Пример трансформации XML-документа в HTML

**XSLT** — это мощное средство для трансформации XML-данных. Его часто используют как web-шаблонизатор: на вход подаётся XML-документ и XSLT-преобразование, а на выходе получают HTML-документ. Процесс получения HTML-документа из XML называют отображением или рендерингом.

Рассмотрим типичный пример рендеринга HTML.

Дан список музыкальных композиций в виде XML-документа.

<?xml version="1.0" encoding="utf-8"?>

<PlayList>

<Track Id="1170056tuNb" Length="280">

<Artist>Рихард Вагнер</Artist>

<Title>Полёт валькирии</Title>

</Track>

<Track Id="938304vu1E" Length="163">

<Artist>Эдвард Григ</Artist>

<Title>В пещере горного короля</Title>

</Track>

<Track Id="35532014SEz" Length="187">

<Artist>Иоган Бах</Artist>

<Title>Токката и фуга ре-минор</Title>

</Track>

<Track Id="264667GXiD" Length="203">

<Artist>Антонио Вивальди</Artist>

<Title>Времена года. Лето. Шторм</Title>

</Track>

<Track Id="613982Fj9E" Length="103">

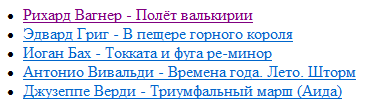
<Artist>Джузеппе Верди</Artist>

<Title>Триумфальный марш (Аида)</Title>

</Track>

</PlayList>

Отобразим данный документ в виде HTML ul/li списка, как это показано ниже:



Для этого используем следующее XSLT-преобразование:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;"> ]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes" />

<xsl:template match="/">

<xsl:apply-templates select="PlayList" />

</xsl:template>

<xsl:template match="PlayList">

<ul>

<xsl:apply-templates select="Track" />

</ul>

</xsl:template>

<xsl:template match="Track">

<li>

<a href="http://prostopleer.com/tracks/{@Id}">

<xsl:value-of select="Artist" />

<xsl:text> &mdash; </xsl:text>

<xsl:value-of select="Title" />

</a>

</li>

</xsl:template>

</xsl:stylesheet>

Данное преобразование вернёт следующий HTML:

<ul>

<li>

<a href="http://prostopleer.com/tracks/1170056tuNb">Рихард Вагнер — Полёт валькирии</a>

</li>

<li>

<a href="http://prostopleer.com/tracks/938304vu1E">Эдвард Григ — В пещере горного короля</a>

</li>

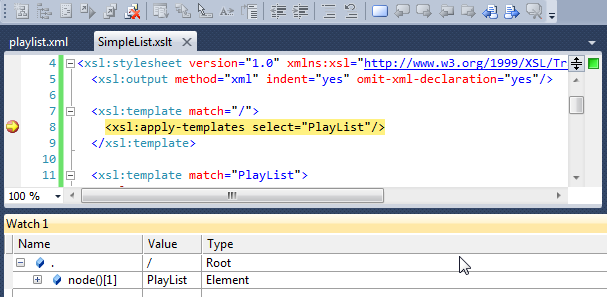
...

</ul>

XSLT-преобразование состоит из трёх шаблонов ([xsl:template](https://xsltdev.ru/xslt/xsl-template/)). Каждый шаблон «обслуживает» свою сущность, что даёт возможность легко вносить изменения. Если необходимо поменять отображение списка (например, добавить атрибут class), то мы редактируем шаблон match="PlayList". Если же мы хотим изменить отображение элементов списка, то тут, совершенно очевидно, что стоить менять шаблон match="Track". Фактически, XSLT не только даёт нам возможность разделить данные и представление (это задача любого шаблонизатора), но и позволяет разделять представления различных сущностей.

Отладка XSLT

Отладка помогает наглядно увидеть логику работы XSTL, структуру документа, значения переменных. Например, отладка поможет увидеть, что за сущность обрабатывает шаблон match="/". В Visual Studio отладка XSLT запускается сочетанием клавиш ALT+F5.

Отладка шаблона root

Добавив в окно Watch XPath выражение "." (точка), мы увидим, что текущий элемент шаблона — это корень (Root) документа. Здесь можно разместить контейнер div, или что-то относящееся ко всему XML-документу.

Работа с сущностями XML

Можно заметить, что в приведенных примерах присутствует сущность &mdash; Мы можем ее использовать, потому что определили ее в начале XSLT-документа

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;">

]>

Таким образом, &mdash; выводится, как символ с кодом &#8212;. Если нужно вывести строку «как есть», то стоит использовать CDATA следующим образом:

<xsl:text disable-output-escaping="yes"><![CDATA[ &mdash; ]]></xsl:text>

Элемент xsl:text

Необходимо обратить внимание на элементе [xsl:text](https://xsltdev.ru/xslt/xsl-text/). Он позволяет контролировать, что именно будет содержать TEXT-элемент.

XSLT-шаблон:

<xsl:template match="Track">

<li>

<a href="http://prostopleer.com/tracks/{@Id}">

<xsl:value-of select="Artist"/>

&mdash;

<xsl:value-of select="Title"/>

</a>

</li>

</xsl:template>

Полученный HTML:

<li>

<a href="http://prostopleer.com/tracks/264667GXiD">Антонио Вивальди

—

Времена года. Лето. Шторм</a>

</li>

Как видно из примера, отсутствие элемента xsl:text привело к появлению в HTML лишних переводов строк и пробелов. Безусловно, можно писать XSLT и без xsl:text, следующим образом:

<xsl:template match="Track">

<li>

<a href="http://prostopleer.com/tracks/{@Id}"><xsl:value-of select="Artist"/> &mdash; <xsl:value-of select="Title"/></a>

</li>

</xsl:template>

Такой шаблон трудночитаем и есть большая вероятность, что при сопровождении в нём будут появлятся ошибки.

Нужно стараться, чтобы форматирование XSLT-шаблона не влияло на результат трансформации. Именно поэтому я считаю, что использовать xsl:text — это хорошая практика.

Ветвления

Для ветвлений в XSLT есть специальные элементы: [xsl:if](https://xsltdev.ru/xslt/xsl-if/) и [xsl:choose](https://xsltdev.ru/xslt/xsl-choose/). Но часто этими инструментами сильно злоупотребляют. Более интересен приём, позволяющий не загромождать шаблон ветвлениями.

Рассмотрим пример реализации ветвлений. Дополним предыдущий пример возможностью выводить сообщение «Список пуст» в случае, если PlayList не содержит элементов Track. Решение с использованием xsl:choose будет таким:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;"> ]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>

<xsl:template match="/">

<xsl:apply-templates select="PlayList"/>

</xsl:template>

<xsl:template match="PlayList">

<xsl:choose>

<xsl:when test="count(Track) = 0">

<div>

<xsl:text><![CDATA[Список пуст]]></xsl:text>

</div>

</xsl:when>

<xsl:otherwise>

<ul>

<xsl:apply-templates select="Track"/>

</ul>

</xsl:otherwise>

</xsl:choose>

</xsl:template>

<xsl:template match="Track">

<li>

<a href="http://prostopleer.com/tracks/{@Id}">

<xsl:value-of select="Artist"/>

<xsl:text> &mdash; </xsl:text>

<xsl:value-of select="Title"/>

</a>

</li>

</xsl:template>

</xsl:stylesheet>

Решение с использованием дополнительного шаблона будет следующим:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;"> ]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>

<xsl:template match="/">

<xsl:apply-templates select="PlayList"/>

</xsl:template>

<xsl:template match="PlayList[count(Track) = 0]" >

<div>

<xsl:text><![CDATA[Список пуст]]></xsl:text>

</div>

</xsl:template>

<xsl:template match="PlayList">

<ul>

<xsl:apply-templates select="Track"/>

</ul>

</xsl:template>

<xsl:template match="Track">

<li>

<a href="http://prostopleer.com/tracks/{@Id}">

<xsl:value-of select="Artist"/>

<xsl:text> &mdash; </xsl:text>

<xsl:value-of select="Title"/>

</a>

</li>

</xsl:template>

</xsl:stylesheet>

Новая функциональность не добавила нового кода в старые шаблоны, новый шаблон максимально изолирован. Если понадобится добавить картинку к сообщению о пустом списке, то в первом случае, скорее всего, «разбухнет» элемент [xsl:when](https://xsltdev.ru/xslt/xsl-when/) в шаблоне match="PlayList". А вот во втором случае изменения будут только в специализированном шаблоне.

В предыдущем примере мы разделили две абсолютно разные ветки рендеринга элемента списка. Но что, если ветки различаются незначительно? Здесь использование xsl:if и xsl:choose вполне оправдано. Но мне бы хотелось показать другой подход: использование параметра mode у элемента [xsl:template](https://xsltdev.ru/xslt/xsl-template/).

В следующем примере навесим разные стили на чётные и нечётные элементы списка.

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;"> ]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>

<xsl:template match="PlayList">

<ul>

<xsl:apply-templates select="Track"/>

</ul>

</xsl:template>

<xsl:template match="Track">

<li>

<xsl:attribute name="class">

<xsl:apply-templates select="." mode="add-track-class"/>

</xsl:attribute>

<a href="http://prostopleer.com/tracks/{@Id}">

<xsl:value-of select="Artist"/>

<xsl:text> &mdash; </xsl:text>

<xsl:value-of select="Title"/>

</a>

</li>

</xsl:template>

<xsl:template mode="add-track-class" match="Track[position() mod 2 = 0]" >

<xsl:text>even</xsl:text>

</xsl:template>

<xsl:template mode="add-track-class" match="Track" >

<xsl:text>odd</xsl:text>

</xsl:template>

</xsl:stylesheet>

Циклы и сортировка в XSLT

Для циклов в XSLT есть элемент [xsl:for-each](https://xsltdev.ru/xslt/xsl-for-each/), но схожий эффект можно получить, используя обычный [xsl:apply-templates](https://xsltdev.ru/xslt/xsl-apply-templates/).

Выведем список композиций, отсортированный по длительности.

<ul>

<li>Рихард Вагнер — Полёт валькирии — 280</li>

<li>Антонио Вивальди — Времена года. Лето. Шторм — 203</li>

<li>Иоган Бах — Токката и фуга ре-минор — 187</li>

<li>Эдвард Григ — В пещере горного короля — 163</li>

<li>Джузеппе Верди — Триумфальный марш (Аида) — 103</li>

</ul>

Вариант с использованием xsl:for-each:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;">

]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>

<xsl:template match="PlayList">

<ul>

<xsl:for-each select="Track">

<xsl:sort select="@Length" data-type="number" order="descending"/>

<li>

<xsl:value-of select="concat(Artist, ' &mdash; ', Title, ' &mdash; ', @Length)"/>

</li>

</xsl:for-each>

</ul>

</xsl:template>

</xsl:stylesheet>

Вариант с использованием xsl:apply-templates:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;">

]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>

<xsl:template match="PlayList">

<ul>

<xsl:apply-templates select="Track">

<xsl:sort select="@Length" data-type="number" order="descending"/>

</xsl:apply-templates>

</ul>

</xsl:template>

<xsl:template match="Track">

<li>

<xsl:value-of select="concat(Artist, ' &mdash; ', Title, ' &mdash; ', @Length)"/>

</li>

</xsl:template>

</xsl:stylesheet>

Первый вариант короче и проще, но он нарушил принцип разделения ответственности для шаблонов. Теперь шаблон match="PlayList" стал содержать логику отображения элемента Track. Казалось бы, ничего страшного, но представим задачу, когда в списке встречаются композиции с Id и без. Для композиции с Id нужно отрендерить ссылку, а для остальных вывести только текст.

Вариант с использованием xsl:for-each:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;">

]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>

<xsl:template match="PlayList">

<ul>

<xsl:for-each select="Track">

<xsl:sort select="@Length" data-type="number" order="descending"/>

<li>

<xsl:choose>

<xsl:when test="not(@Id)">

<xsl:apply-templates select="." mode="TrackName" />

</xsl:when>

<xsl:otherwise>

<a href="http://prostopleer.com/tracks/{@Id}">

<xsl:apply-templates select="." mode="TrackName" />

</a>

</xsl:otherwise>

</xsl:choose>

</li>

</xsl:for-each>

</ul>

</xsl:template>

<xsl:template match="Track" mode="TrackName">

<xsl:value-of select="concat(Artist, ' &mdash; ', Title, ' &mdash; ', @Length)"/>

</xsl:template>

</xsl:stylesheet>

Вариант с использованием xsl:apply-templates:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE xsl:stylesheet [

<!ENTITY mdash "&#8212;">

]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>

<xsl:template match="PlayList">

<ul>

<xsl:apply-templates select="Track">

<xsl:sort select="@Length" data-type="number" order="descending"/>

</xsl:apply-templates>

</ul>

</xsl:template>

<xsl:template match="Track">

<li>

<a href="http://prostopleer.com/tracks/{@Id}">

<xsl:apply-templates select="." mode="TrackName" />

</a>

</li>

</xsl:template>

<xsl:template match="Track[not(@Id)]">

<li>

<xsl:apply-templates select="." mode="TrackName" />

</li>

</xsl:template>

<xsl:template mode="TrackName" match="Track">

<xsl:value-of select="concat(Artist, ' &mdash; ', Title, ' &mdash; ', @Length)"/>

</xsl:template>

</xsl:stylesheet>

В случае xsl:for-each нам потребовалось добавлять ветвление, а в случае xsl:apply-templates – новый шаблон.

Если бы шаблон match="PlayList" уже содержал ветвления и логику, то нам понадобилось некоторое время, чтобы разобраться, куда именно нам нужно вставить ветку. Вариант с xsl:apply-templates лишён этого недостатка, поскольку мы лишь декларируем новый шаблон, а не пытаемся внедриться в старые.

Использование xsl:for-each имеет ещё одну большую опасность. Если вы видите произвольный участок кода внутри шаблона match="PlayList", то предполагаете, что текущий элемент это PlayList, однако xsl:for-each меняет контекст. Увидев следующий код код:

<xsl:apply-templates select="." mode="TrackName" />

Вам потребуется внимательно присмотреться к контексту, чтобы понять, что select="." на самом деле выбирает текущий Track.

Шаблон mode="TrackName" match="Track" был добавлен для избежания дублирования кода, отображаюшего название. Я не сделал этого раньше, потому что в этом не было необходимости. Как только я заметил дублирование, я провёл рефакторинг и вынес общую логику отображения в новый шаблон.

xsl:for-each – это способ не плодить сущности. Вы просто добавляете логику отображения внутрь xsl:for-each и всё прекрасно работает. Проблемы начинаются потом, когда тело цикла разрастается, а проводить рефакторинг xsl:for-each намного сложнее, чем выносить дублированный код.