это не слишком хорошо с точки зрения предсказуемости, понимания).

Атрибуты и их возможные значения определяются в основном спецификациями HTML (http://www.w3.org/TR/html401/index/attributes.html), поэтому - нельзя создавать свои собственные атрибуты. Единственными реальными исключениями являются атрибуты id и class, значения полностью которых предназначены для добавления в документы вашего собственного значения и семантики.

Элемент внутри другого элемента называют "потомком" этого элемента. В примере выше abbr является потомком h1, который в свою очередь является потомком div. И наоборот, div является "предком" элемента h1.

***Элементы блочного уровня и строковые элементы***

Имеется две основные категории элементов в HTML, которые соответствуют типам контента и структуре, которую представляют эти элементы - элементы [блочного уровня](#Блочный_уровень) и [строковые элементы](#Строковые_элементы).

*Блочный уровень* означает более высокий уровень элемента, обычно информирующий о структуре документа. Элементы блочного уровня можно представлять как элементы, которые начинаются с новой строки, отрываясь от того, что было перед этим. Распространенными блочными элементами являются параграфы, пункты списка, заголовки и таблицы.

*Строковые элементы* содержатся внутри структурных элементов блочного уровня и охватывают только части текста документа, а не целые области. Строковый элемент не приводит к появлению в документе новой строки, т.к. они являются элементами, которые появляются в параграфе текста. Распространенными строковыми элементами являются гипертекстовые ссылки, выделенные слова или фразы и краткие цитаты.

***Заголовок***

Заголовок HTML-документа является необязательным элементом разметки. Первоначально существование заголовка определялось необходимостью именования окна браузер. Это достигалось за счет элемента разметки TITLE:

<HTML>

<HEAD>

<TITLE>Это заголовок</TITLE>

...

</HEAD>

<BODY>

...

</BODY>

</HTML>

Еще одной функцией заголовка HTML-документа является управление HTTP-обменом через элемент разметки META. При современной практике размещения Веб-узлов компаний на серверах провайдеров администраторы этих узлов могут не иметь возможности управлять программой-сервером. В этом случае для управления обменом остается только одна возможность - через заголовок HTML-документа.

Заголовок HTML-документа также предназначен для описания поискового образа документа, необходимого для индексирования документа роботами поисковых систем. Элемент [META](#МЕТА) позволяет хранить списки ключевых слов и описания документа, которые будут использоваться для составления индекса поисковой системы и появляться в качестве описания документа в случае выдачи ссылки на него при поиске по ключевым словам.

Основные теги заголовка - это элементы HTML-разметки, которые наиболее часто встречаются в заголовке HTML-документа, т.е. внутри элемента разметки HEAD:

* [TITLE](#TITLE) (заглавие документа);
* [BASE](#BASE) (база URL);
* [ISINDEX](#ISINDEX) (поисковый шаблон);
* [META](#МЕТА) (метаинформация);
* [LINK](#LINK) (общие ссылки);
* [STYLE](#STYLE) (описатели стилей);
* [SCRIPT](#SCRIPT) (сценарии).

Чаще всего применяются элементы TITLE, SCRIPT, STYLE. Использование элемента [META](#МЕТА) говорит об осведомленности автора о правилах индексирования документов в поисковых системах и возможности управления HTTP -обменом данными. BASE и ISINDEX в последнее время практически не применяются. LINK указывают только при использовании внешних относительно данного документа описателей таблиц стилей.

Элемент разметки HEAD содержит заголовок HTML-документа. Данный элемент разметки не является обязательным. При наличии тега начала элемента разметки желательно использовать и тег конца элемента разметки. По умолчанию элемент HEAD закрывается, если встречается либо тег начала контейнера BODY, либо тег начала контейнера FRAMESET.

Контейнер заголовка служит для размещения информации, относящейся ко всему документу в целом.

Элемент разметки TITLE служит для именования документа в World Wide Web. При выборе текста для содержания контейнера TITLE следует учитывать, что отображается он системным шрифтом, так как является заголовком окна браузера.

Синтаксис контейнера TITLE в общем виде выглядит следующим образом:

<TITLE>название документа</TITLE>

Заголовок не является обязательным контейнером документа. Его можно опустить. Роботы многих поисковых систем используют содержание элемента TITLE для создания поискового образа документа. Слова из TITLE попадают в индекс поисковой системы. Из этих соображений элемент TITLE всегда рекомендуется использовать на страницах Веб-узла.

Элемент разметки BASE служит для определения базового URL для гипертекстовых ссылок документа, заданных в неполной (частичной) форме. Кроме того, BASE позволяет определить целевое окно загрузки документа по умолчанию при выборе гипертекстовой ссылки текущего документа. Наиболее часто BASE встречается на страницах узлов, которые имеют "зеркала". Часть документов основного сервера по разным причинам на "зеркальный" сервер не переносится. В этом случае документ с принудительно заданным базовым URL всегда будет ссылаться на основной сервер.

Тег начала контейнера содержит один обязательный атрибут HREF, и может содержать один необязательный атрибут TARGET. Синтаксис контейнера BASE в общем виде выглядит следующим образом:

<BASE HREF="http://www.example.com/intro.html">

<BASE HREF=http://www. example.com/intro.html TARGET=next>

Элемент разметки ISINDEX используется для указания поискового шаблона и унаследован от ранних версий HTML. В HTML 4.0 этот контейнер не определен.

***Элемент разметки*** ***META***

META содержит управляющую информацию, которую браузер использует для правильного отображения и обработки содержания тела документа, например, с помощью атрибута Content-type можно задать перекодировку документа на стороне клиента.

С помощью [META](#МЕТА) также можно задать и другие операторы. Например, запретить кэширование документа. Для запрета кэширования достаточно вставить в заголовок META -тег вида:

<META HTTP-EQUIV="Pragma" CONTENT="no-cache">

В новой версии протокола HTTP (HTTP 1.1) управление кэшированием осуществляется через оператор Cache-Control. Для получения такого же результата, как в случае с Pragma, в заголовке HTML -документа достаточно указать:

<META HTTP-EQUIV="Cache-Control"

CONTENT="no-cache">

Можно запретить хранение документа после пересылки:

<META HTTP-EQUIV="Cache-Control"

CONTENT="no-store">

Точно так же можно задать время последней модификации (Last-Modified) или дату истечения актуальности документа ( Expire ).

META-тег часто используется для описания поискового образа документа.

Собственно, для описания документа используется два META-тега. Один определяет список ключевых слов, а второй - краткое содержание документа. Контейнер TITLE здесь также используется в качестве названия документа.

<TITLE> Веб-технологии</TITLE>

<META NAME="description"

http-equiv="description"

content="Учебный курс Веб-технологии.

Тема: Обзор структуры и возможносетй языка гипертекстовой разметки.">

<META NAME="keywords" HTTP-EQUIV="keywords"

CONTENT="учебный курс; Веб-технологии;

HTML 4.1; язык гипертекстовой разметки;

заголовок HTML-документа; заголовок; HTML;

документ; контейнер; элемент; разметка">

META-тег используют также программы подготовки документов., размещая в нем свой идентификатор. В общем случае контейнер META выглядит следующим образом:

<META [name=имя]

[HTTP-EQUIV=имя\_HTTP-оператора]

CONTENT=текст>

***Элемент разметки LINK***

Элемент разметки LINK - это результат давно предпринятой попытки придать HTML академический вид. Согласно теории гипертекстовых систем, все гипертекстовые связи разделяют на два типа: контекстные и общие. Такое деление чисто условное и определяется тем, что контекстную связь можно привязать к определенному месту документа, а общую - отнести только ко всему документу целиком. Гипертекстовая связь задает отношение на множестве информационных узлов.

Контекстная связь определяет отношение на паре узлов. При этом в модели World Wide Web один из узлов является источником, а второй - целью (target). Собственно, это и отражено в названии элемента разметки A (anchor), который определяет гипертекстовую ссылку (не путать с гипертекстовой связью).

Общие ссылки нельзя привязать по контексту. Например, два информационных узла находятся в отношении следования, т.е. при "линейном" просмотре одна Веб-страница является следующей для другой Веб-страницы. В этом случае речь идет о страницах целиком, а не об отдельных их частях. Такой же общей связью является принадлежность к Веб-узлу, который ассоциируется со своей домашней страницей.

В настоящее время в браузерах не существует единого способа программирования или определения общих гипертекстовых связей.

Существенный сдвиг в этом направлении произошел после реализации поддержки описателей стилей в веб-браузерах. [CSS](#CSS) (Cascade STYLE Sheets, каскадные таблицы стилей) позволяют определять для различных типов гипертекстовых связей вид гипертекстовых ссылок. При этом можно определять различные типы контекстных ссылок. Контейнер LINK позволил загружать внешние описатели стилей:

<LINK REL=stylesheet href="../css/style.css"

TYPE="text/css">

В данном случае атрибут REL определяет тип гипертекстовой связи, HREF (Нуреrtехt REFerence) указывает адрес документа, идентифицирующего связь, а атрибут TYPE определяет тип содержания этого документа.

В общем случае контейнер LINK имеет следующий вид:

<LINK [REL=тип\_отношения] [HREF=URL]

[TYPE=тип\_содержания]>

Для разных типов содержания действия по интерпретации элемента разметки будут различными.

***Элемент разметки STYLE***

Элемент разметки STYLE предназначен для размещения описателей стилей. При этом описание стиля из данного элемента разметки, если оно совпадает по имени класса и/или идентификатору подкласса со стилем, описанным во внешнем файле, заменяет описание стиля из внешнего файла. С точки зрения влияния на весь документ, описатели стилей задают правила отображения контейнеров HTML -документа для всей страницы.

В настоящее время контейнер используется только с одним атрибутом TYPE, который задает тип описателя стиля. Это может быть либо text/css, либо text/javascript. Если элемент разметки открыт открывающим тегом, то он должен завершаться закрывающим тегом. В общем виде запись элемента STYLE выглядит так:

<STYLE TYPE=тип\_описания\_стилей>

описание стиля/стилей

</STYLE>

***Элемент разметки SCRIPT***

Элемент разметки SCRIPT служит для размещения кода JavaScript, VBScript или JScript. В принципе, SCRIPT можно использовать не только в заголовке документа, но и в его теле. В отличие от контейнера STYLE, ему не требуется дополнительный контейнер LINK для загрузки внешних файлов кодов. Это можно сделать непосредственно в самом контейнере SCRIPT:

<SCRIPT TYPE="text/javascript"

SRC=script.code

>

В общем виде запись контейнера выглядит следующим образом:

<SCRIPT [TYPE=тип\_языка\_сценариев]>

JavaScript / VBScript - код

</SCRIPT>

или

<SCRIPT [TYPE=тип\_языка\_сценариев]

[SRC=URL]>

</SCRIPT>

Веб-браузерами поддерживается несколько сценарных языков: JavaScript,VBScript, JScript. По умолчанию подразумевается JavaScript.

## **Лекция 1.4. Семантика тегов**

Семантические элементы HTML5 доступно описывают свой смысл или назначение как для браузеров, так и для веб-разработчиков.  
До появления стандарта HTML5 вся разметка страниц осуществлялась преимущественно с помощью элементов <div>, которым присваивали классы class или идентификаторы id для наглядности разметки (например, <div id="header">). С их помощью в HTML-документе размещали верхние и нижние колонтитулы, боковые панели, навигацию и многое другое.

Стандарт HTML5 предоставил новые элементы для структурирования, группировки контента и разметки текстового содержимого. Новые семантические элементы позволили улучшить структуру веб-страницы, добавив смысловое значение заключенному в них содержимому (было <div id="header">, стало <header>). Для отображения внешнего вида элементов не задано никаких правил, поэтому элементы можно стилизовать по своему усмотрению. Для всех элементов доступны [‎глобальные атрибуты](https://html5book.ru/html-attributes/).

Согласно спецификации HTML5 каждый элемент принадлежит к определенной (ноль или более) категории. Каждая из них группирует элементы со схожими характеристиками. Выделяют следующие общие категории:

* Мета содержимое
* Потоковое содержимое
* Секционное содержимое
* Заголовочное содержимое
* Текстовое содержимое
* Встроенное содержимое
* Интерактивное содержимое
* Описание HTML5-элементов

1. ***Элемент <header>***

Категории контента: потоковое содержимое.

Группирует вводные и навигационные элементы, не является обязательным. Может содержать заголовки, оборачивать содержание раздела страницы, форму поиска или логотип. В HTML-документе может содержаться одновременно несколько элементов <header> и они могут располагаться в любой части страницы.

Элемент <header> нельзя помещать внутрь элементов <footer>,  <address> или другого элемента <header>.

1. ***Элемент <nav>***

Категории контента: потоковое содержимое, секционное содержимое.

Предназначен для создания блока навигации веб-страницы или всего веб-сайта, при этом не обязательно должен находиться внутри <header>. На странице может быть несколько элементов <nav>. Не заменяет теги <ul> или <оl>, он просто их обрамляет. Не все группы ссылок на странице должны быть обёрнуты <nav>, этот элемент предназначен в первую очередь для разделов, которые состоят из главных навигационных блоков.

В качестве элементов панели навигации можно использовать не только элементы списков. Также можно добавлять заголовки внутрь элемента.

1. ***Элемент <article>***

Категории контента: потоковое содержимое, секционное содержимое.

Используется для группировки записей — публикаций, статей, записей блога, комментариев. Представляет собой независимый обособленный блок, предназначенный для многократного использования, как правило, начинается с заголовка. Может дублироваться на других страницах сайта и содержать внутри другие элементы <article>, которые по содержанию имеют близкое отношение к содержанию внешней статьи. Если на странице присутствует только одна статья с заголовком и текстовым содержимым, она не нуждается в обёртке элементом <article>. Элемент рекомендуется использовать только в том случае, если содержимое элемента будет явно указано в схеме документа.

1. ***Элемент <section>***

Категории контента: потоковое содержимое, секционное содержимое.

Элемент представляет собой универсальный раздел документа. Группирует тематическое содержимое и обычно содержит заголовок. Не является блоком-оберткой, для этих целей уместнее использовать элемент <div>. В качестве содержимого может выступать оглавление, разделы научных публикаций, программа мероприятия. Домашняя страница сайта также может быть поделена на секции — блок вводной информации, новости и контакты. Элемент рекомендуется использовать только в том случае, если содержимое элемента будет явно указано в схеме документа.

***<article> внутри <section>***

Можно создавать родительские элементы <section> с вложенными элементами <article>, в которых есть один или несколько элементов <article>. Не все страницы должны быть устроены именно так, но это допустимый способ вложения элементов. Например, основная область контента страницы содержит два блока со статьями разной тематики. Можно сделать на этом акцент, поместив каждую статью одной тематики внутрь элемента <section>.

1. ***Элемент <aside>***

Категории контента: потоковое содержимое, секционное содержимое.

Группирует содержимое, связанное с окружающим его контентом напрямую, но которое можно счесть отдельным (т.е., удаление этого блока не повлияет на понимание основного содержимого). Чаще всего элемент позиционируется как боковая колонка (как в книгах) и включает в себя группу элементов: <nav>, цифровые данные, цитаты, рекламные блоки, архивные записи. Не подходит для блоков, просто позиционированных в стороне.

1. ***Элемент <footer>***

Категории контента: потоковое содержимое.

Представляет собой нижний колонтитул содержащей его секции или корневого элемента. Обычно содержит информацию об авторе статьи, данные о копирайте и т.д. Если используется как колонтитул всей страницы, содержимое дополняется сведениями об авторских правах, ссылками на условия использования, контактную информацию, ссылками на связанное содержимое и т.п.

В одном веб-документе может быть несколько элементов <footer>. Как каждая страница, так и каждая статья может иметь свой элемент <footer>, более того, <footer> можно поместить в элемент <blockquote>, чтобы указать источник цитирования.

<footer>

<address>...</address>

<small>@2014...</small>

</footer>

1. ***Элемент <address>***

Категории контента: потоковое содержимое.

Используется для определения контактной информации автора/владельца документа или статьи. Для обозначения автора документа тег размещают внутри элемента <body>, для отображения автора статьи — внутри тега <article>. В браузере обычно отображается курсивом.

1. ***Элемент <main>***

Категории контента: потоковое содержимое.

Элемент <main> представляет основное содержимое документа (содержимое элемента <body>). Контент, находящийся внутри элемента, должен быть уникальным и не повторяться во всех документах сайта, таких как навигационные ссылки, информация о копирайте, логотипы, формы поиска (в случае, если форма поиска является основной функцией документа).

Элемент <main> не может быть потомком таких элементов как <article>, <aside>, <footer>, <header> или <nav>.

1. ***Элемент <figure>***

Категории контента: потоковое содержимое, корневое секционное содержимое.

Элемент <figure> представляет автономный контент (необязательно с заголовком), являющийся самостоятельным элементом основного потока. Элемент может быть перемещен из основного содержимого документа в боковую колонку или приложение, не затрагивая поток документа. С помощью элемента <figure> можно добавлять краткие характеристики к иллюстрациям, фотографиям, диаграммам, фрагментам кода и т.д.

Элемент <figure> является блочным, по ширине занимает всю ширину блока-контейнера за минусом внешних отступов margin.

1. ***Элемент <figcaption>***

Элемент <figcaption> — потомок элемента <figure>, не принадлежит ни к одной категории контента. Элемент является блочным, по ширине равен ширине элемента <figure>, высота по умолчанию равна 18px.

1. ***Элемент <time>***

Категории контента: потоковое содержимое, текстовое содержимое.

Определяет время (24 часа) или дату по григорианскому календарю с возможным указанием времени и смещения часового пояса. Текст, заключенный в данный тег, не имеет стилевого оформления браузером. Для тега доступен атрибут datetime, в качестве содержимого которого указывается то, что будет видеть пользователь на экране своего компьютера:

<time datetime="2014-09-25"> 25 Сентября 2014</time>

Чтобы дата могла считываться автоматически, она должна быть в формате YYYY-MM-DD. Время, которое также может указываться, задается в формате HH:MM с добавлением разделяющего префикса T (time):

<time datetime="2014-09-25T12:00"> 25 Сентября 2014</time>

1. ***Элемент <mark>***

Категории контента: потоковое содержимое, текстовое содержимое.

Текст, помещенный внутрь тега <mark>, выделяется по умолчанию желтым цветом (цвет фона и цвет шрифта в выделенном блоке можно изменить, задав определенные css-стили). С помощью данного тега можно отмечать важное содержимое, а также ключевые слова.

1. ***Элемент <bdi>***

Категории контента: потоковое содержимое, текстовое содержимое.

Отделяет фрагмент текста, который должен быть изолирован от остального текста для двунаправленного форматирования текста. Используется для текстов, написанных одновременно на языках, читающихся слева направо и справа налево.

1. ***Элемент <wbr>***

Категории контента: потоковое содержимое, текстовое содержимое.

Одиночный тег, показывает браузеру место, где можно добавить разрыв длинной строки в случае необходимости.

## **Лекция 1.5. Каскадные таблицы стилей**

В то время как HTML структурирует документ и сообщает веб-браузеру, какую функцию имеет определенный элемент, CSS выдает браузеру инструкции о том, как отобразить определенный элемент - оформление, размещение пробелов и позиционирование.

CSS - одна из широкого спектра технологий, одобренных консорциумом W3C и получивших общее название "стандарты Web".

|  |  |
| --- | --- |
| начало 1990-х годов | Различные браузеры имели свои стили для отображения веб страниц. HTML развивался очень быстро и был способен удовлетворить все существовавшие на тот момент потребности по оформлению информации, поэтому CSS не получил тогда широкого признания |
| 1994 | Появился термин "каскадные таблицы стилей" |
| 1996 | Консорциумом W3C была издана рекомендация CSS1 |
| 1998 | Консорциумом W3C принята рекомендация CSS2 |
| Сентябрь 2009 | Консорциумом W3C утверждена рабочая версия CSS2.1 |

***Уровень 1 (CSS1)***

Это рекомендация W3C была принята в 1996 году и откорректирована в 1999 году. Основные возможности, предоставляемых этой рекомендацией:

* Параметры шрифтов. Возможности по заданию гарнитуры и размера шрифта, а также его стиля - обычного, курсивного или полужирного.
* Цвета. Спецификация позволяет определять цвета текста, фона, рамок и других элементов страницы.
* Атрибуты текста. Возможность задавать межсимвольный интервал, расстояние между словами и высоту строки (то есть межстрочные отступы)
* Выравнивание для текста, изображений, таблиц и других элементов.
* Свойства блоков, такие как высота, ширина, внутренние (padding) и внешние (margin) отступы и рамки. Так же в спецификацию входили ограниченные средства по позиционированию элементов, такие как float и clear.

***Уровень 2 (CSS2)***

Это рекомендация W3C была принята в 1998 году. Она построена на CSS1 с сохранением обратной совместимости. Среди новых возможностей можно назвать следующие:

* Блочная вёрстка. Появились относительное, абсолютное и фиксированное позиционирование. Позволяет управлять размещением элементов по странице без табличной вёрстки.
* Типы носителей. Позволяет устанавливать разные стили для разных носителей (например монитор, принтер, КПК).
* Звуковые таблицы стилей. Определяет голос, громкость и т. д. для звуковых носителей (например для слепых посетителей сайта).
* Страничные носители. Позволяет, например, установить разные стили для элементов на чётных и нечётных страницах при печати.
* Расширенный механизм селекторов.
* Указатели.
* Генерируемое содержание. Позволяет установить текст или картинку, который будет отображаться до или после нужного элемента.

***Уровень 2.1 ( CSS 2.1 )***

Рабочая версия W3C от 8 сентября 2009 года. Построена на CSS2, в ней исправлен ряд ошибок.

***CSS-верстка***

До появления CSS оформление веб-страниц осуществлялось непосредственно внутри содержимого документа. Однако с появлением CSS стало возможным принципиальное разделение содержания и представления документа. За счёт этого нововведения стало возможным лёгкое применение единого стиля оформления для массы схожих документов, а также быстрое изменение этого оформления.

Использование [CSS](#CSS) дает следующие преимущества:

* Несколько дизайнов страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину, во время печати меню не будет выводиться, а на КПК и сотовом телефоне меню будет следовать за содержимым.
* Уменьшение времени загрузки страниц сайта за счет переноса правил представления данных в отдельный CSS-файл. В этом случае браузер загружает только структуру документа и данные, хранимые на странице, а представление этих данных загружается браузером только один раз и кешируется.
* Простота последующего изменения дизайна. Не нужно править каждую страницу, а лишь изменить CSS-файл.
* Дополнительные возможности оформления. Например, с помощью CSS-вёрстки можно сделать блок текста, который остальной текст будет обтекать (например, для меню) или сделать так, чтобы меню было всегда видно при прокрутке страницы.

Известны также и недостатки:

* Различное отображение вёрстки в различных браузерах (особенно устаревших), которые по разному интерпретируют одни и те же данные CSS.
* Часто встречающаяся необходимость на практике исправлять не только один CSS-файл, но и теги HTML и код PHP, которые сложным и ненаглядным способом связаны с селекторами CSS, что иногда сводит на нет простоту применения единых файлов стилей и значительно удлиняет время редактирования и тестирования.

Отображение элементов реализуется с помощью системы правил, которые определяют, какие элементы HTML должны быть дополнительно оформлены, и в каждом правиле перечислятся свойства (например, цвет, размер, шрифт, и т.д.) этих элементов HTML, которыми они будут манипулировать, какие значения будут для них заданы.

Таким образом, CSS не является ни языком программирования, ни языком разметки.

***Определение правил стилей***

Базовым конструкцией в CSS является правило следующего вида:

селектор {

свойство1: значение;

свойство2: значение;

свойство3: значение;

}

*Селектор* определяет элементы HTML, к которым будет применяться правило, определяемые реальным названием элемента, например, body, или другими средствами, такими как значения атрибута class.

*Фигурные скобки* {} содержат пары свойство/значение, которые разделяются между собой точкой с запятой; свойства отделяются от своих соответствующих значений двоеточием.

*Свойства* определяют, что вы хотите сделать с выделенными элементами. Они могут задавать, например, цвет элемента, цвет фона, позицию, поля, заполнение, тип шрифта, и многое другое.

*Значения* являются теми конкретными характеристиками, которые вы хотите задать каждому свойству выделенных элементов. Эти значения зависят от свойства.

*Свойства*, которые влияют на положение, поля, ширину, высоту и т.д. могут измеряться в пикселях, em, процентах, сантиметрах или других аналогичных единицах измерения.

Рассмотрим конкретный пример:

p {

margin: 10px;

font-family: Times New Roman;

color: green;

}

Это правило выбирает HTML элемент p, и для каждого элемента p в документах HTML, которые используют этот код CSS, будет применяться это правило, если только не будут существовать применяемые к ним более конкретные правила, которые будут переопределять это правило. Данное правило влияет на свойства, которые определяют поля вокруг параграфа (margin), шрифт текста в параграфе (font-family), и цвет этого текста (color). Поля задаются размером в 10 пикселей, шрифт задается как Times New Roman, а цвет текста задается как green.

Все множество подобных правил совместно формируют таблицу стилей. Кроме таких правил в CSS существуют и другие конструкции, например, комментарии CSS, селекторы объединения в группу.

***Комментарии CSS***

Комментарии добавляют, помещая их между символами /\* и \*/. Комментарии могут охватывать несколько строк, и в этом случае браузер будет игнорировать эти строки:

1. ***Объединения селекторов в группу***

Можно также объеднить в группу различные селекторы. Предположим, что вы хотите применить одинаковое оформление к h2 и p, тогда можно было бы написать следующий CSS:

h2 {color: red}

p {color: red}

Однако, можно сократить код CSS, группируя селекторы вместе с помощью запятой - правила в скобках применяются к обоим селекторам:

h2, p {color: red}

Существует несколько различных селекторов, каждый из которых соответствуют различным частям разметки. Тремя наиболее общими селекторами, которые встречаются чаще всего, являются следующие:

|  |  |  |
| --- | --- | --- |
| Синтаксис | Селектор | Описание |
| p {} | селектор элемента | соответствует всем элементам на странице с указанным названием (элементам p, в приведенном выше случае) |
| example{} | селектор класса | соответствует всем элементам, которые имеют атрибут class с указанным значением, так что пример выше будет соответствовать <p class="example">, <li class="example"> или <div class="example">, или любому другому элементу со значением class = "example". |
| #example{} | селектор id | соответствует всем элементам, которые имеют атрибут id с указанным значением, так что пример выше будет соответствовать <p id="example">, <li id="example"> или <div id="example">, или любому другому элементу со значением id = "example". Селекторы id не проверяют название никакого элемента, и можно иметь только один селектор для каждого id в документе HTML - они являются уникальными для каждой страницы. |

1. ***Дополнительные селекторы CSS***

С помощью селекторов CSS, селекторов элемента, класса и id-селекторов можно реализовать многое, но это, конечно, не все возможные селекторы - существуют другие селекторы, которые позволяют выбирать элементы для стилевого оформления на основе более специфических критериев:

|  |  |
| --- | --- |
| Селекторы | Описание |
| Универсальные селекторы | можно использовать для выбора каждого элемента на странице |
| Селекторы атрибутов | позволяют выбирать элементы на основе их атрибутов |
| Селекторы потомков | если необходимо выбрать определенные элементы, которые являются потомками других конкретных элементов |
| Селекторы нижележащих | если необходимо выбирать определенные элементы, которые являются нижележащими относительно других конкретных элементов (не просто прямыми потомками, но также расположенные ниже в дереве) |
| Селекоры смежных одноуровневых | если необходимо выбрать только определенные элементы, которые следуют за другими определенными элементами |
| Псевдо-классы | эти селекторы позволяют оформить элементы не на основе того, чем является элемент, но на основе более сложных факторов, скажем, таких как состояние ссылки (если на нее, например, наведен курсор, или она уже была посещена). |
| Псевдо-элементы | эти селекторы позволяют оформить определенные части элементов, а не весь элемент (например, первую букву в этом элементе), они позволяют также вставлять содержимое перед или после определенных элементов |

1. ***Универсальные селекторы***

[Универсальные селекторы](#Универсальные_селекторы) выбирают каждый элемент на странице для применения к ним стилей оформления. Например, следующее правило определяет, что для каждого элемента на странице должна быть добавлена сплошная черная граница толщиной 1 пиксель:

{

border: 1px solid #000000;

}

1. ***Селекторы атрибутов элементов***

Селекторы атрибутов позволяют выбирать элементы на основе содержащихся в них атрибутов. Например, можно выбрать каждый элемент img с атрибутом alt с помощью следующего селектора:

img[alt] {

border: 1px solid #000000;

}

Используя приведенный выше селектор, можно создать черную границу вокруг любого изображения, которое имеет атрибут alt, и оформить другие изображения ярко-красной.

Можно выбирать элементы также и по значению атрибута, а не только по названию атрибута. Следующее правило задает все изображения с атрибутом src со значением alert.gif:

img[src="alert.gif"] {

border: 1px solid #000000;

}

Значительно более полезной является возможность выбирать определенные части атрибутов, например, расширения файлов. CSS3 фактически вводит три новых типа селекторов атрибутов, которые могут выбирать на основе текстовой строки в значениях атрибутов (в начале, конце, или в любом месте в значении).

1. ***Селекторы потомков элементов***

Можно использовать селектор потомка для выбора только определенных элементов, которые являются потомками других определенных элементов. Например, следующее правило задает цвет текста только тех strong элементов, которые являются потомками элементов h3, как green, но не для других элементов strong:

h3 > strong {

color: green;

}

Селекторы потомоков не поддерживаются в браузере IE 6 (и более младших версиях).

1. ***Селекторы нижележащих элементов***

Селекторы нижележащих элементов очень похожи на селекторы потомков, за исключением того, что селекторы потомков выбирают только непосредственно нижележащих, а селекторы нижележащих выбирают подходящие элементы в любом месте иерархии элементов, а не только непосредственно нижележащих. Рассмотрим пример.

<div>

<em>Привет</em>

<p>и сразу же

<em>Пока</em>.

</p>

</div>

В этом фрагменте элемент div является предком всех других элементов. Он имеет двух потомоков, em и p. Элемент p имеет один элемент потомок, еще один em.

1. ***Селекторы смежных одноуровневых элементов***

Эти селекторы позволяют выбирать определенный элемент, который следует непосредственно после другого определенного элемента на том же уровне в иерархии элементов. Например, если вы хотите выбрать элементы p, которые следуют непосредственно после элементов h2, но никакие другие элементы p, можно воспользоваться следующим правилом:

h2 + p {

...

}

Селекторы смежных одноуровневых элементов не поддерживаются в браузере IE 6 (и более младших версиях).

1. ***Псевдо-классы***

Псевдо-классы используются для обеспечения стилевого оформления не для элементов, а для различных состояний элементов. Наиболее обычным применением, которое можно встретить, является оформление состояний ссылок.

Список ниже представляет различные псевдо-классы и описание состояния ссылки, которое они выбирают:

|  |  |
| --- | --- |
| :link | обычное состояние ссылок по умолчанию, когда вы впервые их находите |
| :visited | ссылки, которые вы уже посетили в используемом в данный момент браузере |
| :focus | ссылка (или поле формы, или что-то еще), в которой в данный момент находится курсор клавиатуры |
| :hover | ссылка, на которой в данный момент находится указатель мыши |
| :active | ссылка, на которой в данный момент происходит щелчок. |

Следующие правила CSS определяют что:

* по умолчанию ссылки будут синими.
* когда курсор мыши оказывается над ссылкой, используемое по умолчанию подчеркивание ссылки исчезает.
* когда ссылка будет посещена, она станет серой.
* когда ссылка активна, она становится жирной, как дополнительный признак, что что-то сейчас произойдет.

a:link{

color: blue;

}

a:hover{

text-decoration: none;

}

a:visited{

color: gray;

}

a:active{

font-weight: bold;

}

Если не определять эти правила в том же порядке, как они показаны выше, они могут работать не так, как вы ожидаете. Это обусловлено правилом специфичности, заставляющем более поздние правила в таблице стилей переопределять более ранние правила.

Можно выделить поле ввода, которое содержит активный мигающий курсор с помощью следующего правила:

input:focus {

border: 2px solid black;

background-color: lightgray;

}

1. ***Псевдо-элементы***

[Псевдо-элементы](#Псевдо_элементы) имеют два назначения. Прежде всего, можно использовать их для выбора первой буквы или первой строки текста в заданном элементе. Чтобы создать буквицу в начале каждого параграфа документа, можно использовать следующее правило:

p:first-letter {

font-weight: bold;

font-size: 250%

background-color: red;

}

Чтобы сделать первую строку каждого параграфа жирной, можно использовать следующее правило:

p:first-line {

font-weight: bold;

}

Вторым применением псевдо-элементов является генерация контента с помощью CSS, что является более сложной задачей. Можно использовать псевдо-элементы :before или :after для определения того, что содержимое должно быть вставлено перед или после элемента, который вы выбираете. Затем определяют то, что должно быть вставлено. В качестве простого примера можно использовать следующее правило для вставки декоративных изображений после каждой ссылки на страницу:

a:after{

content: " " url(flower.gif);

}

Можно также использовать функцию attr() для вставки значений атрибутов элементов после элемента. Например, можно вставить цель каждой ссылки в документ в скобках с помощью следующего кода:

a:after{

content: "(" attr(href) ")";

}

Такие правила прекрасно подходят для таблиц стилей печати, которые являются таблицами стилей, которые автоматически применяются, когда пользователь печатает страницу. Преимущество для пользователя состоит в том, что можно скрыть всю навигацию, которую невозможно использовать в печатном материале, и использовать описанную выше технику, чтобы читатель мог видеть адреса URL, на которые ссылается страница.

Эти селекторы не поддерживаются в IE 6 (и более младших версиях).

## **Лекция 1.6. Основы верстки документов**

Под версткой понимается размещение текста, графических и иных объектов на странице(ах) документа в соответствии с заданной разметкой или шаблоном.

Различают логическую и физическую страницы. Физическая страница (размер бумаги) обычно имеет некоторый стандартный размер, соответствующий размеру бумаги потребительского формата, например, формата А4: 210х297 мм. Логическая страница образуется на поле физической за вычетом установленных пользователем полей (Рисунок 6-1). Количество данных на логической странице определяется, с одной стороны, плотностью печати (количеством знаков на строке), а с другой – разреженностью строк (интервалом между строками).

Аналогично тому, как осуществляется жесткий (нажатием клавиши <Enter>) и мягкий (автоматический) перевод каретки на новую строку, текстовый процессор осуществляет мягкий и ***жесткий переход*** на новую страницу. ***Мягкий переход*** осуществляется автоматически после заполнения последней строки на странице. Однако пользователь может начать новую страницу, не дожидаясь ее оконча­тельного заполнения, подав команду жесткого перехода, например, при помощи сочетания клавиш <Ctrl>+<Enter>.

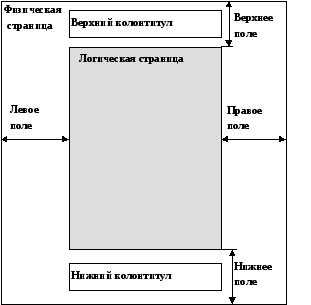


Рисунок 6-1. Параметры страницы

Существует также команда запрета разрыва страниц, которая используется, когда необходимо, чтобы определенная часть документа (например, таблица) находилась на одной странице. В этом случае следует поставить команду запрета разрыва страниц перед таблицей, чтобы предотвратить ее разрыв.

При разрыве абзаца многие текстовые процессоры обеспечивают контроль за так на­зываемыми висячими строками. Висячей строкойназывается первая строка или заголовок нового абзаца, оказавшийся на последней строке страницы или последняя строка абзаца, оказавшаяся в начале страницы. Размещение абзаца при его разрыве может регулироваться по-разному. Например, не менее двух строк в конце страницы и не менее трех строк в начале.

Для введения нумерации страниц в создаваемом документе текстовый процессор предложит специальное меню, в котором можно указать нужные усло­вия нумерации: месторасположение на листе номера страницы, отказ от нумерации первой страницы, использование колонтитулов и др. Номера страниц проставляются в колонтиту­ле.

***Колонтитулы*** представляют собой одну или несколько строк, помещаемых в на­чале или конце каждой страницы документа. Они обычно содержат номера страниц, назва­ние глав и параграфов, название и адрес фирмы и т.п. Колонтитулы могут различаться для четных и нечетных страниц, а также для первой страницы и последующих. Использование колонтитулов позволяет лучше ориентироваться в документе, а также использовать допол­нительные возможности рекламы.

***Использование шаблонов***

Для унификации структуры и внешнего вида документов используются стандарты. Инстру­ментами стандартизации документов, создаваемых в рамках текстовых процессоров, яв­ляются шаблоны. Шаблоны позволяют составлять и хранить универсальные бланки документов различного типа: писем, служебных записок, доверенностей и т.п. Начиная составлять определенный документ, пользователь сначала вызывает шаблон этого типа документов, а уже затем заполняет его. Составление документа при этом сводится к заполнению его определенных полей текстом. Один раз сделанный на основе стандартов шаблон может в дальнейшем многократно использоваться для создания документов опреде­ленного вида.

В [шаблонах](#Шаблоны) хранится не только информация, нужная для создания нового документа, но также и некоторые элементы среды текстового процессора: состав панелей инструментов и меню, набор макросов3, коэффициент масштабирова­ния при отображении документа и т.п.

Можно использовать шаблоны такими, какие они есть, изменять их или создать свои собственные шаблоны. Если при создании нового документа пользователь не выбрал другой шаблон, документ базируется на универсальном шаблоне Обычный (файл normal.dot).

Шаблон Обычный является шаблоном общего назначения для любых типов документов.

Формы предназначены для сокращения трудозатрат на оформление документов массового применения за счет предварительного создания шаблонов. Например, формы различных справок установленного образца, бланков документов, таблиц и др. Форма состоит из постоянной(неизменной) и переменной (изменяемой) при вводе данных части документа.

Обычно форма разрабатывается как шаблон, на базе которого создаются новые документы. Документ может содержать формы, защищенные от изменений.