Чтение и запись XML файлов в Python

Автор: **Специалист ПК** - 09.02.2018

XML, или Extensible Markup Language (расширяемый язык разметки) – это язык разметки, часто используемый, чтобы структурировать, хранить и передавать данные между системами. Хотя и не так часто, как ранее, но он ещё используется в таких сервисах, как RSS и SOAP, а также для структурирования файлов наподобие документов Microsoft Office.

Поскольку Python – популярный язык для сети и анализа данных, вероятно, вам потребуется читать или записывать данные XML, в таком случае вам повезло. Читайте также: "Топ 5 интересных языков программирования для новичков".

На протяжении этой статьи мы в первую очередь взглянем на модуль `ElementTree` для чтения, записи и изменения файлов XML. Мы также сравним его с более старым модулем `minidom` в первых нескольких главах.

Оглавление [hide]

Модули XML

`Minidom`, или Minimal DOM Implementation – это упрощённая реализация объектной модели документа (Document Object Model, DOM). DOM – это интерфейс программирования приложений (Application Programming Interface, API), рассматривающий XML как древовидную структуру, где каждый узел в дереве есть объект. Таким образом, использование этого модуля требует, чтобы мы были знакомы с его функциональностью.

Модуль `ElementTree` предлагает более «питоний» интерфейс обращения с XML и является хорошим выбором для тех, кто не знаком с DOM. Также он кажется лучшим кандидатом для использования программистами-новичками благодаря простому интерфейсу, что вы увидите в этой статье.

Пример файла XML

В примерах ниже мы будем использовать следующий файл XML, который мы сохраним как “items.xml”:

```
<data>
<items>
<item name="item1">item1abc</item>
<item name="item2">item2abc</item>
</items>
</data>
```

Как вы можете видеть, это весьма простой пример XML, содержащий лишь немного вложенных объектов и один атрибут. Хотя этого должно быть достаточно, чтобы показать все операции с XML в этой статье.

Чтение документов XML

Использование minidom

Чтобы обработать документ XML с помощью `minidom`, мы должны сперва импортировать его из модуля `xml.dom`. Этот модуль использует функцию `parse`, чтобы создать объект DOM из нашего файла XML. Функция `parse` имеет следующий синтаксис:

```
xml.dom.minidom.parse(filename_or_file[, parser[, bufsize]])
```

Здесь имя файла может быть строкой, содержащей путь к файлу или объект файлового типа. Функция возвращает документ, который можно обработать как тип XML. Итак, мы можем использовать функцию `getElementByTagName()`, чтобы найти определённый тэг.

Поскольку каждый узел можно рассматривать как объект, мы можем получить доступ к атрибутам и тексту элемента через свойства объекта. В примере ниже мы добрались до атрибутов и текста отдельного узла и всех узлов вместе.

```
from xml.dom import minidom

# обработка файла xml по имени
 mydoc = minidom.parse('items.xml')

items = mydoc.getElementsByTagName('item')

# атрибут отдельного элемента
 print('Item #2 attribute:')
 print(items[1].attributes['name'].value)

# атрибуты всех элементов
 print('\nAll attributes:')
 for elem in items:
 print(elem.attributes['name'].value)

# данные отдельного элемента
 print('\nItem #2 data:')
 print(items[1].firstChild.data)
 print(items[1].childNodes[0].data)
```

```
 for elem in items:
 print(elem.firstChild.data)
```

Результат выглядит так:

```
$ python minidomparser.py
Item #2 attribute:
item2
All attributes:
item1
item2

Item #2 data:
item2abc
item2abc

All item data:
item1abc
item2abc
```

Если мы хотим использовать уже открытый файл, можно просто передать наш файловый объект функции `parse`, как здесь:

```
datasource = open('items.xml')

# обработка открытого файла
 mydoc = parse(datasource)
```

Также, если данные XML уже были загружены как строка, то мы могли бы использовать вместо этого функцию `parseString()`.

Использование ElementTree

`ElementTree` предлагает нам очень простой способ обработать файлы XML. Как всегда, чтобы его применить, мы должны сначала импортировать модуль. В нашем коде мы используем команду `import` с ключевым словом `as`, которое позволяет упростить имя (ET в данном случае) для модуля в коде.

Вслед за импортом мы создаём структуру дерева при помощи функции `parse` и получаем его корневой элемент. Как только добрались до корневого узла, мы можем легко путешествовать по дереву, поскольку оно является связным графом.

С помощью `ElementTree` мы можем, подобно примеру выше, получить атрибуты узла и текст, используя объекты, связанные с каждым узлом.

Код выглядит так:

```
import xml.etree.ElementTree as ET
 tree = ET.parse('items.xml')
 root = tree.getroot()

# атрибут отдельного элемента
 print('Item #2 attribute:')
 print(root[0][1].attrib)

# атрибуты всех элементов
 print('\nAll attributes:')
 for elem in root:
 for subelem in elem:
  print(subelem.attrib)
```

```
# данные всех элементов
 print('\nAll item data:')
 for elem in root:
 for subelem in elem:
  print(subelem.text)
```

Результат будет выглядеть следующим образом:

```
$ python treeparser.py
 Item #2 attribute:
 item2

All attributes:
 item1
 item2

Item #2 data:
 item2abc

All item data:
 item1abc
 item2abc
```

Как вы можете видеть, это очень похоже на пример с `minidom`. Одно из главных различий состоит в том, что объект `attrib` – это просто словарный объект, что делает его чуть более совместимым с другим кодом на Python. Нам также не нужно использовать `value`, чтобы добраться до значения атрибута объекта, как мы делали это ранее.

Вы могли заметить, то доступ к объектам и атрибутам с `ElementTree` чуть более «питоний», как мы упоминали ранее. Дело в том, что данные XML обрабатываются как простые списки и словари, в отличие от `minidom`, где применяется `xml.dom.minidom.Attr` и «текстовые узлы DOM».

Подсчёт элементов в документе XML

Использование minidom

Как и в предыдущем случае, `minidom` должен быть импортирован из модуля `dom`. Этот модуль предоставляет функцию `getElementsByTagName`, которую мы применим, чтобы найти элемент тега. Как только мы её получили, воспользуемся встроенным методом `len()`, чтобы получить количество подэлементов, связанных с узлом. Результат кода показан ниже:

```
from xml.dom import minidom

# обработка файла xml по имени
 mydoc = minidom.parse('items.xml')

items = mydoc.getElementsByTagName('item')

# общее количество элементов
 print(len(items))
```

Результат:

Имейте в виду, что этот код только посчитает число элементов-потомков там, где мы запускаем `len()`, в данном случае у корневого узла. Если вы хотите найти все подэлементы в гораздо большем дереве, вам придётся обойти все элементы и сосчитать каждого из их потомков.

Использование ElementTree

Похожим образом модуль `ElementTree` позволяет нам посчитать количество узлов, соединённых с некоторым узлом.

Пример кода:

```
import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')
 root = tree.getroot()

# общее количество элементов
 print(len(root[0]))
```

Результат выглядит так:

```
$ python counterxml.py
2
```

Запись документов XML

Использование ElementTree

`ElementTree` также хорош для записи данных в файлы XML. Код ниже показывает, как создать файл XML с той же самой структурой, как файл, что мы использовали в прошлых примерах.

Шаги:

1. Создать элемент, который будет вести себя как корень. В нашем случае тэг этого элемента – ”data”.
2. Когда у нас есть корневой элемент, мы можем создать подэлементы с помощью функции `SubElement`. Синтаксис этой функции: `SubElement(parent, tag, attrib={}, **extra)` Здесь `parent` – родительский узел, с которым нужно связаться, `attrib` – словарь, содержащий атрибуты элемента и `extra` – дополнительные ключевые слова (аргументы). Эта функция возвращает нам элемент, к которому можно привязать другие подэлементы, как мы это делаем в следующих строках, передавая элементы конструктору `SubElement`.
3. Хотя мы можем добавить наши атрибуты функцией `SubElement`, мы также можем применить функцию `set()`, как мы делаем в следующем коде. Текст элемента создаётся свойством `text` объекта `Element`.
4. В последних 3 строках кода ниже мы делаем строку из дерева XML и пишем данные в открытый нами файл.

Примеры кода:

```
import xml.etree.ElementTree as ET

# создаём файловую структуру
 data = ET.Element('data')
 items = ET.SubElement(data, 'items')
 item1 = ET.SubElement(items, 'item')
```

```
 item2.text = 'item2abc'

# создаём новый файл XML с результатами
 mydata = ET.tostring(data)
 myfile = open("items2.xml", "w")
 myfile.write(mydata)
```

Запустив этот код, получим новый файл “items2.xml”, который должен совпадать с исходным файлом “items.xml”, по крайней мере в смысле структуры данных XML. Возможно вы заметите, что в результате получается одна строка без отступов.

Поиск элементов XML

Использование ElementTree

Модуль `ElementTree` предлагает функцию `findall()`, которая помогает нам найти конкретные элементы в дереве. Она возвращает все элементы, удовлетворяющие определённому условию. Кроме того в модуле есть функция `find()`, которая возвращает только первый подэлемент, удовлетворяющий нужному критерию. Синтаксис для обеих функций таков:

```
findall(match, namespaces=None)
```

```
find(match, namespaces=None)
```

Для обеих функций параметр `match` может быть тэгом XML или путём. Функция `findall()` возвращает список элементов, и `find` возвращает одиночный объект типа `Element`.

Более того, есть ещё одна вспомогательная функция, которая возвращает текст первого узла, удовлетворяющего заданному критерию:

```
findtext(match, default=None, namespaces=None)
```

Вот пример кода, чтобы показать вам, как работают эти функции:

```
import xml.etree.ElementTree as ET
 tree = ET.parse('items.xml')
 root = tree.getroot()

# находим первый объект 'item'
 for elem in root:
 print(elem.find('item').get('name'))

# находим все объекты "item" и печатаем их атрибут "name"
 for elem in root:
 for subelem in elem.findall('item'):

 # если нам не нужно знать имя атрибута(ов), получаем словарь
  print(subelem.attrib)
```

```
$ python findtree.py
 item1
 {'name': 'item1'}
 item1
 {'name': 'item2'}
 item2
```

Изменение элементов XML

Использование ElementTree

Модуль `ElementTree` предоставляет несколько инструментов, чтобы изменить существующие документы XML. Пример ниже показывает, как изменить имя узла, атрибута и модифицировать его значение, и как добавить лишний атрибут к элементу.

Текст узла можно изменить, определив новое значение в текстовом поле объекта узла. Имя атрибута можно переопределить, используя функцию `set(name, value)`. Функция `set()` работает не только с существующим атрибутом, она также позволяет определить новый.

Код ниже показывает, как проделывать все эти операции:

```
import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')
 root = tree.getroot()

# изменяем текстовое поле
 for elem in root.iter('item'):
 elem.text = 'new text'

# модифицируем атрибут
 for elem in root.iter('item'):
 elem.set('name', 'newitem')

# добавляем атрибут
 for elem in root.iter('item'):
 elem.set('name2', 'newitem2')

tree.write('newitems.xml')
```

После запуска кода итоговый файл XML ”newitems.xml” будет иметь дерево XML со следующими данными:

```
<data>
 <items>
 <item name="newitem" name2="newitem2">new text</item>
 <item name="newitem" name2="newitem2">new text</item>
 </items>
 </data>
```

Как мы можем увидеть, по сравнению с исходным файлом XML, имена элементов

Вы также можете заметить, что запись данных XML подобным образом (вызывая `tree.write` с именем файла) добавляет форматирование к дереву XML, так что оно содержит новые строки и отступы.

Создание подэлементов XML

Использование ElementTree

В модуле `ElementTree` есть несколько способов добавить новый элемент. Первый, на который мы взглянем, состоит в использовании функции `makeelement()`, имеющей имя узла и словарь с атрибутами в качестве параметров.

Второй способ – через класс `SubElement()`, который берёт на вход родительский элемент и словарь атрибутов.

В примере ниже мы показываем оба метода. В первом случае узел не имеет атрибутов, так что мы создали пустой словарь (`attrib = {}`). Во втором случае мы используем заполненный словарь, чтобы создать атрибуты.

```
import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')
 root = tree.getroot()

# добавляем элемент к корневому узлу
 attrib = {}
 element = root.makeelement('seconditems', attrib)
 root.append(element)

# добавляем элемент ко второму узлу
 attrib = {'name2': 'secondname2'}
 subelement = root[0][1].makeelement('seconditem', attrib)
 ET.SubElement(root[1], 'seconditem', attrib)
 root[1][0].text = 'seconditemabc'

# создаём новый файл XML с новым элементом
 tree.write('newitems2.xml')
```

```
<data>
 <items>
 <item name="item1">item1abc</item>
 <item name="item2">item2abc</item>
 </items>
 <seconditems>
 <seconditem name2="secondname2">seconditemabc</seconditem>
 </seconditems>
 </data>
```

Как мы можем видеть, сравнив с исходным файлом, добавлены элемент “seconditems” и его подэлемент “seconditem”. К тому же, узел “seconditem” имеет “name2” в виде атрибута, и его текст “seconditemabc”, как и ожидалось.

Удаление элементов XML

Использование ElementTree

Как вы уже могли заметить, модуль `ElementTree` содержит необходимый функционал, чтобы удалить атрибуты и подэлементы узла.

Удаление атрибута

Код ниже показывает, как удалить атрибут узла, используя функцию `pop()`. Функция обращается к параметру объекта `attrib`. Она определяет имя атрибута и меняет его на `None`.

```
import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')
 root = tree.getroot()

# удаляем атрибут
 root[0][0].attrib.pop('name', None)

# создаём новый файл XML с результатами
 tree.write('newitems3.xml')
```

В итоге получится следующий файл XML:

```
<data>
 <items>
 <item>item1abc</item>
 <item name="item2">item2abc</item>
 </items>
 </data>
```

Как мы можем видеть в коде XML выше, у первого элемента нет атрибута “name”.

Удаление одного подэлемента

Один определённый подэлемент можно удалить, используя функцию `remove()`. Эта функция должна определять узел, который мы хотим удалить.

Следующий пример показывает её использование:

```
import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')
 root = tree.getroot()
```

```
# создаём новый файл XML с результатами
 tree.write('newitems4.xml')
```

В итоге получим следующий файл XML:

```
<data>
 <items>
 <item name="item2">item2abc</item>
 </items>
 </data>
```

Как мы можем видеть из кода XML выше, теперь только один узел “item”. Второй был удалён из исходного дерева.

Удаление всех подэлементов

Модуль `ElementTree` предоставляет нам функцию `clear()`, с помощью которой можно удалить все подэлементы данного элемента.

Пример ниже показывает нам, как использовать функцию `clear()`:

```
import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')
 root = tree.getroot()

# удаляем все подэлементы некоторого элемента
 root[0].clear()

# создаём новый файл XML с результатами
 tree.write('newitems5.xml')
```

В итоге будет следующий файл XML:

```
<data>
 <items />
 </data>
```

Как мы можем видеть в коде XML выше, все подэлементы элемента “items” удалены из дерева.

Подведём итоги

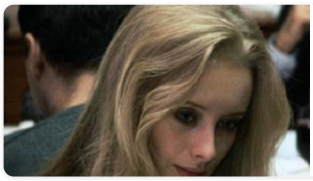
Python предлагает несколько вариантов обработки файлов XML. В этой статье мы рассмотрели модуль `ElementTree` и использовали его, чтобы обработать, создать, изменить и удалить файлы XML. Также мы использовали модель `minidom`, чтобы обработать файлы XML. Лично я бы порекомендовал применять модуль `ElementTree`, поскольку с ним гораздо легче работать и он более современный.

Завещание Градского удивило всех: все до копейки оставит...

Подробнее

С 2 декабря власти крепко возьмутся за антипрививочников: Кремль начнет наказывать по полной

Подробнее



Что вдова творила у гроба Градского: народ опускал глаза

Подробнее

Специалист ПК

Занимаюсь заработком в интернете с 2008 года. Освоила много разных удалённых профессий и влюбилась в создание сайтов на WP.