

Зачастую требуется не просто любой узел, а конкретный узел, выбранный на основе конкретных условий. Вы ранее уже видели пример, когда использовали выражение `/recipes/recipe[2]//instructions`. Это на самом деле сокращенная версия выражения `/recipes/recipe[position() = 2]//instructions`, которое означает, что процессор XPath должен пройти через каждый элемент `recipes` (конечно, такой элемент только один) и для каждого элемента `recipes` пройти через каждый элемент `recipe`. Для каждого элемента `recipe` проверить истинность выражения `position() = 2`. (Другими словами, есть ли в списке этот второй рецепт?) Если это предложение, называемое предикатом, истинно, то процессор использует этот узел и продолжает, возвращая любые инструкции.

С помощью предикатов можно сделать множество вещей. Например, можно вернуть только рецепты, в которых есть название: `/recipes/recipe[name]`. Это выражение просто проверяет, существует ли элемент `name` - потомок элемента `recipe`. Также можно поискать конкретные значения. Например, можно возвращать только элементы, которые называются "A balanced breakfast": `//recipe[name="A balanced breakfast"]`.

Имейте в виду, что предикат просто говорит процессору, возвращать ли текущий узел, поэтому в данном случае возвращается элемент `recipe`, а не имя. С другой стороны, можно указать процессору вернуть только название первого рецепта при помощи любого из этих двух выражений (см. листинг 22).

Листинг 22. Возвращение только названия первого рецепта

```
//recipe[@recipeId='1']/name
```

```
//name[parent::recipe[@recipeId='1']]
```

В первом выражении сначала мы выбираем все элементы `recipe`, а затем возвращаем только тот, у которого есть атрибут элемента `recipeId`, равный 1. Найдя этот узел, мы двигаемся к его дочернему узлу, который называется `name`, и возвращаем его. Во втором выражении сначала отыскиваются все элементы `name`, а затем выбирается только тот, у чьего родителя имеется атрибут 1. В любом случае, вы получите один и тот же вывод (см. листинг 23).

Листинг 23. Вывод

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<name>Gush'gosh</name>
```

Функции

XPath также предоставляет ряд функций. Некоторые из них относятся собственно к узлам, - например, те, которые ищут конкретную позицию, - некоторые обрабатывают строки, некоторые, - такие как суммирование, - с числами, некоторые с булевыми значениями.

Функции, связанные с узлами

Функции, связанные с узлами, помогают, например, выбрать конкретный узел в зависимости от позиции. Например, вы можете специально запросить последний рецепт: `//recipe[last()]`. Данное выражение выбирает все элементы `recipe`, а затем возвращает только последний. Вы также можете использовать функции отдельно, вместо того чтобы использовать их как часть предиката. Например, вы можете специально запросить посчитать элементы `recipe`: `count(//recipe)`.

Вы уже видели функцию `position()` и то, как она работает. Другие функции, связанные с узлами, включают `id()`, `local-name()`, `namespace-uri()` и `name()`.

Строковые функции

Большинство строковых функций предназначено для обработки строк, а не для их проверки, за исключением функции `contains()`. Функция `contains()` показывает, является ли данная строка частью большего целого. Это позволит, например, вернуть только узлы, содержащие определенные строки, такие как: `//recipe[contains(name, 'breakfast')]`.

Это выражение возвращает элемент `recipe`, который содержит в своем элементе `name` строку "breakfast".

Функция `substring()` позволяет выбрать конкретный диапазон символов из строки. Например, выражение: `substring(//recipe[2]/name, 1, 5)` возвращает `A bal`.

Первый аргумент - это полная строка, второй - позиция первого символа, а третий – это длина диапазона.

Среди прочих строковых функций имеются: `concat()`, `substring-before()`, `substring-after()`, `starts-with()` и `string-length()`.

Числовые функции

Числовые функции включают функцию `number()`, которая переводит значение в числовое, чтобы другие функции могли с ним работать. Числовые функции также включают: `sum()`, `floor()`, `ceiling()` и `round()`. Например, вы можете найти сумму всех значений `recipeld` при помощи выражения: `sum(//recipe/@recipeld)`.

Конечно, особого повода проводить подобное вычисление нет, однако это просто числовое значение, приведенное для примера.

Функция `floor()` находит наибольшее целое, которое меньше или равно данному значению, а функция `ceiling()` находит наименьшее целое, которое больше или равно данному значению. Функция `round()` работает традиционным образом - округляет (см. листинг 24).

Листинг 24. Результаты числовых функций

`floor(42.7) = 42`

`ceiling(42.7) = 43`

`round(42.7) = 43`

Булевы функции

Булевы функции наиболее полезны при работе с условными выражениями, о которых будет рассказано в разделе Условная обработка. Возможно, наиболее полезной функцией является `not()`, которая может быть использована для определения того, что определенный узел не существует. Например, выражение `//recipe[contains(name, 'breakfast')]` возвращает каждый рецепт, содержащий в элементе `name` строку "breakfast". Но что, если вам нужны все рецепты, кроме завтрака? Можно использовать выражение: `//recipe[not(contains(name, 'breakfast'))]`.

Другие булевы функции включают `true()` и `false()`, которые возвращают константы, и `boolean()`, которая преобразует значение в булево, чтобы его можно было использовать в качестве проверочного.

Организация циклов и импорт

Рассмотрим еще два важных аспекта использования таблиц стилей XSLT: создание циклов и импорт внешних таблиц стилей.

Организация циклов

При работе с XSLT необходимо привыкнуть к тому, что это функциональный язык, а не процедурный. Другими словами, обычно вы явно не контролируете способ, которым он выполняет данные ему инструкции. Однако существуют исключения. Например, существует возможность организовывать циклы и выполнять условные операции. Давайте начнем с циклов.

В предшествующем примере мы использовали выражения XSLT, встроенные в шаблон публикации, чтобы применить стиль к конкретным элементам. В некоторых случаях это отлично работает. Однако в ситуации, когда имеются сложные XML файлы или сложные

требования, иногда проще обратиться к информации явно (см. листинг 25).

Листинг 25. Прямое применение стилей с использованием циклов

```
<xsl:template match="recipe">
```

```
<h2><xsl:value-of select="./name"/></h2>
```

```
<h3>Ingredients:</h3>
```

```
<p>
```

```
<xsl:for-each select="./ingredients/ingredient">
```

```
<xsl:value-of select="./qty"/><xsl:text> </xsl:text>
```

```
<xsl:value-of select="./unit"/><xsl:text> </xsl:text>
```

`<xsl:value-of select="./food"/>
`

`</xsl:for-each>`

`</p>`

`<h3>Directions:</h3>`

``

`<xsl:for-each select="./instructions/instruction">`

`<xsl:value-of select="."/>`

`</xsl:for-each>`

</xsl:template>

</xsl:stylesheet>

Конструкция цикла очень похожа на структуру `for-each`, в честь которой она названа. Подобно тезке, каждый экземпляр цикла несет в себе следующее значение списка. В Java-программировании это могло бы быть следующее значение в массиве; здесь это следующий узел в совокупности узлов, возвращенных выражением XPath в атрибуте `select`. Это означает, что когда вы первый раз выполняете первый цикл, контекстным узлом является первый элемент `ingredient`. Это позволяет выбрать потомков этого элемента `qty`, `unit` и `food` и добавить их в документ так же, как это было сделано ранее при помощи шаблона. То же самое и с инструкциями, за исключением того, что они просто выводятся напрямую.

Результаты идентичны тем, что получаются при публикации через шаблоны - почти. Каждый шаблон добавляется к документу как индивидуальная строка, но так как нам необходимо обрабатывать эту информацию как один шаблон, мы теряем множество пробелов, которые видели раньше (см. рисунок 7).

Рисунок 7. Результаты


```
C:\Documents and Settings\Wick\Desktop\unit3>more result.html
Test version: 1.0 encoding: UTF-8 ?>html xmlns: "http://www.w3.org/TR/xhtml1/"
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet href="http://www.w3.org/TR/xhtml1/" type="text/css" />
<html>
<head>
<title>Recipe</title>
<head>
<body>
<h2>Dash gosh</h2>
<h3>Ingredients</h3>
<ol>
<li>1 pound hamburger</li>
<li>2 cups brown sugar</li>
<li>1 bag chopped onions</li>
<li>1 teaspoon dried dill</li>
</ol>
<h3>Directions</h3>
<ol>
<li>1. Drain the hamburger.
<li>1. Heat brown sugar and dill.
<li>1. Cook and drain pasta.
<li>1. Combine meat and pasta.
<li>1. Add balanced breakfast ingredients.
<li>1. Add cup of cereal.
<li>1. Add glass orange juice.
<li>1. Add cup milk.
<li>1. Add 2 slices toast.
<li>1. Combine cereal and milk in bowl.
</ol>
</body>
</html>
```

```
C:\Documents and Settings\Wick\Desktop\unit3>more result.html
Test version: 1.0 encoding: UTF-8 ?>html xmlns: "http://www.w3.org/TR/xhtml1/"
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet href="http://www.w3.org/TR/xhtml1/" type="text/css" />
<html>
<head>
<title>Recipe</title>
<head>
<body>
<h2>Dash gosh</h2>
<h3>Ingredients</h3>
<ol>
<li>1 pound hamburger</li>
<li>2 cups brown sugar</li>
<li>1 bag chopped onions</li>
<li>1 teaspoon dried dill</li>
</ol>
<h3>Directions</h3>
<ol>
<li>1. Drain the hamburger.
<li>1. Heat brown sugar and dill.
<li>1. Cook and drain pasta.
<li>1. Combine meat and pasta.
<li>1. Add balanced breakfast ingredients.
<li>1. Add cup of cereal.
<li>1. Add glass orange juice.
<li>1. Add cup milk.
<li>1. Add 2 slices toast.
<li>1. Combine cereal and milk in bowl.
</ol>
</body>
</html>
```

```
C:\Documents and Settings\Wick\Desktop\unit3>more result.html
Test version: 1.0 encoding: UTF-8 ?>html xmlns: "http://www.w3.org/TR/xhtml1/"
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet href="http://www.w3.org/TR/xhtml1/" type="text/css" />
<html>
<head>
<title>Recipe</title>
<head>
<body>
<h2>Dash gosh</h2>
<h3>Ingredients</h3>
<ol>
<li>1 pound hamburger</li>
<li>2 cups brown sugar</li>
<li>1 bag chopped onions</li>
<li>1 teaspoon dried dill</li>
</ol>
<h3>Directions</h3>
<ol>
<li>1. Drain the hamburger.
<li>1. Heat brown sugar and dill.
<li>1. Cook and drain pasta.
<li>1. Combine meat and pasta.
<li>1. Add balanced breakfast ingredients.
<li>1. Add cup of cereal.
<li>1. Add glass orange juice.
<li>1. Add cup milk.
<li>1. Add 2 slices toast.
<li>1. Combine cereal and milk in bowl.
</ol>
</body>
</html>
```

В результате использования предикатов, мы получили результат, который заменил шаблоны в файле result.html.