Федеральное государственное бюджетное образовательное учреждение высшего образования

**Национальный исследовательский университет «МЭИ»**

Институт информационных и вычислительных технологий

Кафедра Вычислительных машин, систем и сетей

Курсовой проект:

Разработка учебного портала

по курсу Web-технологии

Задание выполнил: Челышев Э.А.

Студент группы A-08-17

Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2020

**Описание решаемой задачи**

Целью данного курсового проекта являлась разработка учебного портала по курсу Web-технологии, содержащего информацию по следующим темам:

*Язык разметки: понятие, классификация. GML. SGML. Язык гипертекстовой разметки HTML. XML. XHTML. HTML5. Структура тега. Структура документа. DTD.*

Разработанное приложение обладает возможностью редактирования, добавления и удаления информации.

Для разработки портала были использованы следующие инструменты:

* Веб-сервер Apache v.2.2.22 ;
* Приложение phpMyAdmin v.3.5.4;
* СУБД MySQL v.5.5.23;
* Приложение CKEditor v.4.14.0 с добавленным плагином CodeSnippet v.4.14.0;
* Библиотека highlight.js v.10.0.0.

**Перечень операционных систем в которых возможно использовать разрабатываемое приложение.**

Windows 7/8/10

Linux Ubuntu/Debian/

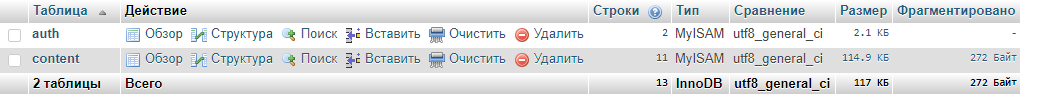
Android

iOS

**Этапы разработки**:

1. На первом этапе была разработана **структура базы данных** (см. ниже) и выделены отдельные страницы соответственно имеющимся темам.
2. На втором этапе был создан файл **content.php**, который является прототипом для всех остальных страниц сайта. Так же были созданы файлы **config.php** и **functions.php**, позволяющие осуществлять работу с базой данных.
3. На третьем этапе был подключен **CKEditor**.
4. На четвертом этапе был создан индексный файл, проработано меню сайта. Здесь и далее все заполнение сайта выполнено с использованием **CKEditor**.
5. Был подключен плагин **CodeSnippet** и скрипт **highlight.js** для подсветки кода.
6. Была добавлена **система авторизации**.
7. На последнем этапе велось наполнение сайта. Был создан файл стилей, который пополнялся параллельно с разработкой сайта.

**Описание структуры базы данных**

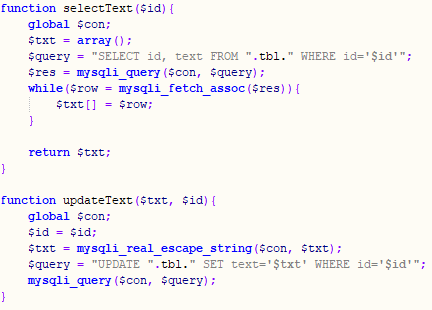
****

База данных curs содержит две таблицы: **content** со столбцами id и text с типами varchar и text соответственно. В этой таблице содержится наполнение сайта.

Во второй таблице **auth** содержится информация о пользователях: username и password. По умолчанию там содержатся два пользователя admin и stud. Пароли совпадают с именами пользователя.

**Описание разработки скриптов на PHP**

Для возможности редактирования информации сайта и ее представления на экране используются функции работы с БД:



Для авторизации использован механизм сессий. Пароли в БД хранятся в зашифрованном виде с использованием встроенной функции **md5**:



Учитывая тот факт, что сайт посвящен языкам разметки и содержит код, который не должен быть преобразован в сущности, при загрузке текста в CKEditor используется функция



А для того, чтобы текст корректно отображался на сайте:



**Учебно методический материал, размещенный на портале**

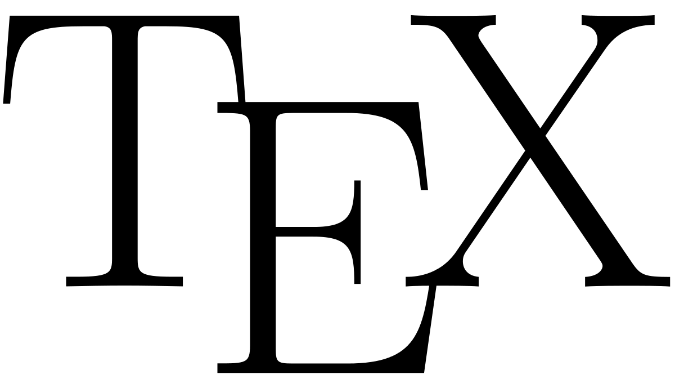
[](http://localhost/Chelyshev/image/markup_1.jpg)**Язы́к разме́тки** (текста) в компьютерной терминологии — набор символов или последовательностей, вставляемых в текст для передачи информации о его выводе или строении. Принадлежит классу *компьютерных языков*.

Текстовый документ, написанный с использованием языка разметки, содержит не только сам текст (как последовательность слов и знаков препинания), но и дополнительную информацию о различных его участках — например, указание на заголовки, выделения, списки и т. д. В более сложных случаях язык разметки позволяет вставлять в документ интерактивные элементы и содержание других документов.Следует отметить, что язык разметки обычно не считается языком программирования.

Различают *логическую и визуальную разметки*. В первом случае речь идёт только о том, какую роль играет данный участок документа в его общей структуре (например, «данная строка является заголовком»). Во втором — определяется, как именно будет отображаться этот элемент (например, «данную строку следует отображать жирным шрифтом»).

**Идея языков разметки состоит в том, что визуальное отображение документа должно автоматически получаться из логической разметки и не зависеть от его непосредственного содержания.**Это упрощает автоматическую обработку документа и его отображение в различных условиях (например, один и тот же файл может по-разному отображаться на экране компьютера, мобильного телефона и на печати, поскольку свойства этих устройств вывода существенно различаются).

**Примеры языков разметки**

[](http://localhost/Chelyshev/image/markup_2.png)Языки разметки используются везде, где требуется получение форматированного текста на основании неформатированного:

* в типографии ([GML](http://localhost/Chelyshev/gml.php), [SGML](http://localhost/Chelyshev/gml.php#SGML), TeX, PostScript, RTF),
* пользовательских интерфейсах компьютеров (Microsoft Word, OpenOffice, troff),
* Всемирной Сети ([HTML](http://localhost/Chelyshev/html.php), [XHTML](http://localhost/Chelyshev/xhtml.php), [XML](http://localhost/Chelyshev/xml.php), [HTML5](http://localhost/Chelyshev/html5.php), WML, VML, PGML, SVG, XBRL).

Для написания исходных текстов Википедии её участники используют особый язык разметки, а для отображения сложных математических формул — язык TeX.

**GML**(англ. **Generalized Markup Language**— обобщённый язык разметки) представляет собой набор макросов, основной целью которых является реализация разметки текста на IBM, используя процедурную разметку SCRIPT. SCRIPT/VS — это основной компонент DCF (англ. Document Composition Facility — оборудование для составления документов) в IBM. Стартовый набор тегов в GML поставляется с продуктом DCF.

GML стал одним из первых созданных языков разметки. Он был разработан в конце 1960-х годов тремя научными сотрудниками американской [компании IBM](https://www.ibm.com/ru-ru): Чарльзом Голдфарбом, Эдвардом Мошером и Рэймондом Лори. Название языка выбрано отчасти потому, что аббревиатуру GML образуют первые буквы фамилий разработчиков.

Создателями языка были сформулированы **три основных принципа языка разметки GML**:

1. **Использование единых принципов форматирования** во всех программах, выполняющих обработку документов. Наличие единого набора синтаксических конструкций и общей семантики заметно упрощает взаимодействие между программами.
2. **Специализация языков форматирования**. Благодаря возможности построения специализированного языка на базе набора стандартных правил программист перестает зависеть от внешних реализаций и их представлений о потребностях конечного пользователя
3. **Четкое определение формата документа**. Правила, определяющие формат документа, задают количество и маркировку языковых конструкций, используемых в документе. Применение стандартного формата гарантирует, что пользователь будет точно знать структуру содержимого документа. Обратите внимание: речь идет не о формате отображения документа, а о его структурном формате. Набор правил, описывающих этот формат, называется «определением типа документа» (document type definition, [DTD](http://localhost/Chelyshev/dtd.php)).

GML поддерживает редактирование и форматирование текстов, позволяет подсистемам извлечения информации обмениваться документами. В то же время вместо простой схемы разметки GML представляет концепцию формально определенных типов документов, содержащих строгую иерархию структурированных элементов. До 90% всех документов компании IBM созданы с использованием этого языка разметки.

При использовании GML документ помечается тегами, которые определяют, чем является текст с точки зрения параграфов, заголовков, списков, таблиц. Такой документ может быть автоматически отформатирован для различных устройств, остается только указать разновидность устройства. Например, можно форматировать документ для лазерного принтера или матричного или просто вывести на экран, всего лишь указав профиль оборудования, без изменения самого документа.

Рассмотрим пример GML-кода:

:h1.Chapter 1:  Introduction

:p.GML supported hierarchical containers, such as

:ol

:li.Ordered lists (like this one),

:li.Unordered lists, and

:li.Definition lists

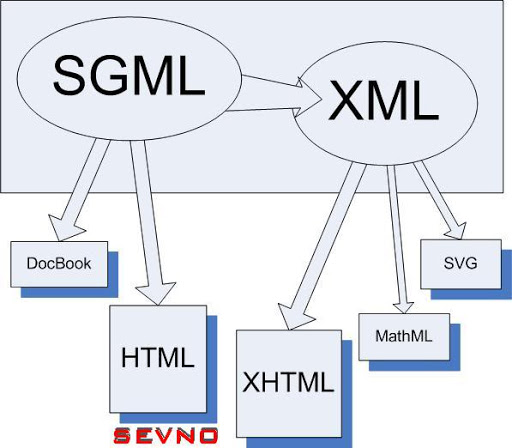
:eol.

as well as simple structures.

:p.Markup minimization (later generalized and formalized in SGML),

allowed the end-tags to be omitted for the "h1" and "p" elements.

Любой человек, так или иначе знакомый с популярным языком разметки [HTML](http://localhost/Chelyshev/html.php), способен без труда определить смысл использованных в этом коде тэгов: **:h1**, **:p**, **:ol**, **:li**. Примерный результат выполнения данного кода можно посмотреть [здесь](http://localhost/Chelyshev/ex_gml.html).

[](http://localhost/Chelyshev/image/diag.jpg)Исследования и разработка GML продолжались около десяти лет, пока в результате соглашения, заключенного международной группой разработчиков, не появился стандарт **SGML** (англ. Standard Generalized Markup Language — стандартный обобщённый язык разметки), который является [ISO](https://www.iso.org/ru/home.html)-стандартом технологии для определения обобщенного языка разметки для документов.

**SGML**- это метаязык для создания других языков разметки документов. Тем не менее, он был мало известен до 1990 года, когда Тим Бернерc-Ли, автор идеи World Wide Web, создал Hypertext Markup Language, который является подмножеством SGML. Вскоре после этого все виды документов и данных в начале и в конце каждого текстового элемента стали содержать теги в стиле <тег>. В конце 1990-х годов появился Extensible Markup Language ([**XML**](http://localhost/Chelyshev/xml.php))... и вселенная ИТ необратимо изменилась.

[](http://localhost/Chelyshev/image/html.jpeg)Язык разметки, определяемый с помощью SGML (приложение SGML), состоит из SGML-декларации, определения типа документа (Document Type Definition, DTD) и семантического описания, дополняющего определение типа документа. В SGML-декларации указывается, какие символы и разделители могут отображаться в данном типе документов. Термин DTD, если вы работали с XML, должен быть вам знаком. Определение типа документа содержит в себе описание синтаксиса языка разметки.

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык разметки документов во Всемирной паутине.

Большинство веб-страниц содержат описание разметки на языке HTML (или [XHTML](http://localhost/Chelyshev/xhtml.php)). Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Язык HTML был создан британским ученым Тимом Бернерсом-Ли, создателем World Wide Web, приблизительно в 1986—1991 годах в стенах [ЦЕРНа](https://home.cern/" \t "_blank)в Женеве в Швейцарии для передачи гипертекстовых документов через web-браузер. HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. HTML успешно справлялся с проблемой сложности [SGML](http://localhost/Chelyshev/gml.php) путём определения небольшого набора структурных и семантических элементов — дескрипторов. Дескрипторы также часто называют «тегами». С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Помимо упрощения структуры документа, в HTML внесена поддержка гипертекста. Мультимедийные возможности были добавлены позже. Язык HTML является подмножеством языка SGML (стандартного обобщённого языка разметки по стандарту [ISO](https://www.iso.org/ru/home.html) 8879) и вплоть до 5-й версии определялся как его приложение. Начиная с HTML5 спецификации языка формулируются в терминах [DOM](http://localhost/Chelyshev/html5.php#DOM)(объектной модели документа).

Первым общедоступным описанием HTML был документ «[Теги HTML](https://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html)», впервые упомянутый в Интернете Тимом Бернерсом-Ли в конце 1991 года. В нём описываются 18 элементов, составляющих первоначальный, относительно простой дизайн HTML. За исключением тега гиперссылки, на них сильно повлиял SGMLguid, внутренний формат документации, основанный на стандартном обобщенном языке разметки (SGML), в CERN. Одиннадцать из этих элементов всё ещё существуют в HTML.

Официальной спецификации HTML 1.0 не существует. До 1995 года существовало множество неофициальных стандартов HTML. Это привело к так называемой «войне браузеров», в ходе которой разработчики браузеров начали создавать свои собственные усовершенствования языка HTML. Эти модификации существенно отклонялись от идеи работы с единым стандартом HTML и вызвали настоящий хаос среди разработчиков, которые хотели создавать web-сайты, не зависящие от браузера. Более того, долгий период неопределенности в области стандартов привел к тому, что разработчики вывели язык из первоначально задуманных границ. Чтобы стандартная версия отличалась от них, ей сразу присвоили второй номер.

[](http://localhost/Chelyshev/image/html5.png)Версия 3 была предложена [Консорциумом Всемирной паутины](https://w3c.org.ru/index.html) (W3C) в марте 1995 года и обеспечивала много новых возможностей, таких как создание таблиц, «обтекание» изображений текстом и отображение сложных математических формул, поддержка gif-формата. Даже при том, что этот стандарт был совместим со второй версией, реализация его была сложна для браузеров того времени. Версия 3.1 официально никогда не предлагалась, и следующей версией стандарта HTML стала 3.2, в которой были опущены многие нововведения версии 3.0, но добавлены нестандартные элементы, поддерживаемые браузерами Netscape Navigator и Mosaic.

В версии HTML 4.0 произошла некоторая «очистка» стандарта. Многие элементы были отмечены как устаревшие и не рекомендованные. В частности, тег , используемый для изменения свойств шрифта, был помечен как устаревший (вместо него рекомендуется использовать таблицы стилей CSS).

В настоящее время Консорциум Всемирной паутины разработал [HTML версии 5](http://localhost/Chelyshev/html5.php). В соответствующем разделе нашего сайта вы найдете больше информации об HTML5.

Рассмотрим некоторые основные теги языка HTML.

| **Тег** | **Описание** |
| --- | --- |
| **<html></html>** | Тег **<html>** является контейнером, который заключает в себе все содержимое веб-страницы, включая теги **<head>** и **<body>**. Открывающий и закрывающий теги **<html>** в документе необязательны, но хороший стиль диктует непременное их использование. Как правило, тег**<html>** идет в документе вторым, после определения типа документа (Document Type Definition, DTD), устанавливаемого через элемент **<!DOCTYPE>**. Закрывающий тег **<html>** должен всегда стоять в документе последним. |
| **<head></head>** | Тег **<head>** предназначен для хранения других элементов, цель которых — помочь браузеру в работе с данными. Также внутри контейнера**<head>** находятся метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. |
| **<title></title>** | Определяет заголовок документа. Элемент **<title>** не является частью документа и не показывается напрямую на веб-странице. В операционной системе Windows текст заголовка отображается в левом верхнем углу окна браузера (рис. 1). Допускается использовать только один тег **<title>** на документ и размещать его в контейнере **<head>**. |
| **<body></body>** | Элемент **<body>** предназначен для хранения содержания веб-страницы (контента), отображаемого в окне браузера. Информацию, которую следует выводить в документе, следует располагать именно внутри контейнера **<body>**. К такой информации относится текст, изображения, теги, скрипты JavaScript и т.д. |
| **<p></p**> | Определяет текстовый абзац. Тег **<p>** является блочным элементом, всегда начинается с новой строки, абзацы текста идущие друг за другом разделяются между собой отбивкой. Величиной отбивки можно управлять с помощью стилей. Если закрывающего тега нет, считается, что конец абзаца совпадает с началом следующего блочного элемента. |
| **<h№></h№>**, где № - цифра от 1 до 6 | HTML предлагает шесть заголовков разного уровня, которые показывают относительную важность секции, расположенной после заголовка. Так, тег **<h1>** представляет собой наиболее важный заголовок первого уровня, а тег **<h6>** служит для обозначения заголовка шестого уровня и является наименее значительным. По умолчанию, заголовок первого уровня отображается самым крупным шрифтом жирного начертания, заголовки последующего уровня по размеру меньше. Теги **<h1>**,...,**<h6>** относятся к блочным элементам, они всегда начинаются с новой строки, а после них другие элементы отображаются на следующей строке. Кроме того, перед заголовком и после него добавляется пустое пространство. |
| **<br>** | Тег **<br>** устанавливает перевод строки в том месте, где этот тег находится. В отличие от тега абзаца **<p>**, использование тега **<br>** не добавляет пустой отступ перед строкой. Если текст, в котором используется перевод строки, обтекает плавающий элемент, то с помощью атрибута clear тега **<br>** можно сделать так, чтобы следующая строка начиналась ниже элемента. |
| **<hr>** | Рисует горизонтальную линию, которая по своему виду зависит от используемых параметров, а также браузера. Тег **<hr>** относится к блочным элементам, линия всегда начинается с новой строки, а после нее все элементы отображаются на следующей строке. |
| **<b></b>** | Устанавливает жирное начертание шрифта. Допустимо использовать этот тег совместно с другими тегами, которые определяют начертание текста. |
| **<i></i>** | Устанавливает курсивное начертание шрифта. Допустимо использовать этот тег совместно с другими тегами, которые определяют начертание текста. |
| **<div></div>** | Элемент **<div>** является блочным элементом и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей. Чтобы не описывать каждый раз стиль внутри тега, можно выделить стиль во внешнюю таблицу стилей, а для тега добавить атрибут class или id с именем селектора. |
| **<ul><li></ul>** | Тег **<ul>** устанавливает маркированный список. Каждый элемент списка должен начинаться с тега **<li>**. Если к тегу **<ul>** применяется таблица стилей, то элементы **<li>** наследуют эти свойства. |
| **<ol><li></ol>** | Тег **<ol>** устанавливает нумерованный список. Каждый элемент списка должен начинаться с тега **<li>**. Если к тегу **<ol>** применяется таблица стилей, то элементы **<li>** наследуют эти свойства. |
| **<table></table>** | Элемент **<table>** служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов **<tr>** и **<td>**. Внутри **<table>** допустимо использовать следующие элементы: **<caption>**, **<col>**, **<colgroup>**, **<tbody>**, **<td>**, **<tfoot>**, **<th>**, **<thead>**и **<tr>**. |

Рассмотрим пример HTML-кода, результат выполнения которого можно посмотреть [здесь](http://localhost/Chelyshev/ex_html.html).

<!DOCTYPE html>

<html>

   <head>

      <meta charset="utf-8" />

      <title>HTML Document</title>

   </head>

   <body>

      <p>

         <b>

            Этот текст будет полужирным, <i>а этот — ещё и курсивным</i>.

         </b>

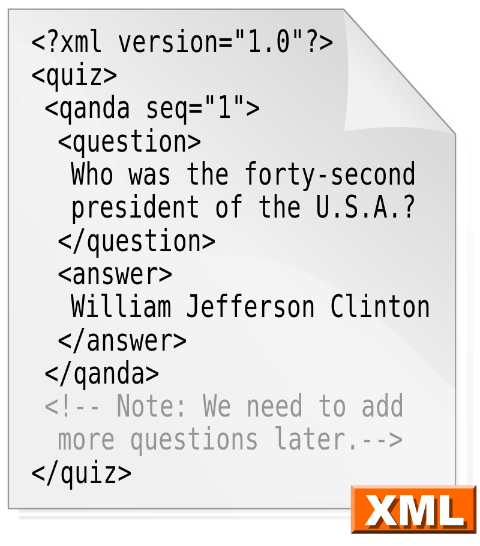
      </p>

   </body>

</html>

Конечно, данный список не является конечным: этот портал не ставит своей целью подробное описание спецификации языка HTML, а является лишь кратким обзором. Более того, практически каждый из приведенных выше тегов имеет атрибуты, которые разработчик может задавать, тем самым изменяя внешний вид и функционал страницы. Вот ссылки на книги и интернет-ресурсы, посвященные языку HTML:

* [**htmlbook.ru**](http://htmlbook.ru/)
* [**HTMLBASE**](https://htmlbase.ru/)
* [**HTML5Book**](https://html5book.ru/osnovy-html/)
* **Б. Лоусон, Р. Шарп. — Изучаем HTML 5.**
* **П. Лабберс. — HTML 5 для профессионалов.**
* **Ч. Муссиано, Б Кеннеди. — HTML & XHTML. Подробное руководство.**
* **Бен Хеник. — HTML и CSS Путь к совершенству.**
* **Дронов. В. — HTML.5.CSS.3.и.Web 2.0.Разработка.современных.Web-сайтов.**

[](http://localhost/Chelyshev/image/xml_1.png)Спецификация **XML** (Extensible Markup Language) была разработана в 1996 году Рабочей группой [консорциума Всемирной паутины](https://w3c.org.ru/index.html) (World Wide Web Consortium (W3C)) и впоследствии предложена в качестве рекомендации. По своему замыслу технология XML должна обеспечить *отделение информации от разметки, что позволяет производить обработку, поиск и представление информации на более высоком технологическом уровне*. Отметим, что спецификация XML является основой для построения грамматики языков разметки, и лишь условно сам XML можно назвать языком разметки. Правильнее сказать, что XML, как и [**SGML**](http://localhost/Chelyshev/gml.php), не является языком; он также представляет собой набор рекомендаций, на базе которых создаются другие языки. Точнее говоря, XML является подмножеством языка SGML и представляет из себя конгломерат из трех отдельных спецификаций:

1. **XML**(Extensible Markup Language) — спецификация, определяющая базовый синтаксис XML;

2. **XSL**(Extensible Style Language) — спецификация, направленная на отделение визуального оформления страницы от ее содержимого за счет применения к документу стилей (style sheets), определяющих конкретные атрибуты форматирования;

3. **XLL**(Extensible Linking Language) — спецификация, определяющая представление ссылок на другие ресурсы.

На основе этой технологии обработка, поиск и представление информации переходят на совершенно иной уровень. Первоначальный веб был ориентирован на работу человека, но веб следующего поколения (семантический веб) должен в значительной мере опираться на *машинную обработку информации*, стандарту XML при этом отводится роль одной из ключевых технологий. Синтаксис XML построен на основе тегов, но в отличие от HTML, в котором множество тегов фиксировано, в рамках XML *пользователь создает собственное множество тегов и задает структурные отношения между ними*.

В настоящее время создано большое число языков разметки, являющихся подмножествами XML, как пример укажем язык беспроводной разметки **WML** (Wireless Markup Language ), программная поддержка которого встроена во все мобильные телефоны и коммуникационные устройства, заявленные как WAP–совместимые.

**Правильно построенные XML-документы**

При создании документов необходимо учитывать правила, основная задача которых – в отделении данных и формата. Эти правила определены в рекомендации консорциума W3C и состоят из следующих требований:

* документы XML должны начинаться с объявления XML, в котором определяется версия XML, например, (это объявление является инструкцией приложению, обрабатывающему документ, в частности, браузеру);
* любому открывающему тегу должен соответствовать закрывающий тег;
* в XML учитывается регистр символов;
* [](http://localhost/Chelyshev/image/xml_2.jpg)значения атрибутов, используемых в определении тегов, должны заключаться в кавычки;
* в любом XML-документе должен содержаться один корневой элемент для всего документа;
* элементы не должны перекрываться;
* вся информация, размещенная между открывающим и закрывающим тегами, рассматривается как данные, при этом учитываются все символы форматирования (пробелы, конец строки, табуляция);
* в XML имеются зарезервированные символы, которые используются как элементы синтаксиса;
* в тексте эти символы нужно заменять последовательностями других символов, называемых объектами (entities; в русскоязычной литературе часто используют термин сущности).

**Зарезервированные символы и объекты**

|  |  |
| --- | --- |
| **Зарезервированный**  **символ** | **Объект** |
| < | < |
| > | > |
| & | & |
| " | " |
| ' | &apos |

**Если перечисленные правила выполнены, то говорят, что документ правильно построен (well-formed), и только в этом случае он является документом XML.** Такой документ будет правильно отображен в браузере, в ином случае будет выдано сообщение об ошибке. Для обработки документов XML используется программный модуль, называемый *XML-процессором.* Поддержка XML браузером зависит от того, включен ли в него XML-процессор и какие возможности этот процессор имеет. Например, браузеры Mozilla и Firefox поддерживают почти все технологии, основанные на XML.

Рассмотрим небольшой пример: простейшее описание электронного журнала. Данный можно набрать в любом текстовом редакторе. При просмотре этого файла в каком-либо браузере будет выведено представление XML-документа в виде дерева (посмотреть можно [здесь](http://localhost/Chelyshev/ex_xml.xml)).

<?xml version="1.0" encoding="windows-1251" standalone="yes"?>

<!-- XML пример -->

<journal>

<title>Lobachevsky&apos;Journal</title>

9

<contacts>

<address>Kazan State University</address>

<url>ljm.ksu.ru</url>

</contacts>

<articles>

<article ID="1">

<title>MathML and TeX</title>

<author>M. Malakhaltsev</author>

</article>

<article ID="2">

<title>MathML and RDF</title>

<author>E. Lipachev</author>

</article>

</articles>

</journal>

Все документы XML начинаются с **пролога,** в котором сообщается, что документ написан на XML, а также указывает, какая версия XML при этом использовалась. Атрибут **version** является обязательным и указывает на версию XML, применяемую для структурирования. Например, стандартный XML-файл начинается строкой:

<?xml version="1.0"?>

Получая документ, версию которого он не поддерживает, XML-процессор должен выдать соответствующие сообщения. Пролог может содержать другие инструкции. Например, объявление можно расширить, указав, что документ является автономным:

<?xml version="1.0" standalone="yes"?>

Присваивание yes атрибуту **standalone** сообщает механизму обработки XML-кода о том, что документ не импортирует других файлов (например, [DTD](http://localhost/Chelyshev/dtd.php)). Атрибут **encoding** определяет кодировку документа, и его желательно указывать. Например, если документ содержит символы в одной из кодировок кириллицы (к примеру, CP1251, KOI8-R, CP866, UTF-8), в отсутствие этого атрибута XML-процессор выдаст сообщение об ошибке и прекратит обработку. Во второй строке показано, как добавить комментарии, – правила те же, что и в HTML.

В последующих строках объявлены элементы. Элемент состоит из открывающего и закрывающего тегов, а также данных между этими тегами. Пустой элемент, например **<br></br>**, может записываться в виде **<br/>**. Имена не могут содержать пробелы, кроме того, имеет значение регистр символов. Одни элементы могут быть вложены в другие, как, например, элемент **<author></author>** из приведенного примера вложен в элемент **<article></article>**, который в свою очередь вложен в элемент **<articles></articles>**. Непустые элементы должны содержать как открывающий, так и закрывающий тег. В элементах, которые логически не могут иметь закрывающего тега, используется альтернативная форма синтаксиса **<элемент/>**. Один из элементов документа должен содержать все остальные, этот элемент называют корневым элементом или элементом документа. В нашем примере таким элементом является **<journal></journal>**. Таким образом, элементы документа должны образовывать древовидную структуру.

**Теги XML могут обладать атрибутами**. Атрибуты содержат дополнительную информацию о содержании, которая в дальнейшем используется при форматировании или обработке XML. Значения атрибутов присваиваются в формате «имя=значение», атрибуты XML должны быть заключены в апострофы или кавычки. В приведенном примере элемент **<article></article>** содержит атрибут **ID**, с помощью которого каждой статье присваивается числовой идентификатор.

Вот ссылки на некоторые учебные пособия по языку XML:

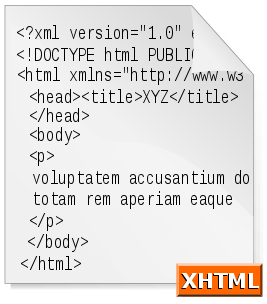
* Шеперд Д. - Освой самостоятельно XML за 21 день;
* Маршал Б. - XML в действии;
* Штайнер Гюнтер - HTML/XML/CSS справочник;
* Мартин Д. и др. - XML для профессионалов;
* [<MyRusakov.ru>](https://myrusakov.ru/xml.html).

**XHTML** (англ. Extensible Hypertext Markup Language — расширяемый язык гипертекстовой разметки) — семейство языков разметки веб-страниц на основе [XML](http://localhost/Chelyshev/xml.php), повторяющих и расширяющих возможности [HTML](http://localhost/Chelyshev/html.php) 4. Спецификации XHTML 1.0 и XHTML 1.1 являются рекомендациями [консорциума Всемирной паутины](https://w3c.org.ru/index.html). Развитие XHTML остановлено; новые версии XHTML не выпускаются.  
*Главное отличие XHTML от HTML заключается в обработке документа*. Документы XHTML обрабатываются своим модулем (парсером) аналогично документам XML. В процессе этой обработки ошибки, допущенные разработчиками, не исправляются. Если HTML "закрывает глаза" на мелкие огрехи разработчика, то XHTML будет выдавать сообщение о том, что документ неверно сформирован до тех пор, пока разработчик не исправит все ошибки и недочеты.

XHTML соответствует спецификации SGML, поскольку XML является его подмножеством. HTML же, в свою очередь, обладает множеством особенностей в процессе обработки и фактически перестал относиться к семейству SGML, что и закреплено в спецификации HTML 5 (она сформулирована в терминах DOM).

Согласно синтаксису XHTML:

* все элементы должны быть закрыты. Теги, которые не имеют закрывающего тега (например, **<img>** или **<br>**), должны иметь на конце / (например, **<br />**);
* логические атрибуты записываются в развёрнутой форме. Например, следует писать **<option selected="selected">** или **<td nowrap="nowrap">**;
* имена тегов и атрибутов должны быть записаны строчными буквами (например, **<img alt="" />** вместо <IMG ALT="" />);
* XHTML гораздо строже относится к ошибкам в коде; По рекомендации консорциума Всемирной паутины браузеры, встретив ошибку в XHTML, должны сообщить о ней и не обрабатывать документ. Для HTML браузеры должны были попытаться понять, что хотел сказать автор.
* < и & везде, даже в URL, должны замещаться **&lt;**и **&amp;** соответственно. ;
* кодировкой по умолчанию является **UTF-8** (в отличие от HTML, где кодировкой по умолчанию является ISO 8859-1).

[](http://localhost/Chelyshev/image/xhtml_1.png)Существует три типа документов XHTML: **strict**, **transitional**и **frameset**. Наиболее употребительной и универсальной из версий XHTML является **переходная**(англ. transitional), поскольку она позволяет использовать **iframe**(включение содержимого одной веб-страницы в другую) и атрибут **target**у ссылок (для указания того, например, что ссылке необходимо открываться в новом окне). **Фреймовая**версия (англ. frameset) представляет собой расширенный вариант transitional и добавляет к нему, как следует из названия, возможность установки **frameset**вместо **body**. **DTD**(англ. document type definition - определение типа документа) **строгой**версии XHTML (англ. strict) не содержит многих тегов и атрибутов, описанных в DTD transitional и признанных устаревшими.

Валидным (то есть отвечающим всем правилам) XHTML-документом считается документ, удовлетворяющий спецификации. В идеале все браузеры должны следовать веб-стандартам и, в соответствии с ними, валидные документы должны отображаться во всех браузерах на всех платформах. Валидация XHTML-документа рекомендована даже несмотря на то, что она не гарантирует кросс-браузерности. Соответствие документа спецификации может быть проверено на сайте [validator.w3.org](http://validator.w3.org/). При валидации будут обнаружены ошибки в XHTML-разметке; валидатор разъяснит суть ошибок.

**Самыми распространёнными ошибками в XHTML-разметке являются**:

* незакрытые элементы (XHTML, в отличие от HTML, требует закрытия всех элементов, в том числе не имеющих закрывающего тега, как, например, **<br />**);
* отсутствие альтернативных текстов для изображений (достигающихся применением атрибута alt, который помогает сделать документы доступнее для устройств, которые не в состоянии отображать изображения, или предназначенных для слабовидящих людей);
* присутствие текста непосредственно в теге **<body>** документа (должен быть объявлен блочный элемент, внутрь которого следует помещать содержимое);
* вложение блочных элементов внутрь внутристрочных (inline) (например, блочные элементы **<div>** или **<p>** не могут быть вложены внутрь инлайновых элементов <a>, <span>, <em> и так далее);
* пренебрежение заключением значений атрибутов в кавычки (<a href=http://example.com/> вместо <a href="http://example.com/">);
* неправильное вложение элементов (конструкции вида **<strong><em> </strong></em>**);
* неправильное использование ссылок-мнемоник (например, & вместо **&amp;**);
* написание тегов и/или атрибутов прописными буквами вместо строчных (<DIV STYLE="…"> вместо **<div style="...">**);
* задание в теге **<!DOCTYPE …>** относительного пути к DTD-файлу ("DTD/xhtml11.dtd" вместо "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd").

Этот список не полный и содержит самые частодопускаемые ошибки при составлении XHTML-документов.

Рассмотрим пример верстки страницы на языке XHTML (результат исполнения можно посмотреть [здесь](http://localhost/Chelyshev/ex_xhtml.xhtml)):

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />

<title>XHTML верстка</title>

</head>

<body style="background-color:#ffffcc; color:#008800">

<br />

<h2 align="center">Пробная XHTML страничка</h2>

<br />

<div align="center">

<img src="/image/xhtml\_1.png" width="400" height="250" alt="XHTML" vspace="20" />

</div>

<p align="center" style="font-size:17px">Среди полей и диких трав,<br />

Жужжа на встречу ветру,<br />

Она летит, свой дикий нрав<br />

Вплетая в песню лета.</p>

<p align="center"><em>Эргэл</em></p>

<br />

<hr />

</body>

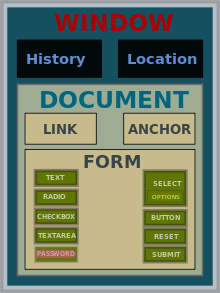
</html>

В самом начале данного кода мы видим тег **<!DOCTYPE>**- это как раз и есть определение типа документа ([DTD](http://localhost/Chelyshev/dtd.php)).

**HTML5**(англ. HyperText Markup Language, version 5) — язык для структурирования и представления содержимого всемирной паутины. Это пятая версия [HTML](http://localhost/Chelyshev/html.php). Хотя стандарт был завершён (рекомендованная версия к использованию) только в 2014 году, уже с 2013 года браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта (англ. HTML Living Standard). Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров.

Во всемирной паутине долгое время использовались стандарты HTML 4.01, [XHTML](http://localhost/Chelyshev/xhtml.php)1.0 и XHTML 1.1. Веб-страницы на практике оказывались свёрстаны с использованием смеси особенностей, представленных различными спецификациями, включая спецификации программных продуктов, например веб-браузеров, а также сложившихся общеупотребительных приёмов. HTML5 был создан как единый язык разметки, который мог бы сочетать синтаксические нормы HTML и XHTML. *Он расширяет, улучшает и рационализирует разметку документов.*

В HTML5 реализовано множество новых синтаксических особенностей. Например, элементы **<video>**, **<audio>** и **<canvas>**, а также возможность использования **SVG** и математических формул. Эти новшества разработаны для упрощения создания и управления графическими и мультимедийными объектами в сети без необходимости использования сторонних API и плагинов. Другие новые элементы, такие как **<section>**, **<article>**,**<header>** и **<nav>**, разработаны для того, чтобы обогащать семантическое содержимое документа (страницы). Новые атрибуты были введены с той же целью, хотя ряд элементов и атрибутов был удалён. Некоторые элементы, например **<a>**, **<menu>** и **<cite>**, были изменены, переопределены или стандартизированы. Спецификация языка сформулирована в терминах **DOM**.

**[](http://localhost/Chelyshev/image/dom.png)DOM** (от англ. Document Object Model — «объектная модель документа») — это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

*Модель DOM не накладывает ограничений на структуру документа*. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями «родительский-дочерний». На рисунке представлен пример иерархии DOM.

Изначально различные браузеры имели собственные модели документов (DOM), несовместимые с остальными. Для обеспечения взаимной и обратной совместимости специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.

**Разметка**

HTML5 вводит несколько новых элементов и атрибутов, которые отражают типичное использование разметки на современных веб-сайтах. Некоторые из них — семантические замены для использования универсальных блочных (**<div>**) и строчных (**<span>**) элементов, например, **<nav>** (блок навигации по сайту), **<footer>** (обычно относится к нижней части страницы или последней строке HTML кода) или **<audio>** и **<video>** вместо **<object>**. Некоторые устаревшие элементы, которые можно было использовать в HTML 4.01, были исключены, включая чисто оформительские элементы, такие как **<font>** и **<center>**, чьи эффекты выполняются с помощью каскадных таблиц стилей. Также в поведении гипертекстовых страниц снова заострено внимание на важности скриптов DOM (например, **Javascript**).

Синтаксис HTML5 больше не базируется на SGML, несмотря на подобие его разметки. Однако он был разработан обратно совместимым с обычным парсингом более старых версий HTML. В HTML5 применяется новая вводная строка, которая выглядит как объявление типа документа в SGML, <!DOCTYPE html>, запускающая соответствующий стандартам режим визуализации.

**Новые API**

**API**(программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface) — описание способов (набор классов, процедур, функций, структур или констант.

Возможности DOM в HTML5 расширены и фактически используемые свойства задокументированы. Также добавлены новые API, которыми одна компьютерная программа может взаимодействовать с другой программой), например:

* элемент холст (**canvas**) для непосредственного метода рисования в 2D;
* контроль над проигрыванием медиафайлов, который может использоваться, например, для синхронизации субтитров с видео;
* хранение данных в браузере;
* File API: возможность загрузки документа через выбор (тег **<input type="file">**) или перетаскиванием (Drag-and-drop);
* Drag-and-drop: предоставляет набор событий для каждого элемента DOM, таких как появление и нахождение в его зоне, благодаря которым разработчик может информировать пользователя о необходимых действиях и идентификаторе перетаскиваемого файла, содержащего адрес, имя, тип, размер и дату изменения;
* управление историей браузера;
* и другие.

Рассмотрим пример кода на HTML5, результат выполнения которого можно посмотреть [здесь](http://localhost/Chelyshev/ex_html5.html).

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>

(Это title) Пример страницы на HTML5

</title>

</head>

<body>

<header>

<hgroup>

<h1>

Заголовок "h1" из hgroup

</h1>

<h2>

Заголовок "h2" из hgroup

</h2>

</hgroup>

</header>

<nav>

<menu>

<li>

<a href="link1.html">

Первая ссылка из блока "nav"

</a>

</li>

<li>

<a href="link2.html">

Вторая ссылка из блока "nav"

</a>

</li>

</menu>

</nav>

<section>

<article>

<h1>

Заголовок статьи из блока "article"

</h1>

<p>

Текст абзаца статьи из блока "article"

</p>

<details>

<summary>

Блок "details", текст тега "summary"

</summary>

<p>

Абзац из блока "details"

</p>

</details>

</article>

</section>

<footer>

<time>

Содержимое тега "time" блока "footer"

</time>

<p>

Содержимое абзаца из блока "footer"

</p>

</footer>

</body>

</html>

Предполагается, что читатель уже знаком с языком HTML, поэтому изучение данного кода не должно вызвать затруднений. Поясним лишь некоторые теги:

| **Тег** | **Описание** |
| --- | --- |
| **<hgroup></hgroup>** | Используется для группирования заголовков веб-страницы или раздела. Внутри располагаются теги заголовков от **<h1>** до **<h6>**. Данный тег исключён из версии HTML от W3C, но остался в версии WHATWG, будущее элемента пока не определено. |
| **<header></header>** | Тег <header> задает «шапку» сайта или раздела, в которой обычно располагается заголовок.Тег **<header>** задает «шапку» сайта или раздела, в которой обычно располагается заголовок. |
| **<nav></nav>** | Тег **<nav>** задает навигацию по сайту. Если на странице несколько блоков ссылок, то в **<nav>** обычно помещают приоритетные ссылки. Также допустимо использовать несколько тегов**<nav>** в документе. Запрещается вкладывать **<nav>** внутрь **<address>**. |
| **<menu></menu>** | Тег **<menu>** предназначен для отображения списка пунктов меню. Аналогично тегам **<ol>** и **<ul>** внутри контейнера **<menu>** список формируется с помощью тегов **<li>**. |
| **<section></section>** | Задаёт раздел документа, может применяться для блока новостей, контактной информации, глав текста, вкладок в диалоговом окне и др. Раздел обычно содержит заголовок. Допускается вкладывать один тег **<section>** внутрь другого. |
| **<article></article>** | Тег **<article>** задает содержание сайта вроде новости, статьи, записи блога, форума или др. |
| **<details></details>** | Тег **<details>** используется для хранения информации, которую можно скрыть или показать по требованию пользователя. По умолчанию содержимое тега не отображается, для изменения статуса применяется атрибут open. |
| **<summary></summary>** | Указывает заголовок для тега **<details>**, по которому можно щелкать для разворачивания/сворачивания информации. Тег **<summary>** должен идти первым внутри **<details>**. |
| **<footer></footer>** | Тег **<footer>** задаёт «подвал» сайта или раздела, в нём может располагаться имя автора, дата документа, контактная и правовая информация. |
| **<time></time>** | Помечает текст внутри тега **<time>** как дата, время или оба значения. Может указываться непосредственно внутри контейнера **<time>**, либо задаваться через атрибут datetime. |

**Тег**(иногда тэг, англ. **tag**— именованная метка, читается; более правильное название — **дескриптор**). В [SGML](http://localhost/Chelyshev/gml.php)([HTML](http://localhost/Chelyshev/html.php), WML, AmigaGuide, языках семейства [XML](http://localhost/Chelyshev/xml.php)) — элемент языка разметки гипертекста. Текст, содержащийся между начальным и конечным тегом, **отображается и размещается в соответствии со свойствами, указанными в начальном теге**.

Обычно используются парные теги — открывающий, или начальный, и закрывающий, или конечный. Возможно также применение одиночного тега. Например, тег отступа абзаца может оформлять абзац между тегами **<p>** и **</p>**, а может в одиночной форме до следующего первого попавшегося тега **<p>**.

Тег с пустым текстом: **<HR></HR>** вставляет разрыв текста без сохранения отступов, можно также использовать специальную форму записи тега — **<HR/>**.  
Набор и рекомендуемые интерпретации тегов определены организацией [W3C](http://w3c.org.ru/index.html).

В SGML можно назначать другие символы для обрамления тега (например, фигурные скобки), но в некоторых подмножествах этого языка (например, HTML и XML) такая возможность отсутствует.

Кроме того, существуют разные системы языков-подмножеств с меньшими возможностями. Например, на веб-форумах и досках объявлений используется язык разметки **BBCode**, теги которого ограничиваются символами квадратных скобок: **[ ]**.

**Весь текст, заключённый между начальным и конечным тегом, включая и сами эти теги, называется элементом**. Сам же текст между тегами — содержанием элемента. Содержание элемента может включать в себя любой текст, в том числе и другие элементы.

**Атрибуты**

**У тега могут быть свойства, называемые атрибутами, дающие дополнительные возможности форматирования текста.** Они записываются в виде сочетания: *имя атрибута-значения*, причём текстовые значения заключаются в кавычки.  
Например, можно выделить фрагмент текста определённым шрифтом, используя тег **<font>** и указав в этом теге название шрифта и желаемый размер:

<font face="Times, Arial, Courier" size=4>

оформляемый текст**</font>**.

**Тип элемента документа**

Имя тега определяет тип элемента. В HTML имя тега определяет лишь правило разметки, так, имя тега **<i>** (например **<i>** Курсивный текст**</i>**) определяет, что между тегами расположен текст, который следует отображать в браузере в курсивном начертании. XML представляет собой более гибкий стандарт. Имена тегов в нём не регламентируются жёстко: *пользователи могут вводить и использовать для своих нужд новые теги*(см. XML).

Например, мы сами для себя можем определить, что элементом с именем тега **<person>** мы определяем тип этого XML элемента, как фамилия имя и отчество. И скажем, год рождения, как часть информации по данному человеку:

**XML**:

<person>Иванов Иван Иванович <year>1984</year></person>

Можно даже задать дополнительно в таблицах CSS, что все теги **<person>** будут определённого цвета.

**HTML**:

<div name="person"> Иванов Иван Иванович, 1984 </div>

Тут уже нет возможности сформировать элемент документа для анализа текста, но оформить можно.

Таким образом, *HTML представляет собой неструктурированный текст, а XML — документ с иерархической структурой*, что даёт возможность обработки документа: трансформацию данных, поиск нужных элементов документа и т. д.

**Структура элемента**

В SGML и основанных на нём языках (HTML версий 2—4.01, языки семейства XML и т. д.) элементы должны быть строго вложенными друг в друга, «перехлёст» элементов недопустим:

неправильно:

<a><b></a></b>

правильно:

<person><i>Имя Фамилия</i></person>

Распространённое заблуждение, что SGML и HTML разрешают «перехлёст» элементов, основано на нарушающем стандарты поведении старых версий браузеров, пытавшихся «исправлять» ошибки разметки по собственным алгоритмам.  
Однако SGML (и HTML) допускают применение ряда сокращённых синтаксических конструкций, в том числе необязательные (опциональные) закрывающие и даже открывающие теги (автоматически достраиваемые при разборе документа на основе указанной схемы документа и текущего контекста). *В XML*, напротив, *все теги должны быть открыты и закрыты явным образом, что значительно упрощает алгоритм его разбора и снижает требования к аппаратному обеспечению для него*.

**Языки разметки предъявляют особые требования к структуре документа.** Рассмотрим, к примеру, требования языка [XML](http://localhost/Chelyshev/xml.php).

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

<?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](http://localhost/Chelyshev/html.php)-страницу. Также, 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 , внутри которой также можно задавать номер версии языка, номер кодовой страницы и другие параметры, необходимые программе-анализатору в процессе разбора документа.

**Правила создания XML- 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", т.е., 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**](http://localhost/Chelyshev/dtd.php)- определения (Document Type Definition) и **схемы данных** (Semantic Schema). Более подробно об использовании DTD и схемах будет описано в следующих разделах. В отличии от [SGML](http://localhost/Chelyshev/gml.php), 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.

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

**Элементы данных**

**Элемент**- это структурная единица XML- документа. Заключая слово в в тэги , мы определяем непустой элемент, называемый , содержимым которого является слово. В общем случае в качестве содержимого элементов могут 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.* Например, 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>

<universities-list>

<university id="1">

<title>Сибирский Государственный Университет Телекоммуникаций и Информатики</title>

<address URL="www.neic.nsk.su"/>

</university>

<university id="2">

<title>Новосибирский Государственный Университет</title>

<address URL="www.nsu.ru"/>

</university>

</universities-list>

</city>

<city>

<title>Москва</title>

<universities-list>

<university id="1">

<title>Московский Государственный Университет</title>

<address URL="www.msu.ru"/>

</university>

</universities-list>

</city>

</cities-list>

</country>

Производя в последствии поиск в этом документе, программа клиента будет опираться на информацию, заложенную в его структуру - используя элементы документа. Т.е. 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 **<river>Lena</river>** мы можем догадаться, что речь в этой части документа идет о реке, а вот во фрагменте **<name>Lena</name>** - о имени.

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 **<br>,** **<hr>**, **<img>**. Необходимо только помнить, что начальный и конечные тэги пустого элемента как бы объединяется в один, и надо обязательно ставить косую черту перед закрывающей угловой скобкой (например, **<empty/>**;)

**Комментарии**

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

**Атрибуты**

Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то имеется возможность использовать атрибуты эдлемента. 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</autho>

Примером использования атрибутов в HTML является описание элемента <font>:

<font color="white" name="Arial">Black</font>

**Cпециальные 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 спецсиволов могут определяться в XML документе при помощи компонентов (entity).

**Директивы анализатора**

Инструкции, предназначенные для анализаторов языка, описываются в XML документе при помощи специальных тэгов - ;. Программа клиента использует эти инструкции для управления процессом разбора документа. Наиболее часто инструкции используются при определении типа документа или создании пространства имен.

**CDATA**

Чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы, но, в отличии от комментариев, иметь возможность использовать их в приложении, необходимо использовать тэги . Внутри этого блока можно помещать любую информацию, которая может понадобится 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 ]].

**DTD**(англ. Document Type Definition — определение типа документа) — включает в себя два понятия:

* Термин, который используется для описания схемы документа или его части языком схем DTD;
* Язык схем DTD (DTD schema language) — компьютерный язык, который используется для записи фактических синтаксических правил метаязыков разметки текста SGML и XML. С момента его внедрения другие языки схем для спецификаций, такие как XML Schema и RELAX NG, выпускаются с дополнительной функциональностью.

*Из-за определённых различий между*[*XML*](http://localhost/Chelyshev/xml.php)*и*[*SGML*](http://localhost/Chelyshev/gml.php)*применение DTD также имеет некоторые особенности в зависимости от целевого документа*. В XML- документах DTD определяет набор действительных элементов, идентифицирует элементы, которые могут находиться в других элементах, и определяет действительные атрибуты для каждого из них. *Синтаксис DTD весьма своеобразен и от автора-разработчика требуются дополнительные усилия при создании таких документов*(сложность DTD является одной из причин того, что использование SGML, требующего определение DTD для любого документа, не получило столь широкого распространения как, например, [HTML](http://localhost/Chelyshev/html.php)). Как уже отмечалось,*в XML использовать DTD не обязательно* - документы, созданные без этих правил, будут правильно обрабатываться программой-анализатором, если они удовлетворяют основным требованиям синтаксиса XML. Однако контроль за типами элементов и корректностью отношений между ними в этом случае будет полностью возлагаться на автора документа. До тех пор, пока грамматика нашего нового языка не описана, его сможем использовать только мы, и для этого мы будем вынуждены применять специально разработанное программное обеспечение, а не универсальные программы-анализаторы..

Для того, чтобы использовать DTD в нашем XML-документе, мы можем или описать его во внешнем файле и при описании DTD просто указать ссылку на этот файл или же непосредственно внутри самого документа выделить область, в которой определить нужные правила. В первом случае в документе указывается имя файла, содержащего DTD- описания:

<?xml version="1.0" standalone="yes" ?>

<!DOCTYPE journal SYSTEM "journal.dtd">

...

Внутри же документа DTD- декларации включаются следующим образом:

...

<!DOCTYPE journal [

<!ELEMENT journal (contacts, issues, authors)>

...

]>

...

В том случае, если используются одновременно внутренние и внешние описания, то программой-анализатором будут сначала рассматриваться внутренние, т.е. их приоритет выше. При проверке документа XML- процессор в первую очередь ищет DTD внутри документа. Если правила внутри документа не определены и не задан атрибут **standalone ="yes"** , то программа загрузит указанный внешний файл и правила, находящиеся в нем, будут считаны оттуда. Если же атрибут standalone имеет значение "yes", то использование внешних DTD описаний будет запрещено.

**Определение элемента**

Элемент в DTD определяется с помощью дескриптора **!ELEMENT**, в котором указывается название элемента и структура его содержимого.

Например, для элемента **<flower>** можно определить следующее правило:

<!ELEMENT flower  PCDATA>

Ключевое слово **ELEMENT** указывает, что данной инструкцией будет описываться элемент XML. Внутри этой инструкции задается название элемента(**flower**) и тип его содержимого.

В определении элемента мы указываем сначала название элемента(**flower**), а затем его модель содержимого - определяем, какие другие элементы или типы данных могут встречаться внутри него. В данном случае содержимое элемента **flower** будет определяться при помощи специального маркера **PCDATA**(что означает *parseable character data - любая информация, с которой может работать программа-анализатор*). Существует еще две инструкции, определяющие тип содержимого: **EMPTY**, **ANY**. Первая указывает на то, что элемент должен быть пустым (например, **<red/>**), вторая - на то, что содержимое элемента специально не описывается.

Последовательность дочерних для текущего элемента объектов задается в виде списка разделенных запятыми названий элементов. При этом для того, чтобы указать количество повторений включений этих элементов могут использоваться символы **+**,**\***, **?**:

<!ELEMENT issue (title, author+, table-of-contents?)>

В этом примере указывается, что внутри элемента **<issue>** должны быть определены элементы **title**, **author**и **table-of-contents**, причем элемент **title**является обязательным элементом и может встречаться лишь однажды, элемент **author**может встречаться несколько раз, а элемент **table-of-contents** является опциональным, т.е. может отсутствовать. В том случае, если существует несколько возможных вариантов содержимого определяемого элемента, их следует разделять при помощи символа "**|**" :

<!ELEMENT flower (PCDATA | title )\*>

Символ **\***в этом примере указывает на то, что определяемая последовательность внутренних элементов может быть повторена несколько раз или же совсем не использоваться.

Если в определении элемента указывается "смешанное" содержимое, т.е. текстовые данные или набор элементов, то необходимо сначала указать **PCDATA**, а затем разделенный символом "**|**" список элементов.

Пример корректного XML- документа с явно определенным DTD:

<?xml version="1.0"?>

<!DOCTYPE journal [

<!ELEMENT contacts (address, tel+, email?)>

<!ELEMENT address (street, appt)>

<!ELEMENT street PCDATA>

<!ELEMENT appt (PCDATA | EMPTY)\*>

<!ELEMENT tel PCDATA>

<!ELEMENT email PCDATA>

]>

...

<contacts>

<address>

<street>Marks avenue</street>

<appt id="4">

</address>

<tel>12-12-12</tel>

<tel>46-23-62</tel>

<email>info@j.com</email>

</contacts>

Начало формы

Конец формы

**Коды**

**config.php**

<?php

define('host','localhost');

define('db\_name', 'curs');

define('user', 'root');

define('pass', 'Aleksiadis17');

define('tbl', 'content');

define('auth','auth');

$con=mysqli\_connect(host, user, pass) or die("Нет соединения с сервером");

mysqli\_select\_db($con, db\_name) or die("Нет соединения с БД");

mysqli\_query($con, "SET NAMES 'utf8'") or die("Не удалось установить кодировку соединения");

?>

**functions.php**

<?php

require\_once 'config.php';

function selectText($id){

global $con;

$txt = array();

$query = "SELECT id, text FROM ".tbl." WHERE id='$id'";

$res = mysqli\_query($con, $query);

while($row = mysqli\_fetch\_assoc($res)){

$txt[] = $row;

}

return $txt;

}

function updateText($txt, $id){

global $con;

$id = $id;

$txt = mysqli\_real\_escape\_string($con, $txt);

$query = "UPDATE ".tbl." SET text='$txt' WHERE id='$id'";

mysqli\_query($con, $query);

}

?>

**content.php**

<?php

echo "<head>

<meta http-equiv='content-type' content='text/html; charset=utf-8'>

<link rel='stylesheet' rev='stylesheet' type='text/css' href='style.css' />

<script type='text/javascript' src='ckeditor/ckeditor.js'></script>

<script src='js/highlight.pack.js'></script>

<script>hljs.initHighlightingOnLoad();</script>

<link rel='stylesheet' rev='stylesheet' type='text/css' href='js/styles/darcula.css' />

</head>";

require\_once 'functions.php';

require "auth.php";

$txt = selectText($id);

$txt[0]['text'] = str\_replace('&', '&amp;', $txt[0]['text']);

if($\_POST['update']){

updateText($\_POST['txt'], $\_POST['id']);

header("Location: ".$\_SERVER['PHP\_SELF']);

exit;

}

if($\_POST['edit']){

echo "

<body>

<div class='content1'>

<form method='post'>

<textarea id='editor1' name='txt' cols='100' rows='50'>";

echo $txt[0]['text'];

echo "</textarea>

<script type='text/javascript'>

var ckeditor1 = CKEDITOR.replace('editor1');

</script>

<br />

<input type='hidden' name='id' value=";

echo $txt[0]['id'];

echo " />";

echo "<input type='submit' name='update' value='Обновить' />";

echo "</form>

</div>

</body>";

}

else {

echo "<div class='content'>";

$txt[0]['text']=htmlspecialchars\_decode($txt[0]['text']);

echo $txt[0]['text'];

echo "</div>";

if ($\_SESSION['user']=="admin"){

echo "<form method='POST'>";

echo "<input type='submit' name='edit' value='Редактировать'>";

echo "</form>";

}}

?>

**auth.php**

<?php

session\_start();

if($\_GET['do'] == 'logout'){

unset($\_SESSION['user']);

session\_destroy();

}

if(!$\_SESSION['user']){

header("Location: enter.php");

exit;

}

?>

**enter.php**

<head>

<meta http-equiv='content-type' content='text/html; charset=utf-8'>

<link rel='stylesheet' rev='stylesheet' type='text/css' href='style.css' />

</head>

<?php

require\_once 'config.php';

session\_start();

if($\_SESSION['user']){

header("Location: index.php");

exit;

}

if($\_POST['submit']){

$query = "SELECT username, password FROM ".auth.";";

$res = mysqli\_query($con, $query);

while($row = mysqli\_fetch\_assoc($res)){

if($row['username'] == $\_POST['user'] AND $row['password'] == md5($\_POST['pass'])){

$\_SESSION['user'] = $row['username'];

header("Location: index.php");

exit;

}

}echo '<p>Логин или пароль неверны!</p>';

}

?>

Это страница авторизации. Введите логин и пароль

<hr>

<form method="post">

Username: <input type="text" name="user" value="" /><br />

Password: <input type="password" name="pass" value=""/><br />

<input type="submit" name="submit" value="Войти" />

</form>

**index.php**

<?php

require "auth.php";

?>

<head>

<title>Учебный портал: Языки разметки</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

</head>

<FRAMESET ROWS= "15%, \*" frameborder=1 border=4 bordercolor="black">

<frame src="logo.php" name="logo" scrolling=no noresize >

<FRAMESET ROWS= \*% COLS="13%, \*%" frameborder=1 border=4 bordercolor="black">

<frame src="menu.php" name="menu" scrolling=no noresize frameborder=1>

<frame src="markup.php" name="content" frameborder=1>

</frameset>

</frameset>

**файлы страниц**

<?php

$id="имя\_страницы";

require\_once 'content.php';

?>

**style.css**

body{

background:#ccc;

color:#000;

text-align:left;

text-indent: 0px;

}

.content{

top:0px;

font-family: 'Times New Roman';

font-size:16px;

text-indent: 30px;

}

h1{

font-family: 'Times New Roman';

font-size:50px;

}

ul{

text-align:left;

text-indent:0px;

left:0px;

}

table {

font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-Serif;

font-size: 14px;

border-radius: 10px;

border-spacing: 0;

text-align: center;

text-indent: 0px;

}

th {

background: #BCEBDD;

color: black;

text-shadow: 0 1px 1px #2D2020;

padding: 10px 20px;

}

th, td {

border-style: solid;

border-width: 0 1px 1px 0;

border-color: white;

}

td {

padding: 10px 20px;

background: #F8E391;

}

code{

text-indent:0px;

}