XML TOOLS

Xin Chen - xinchen@zju.edu.cn

Semester 2, 2018/19

TODAY

- DOM API
- SAX API

XML IN PYTHON

- The Python **standard library** provides a minimal but useful set of interfaces to work with XML.

- The two most basic and broadly used APIs to XML data are the **SAX** and **DOM** interfaces.

- **Simple API for XML *SAX*** : This is useful when your documents are **large** or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.
- **Document Object Model DOM API :** This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a **hierarchical tree**

Parsing XML with DOM APIs

- The Document Object Model "**DOM**" is a cross-language API from the World Wide Web Consortium **W3C** for accessing and modifying XML documents.
- Here is the easiest way to quickly load an XML document and to create a minidom object using the **xml.dom** module.

```
from xml.dom.minidom import parse
import xml.dom.minidom
```

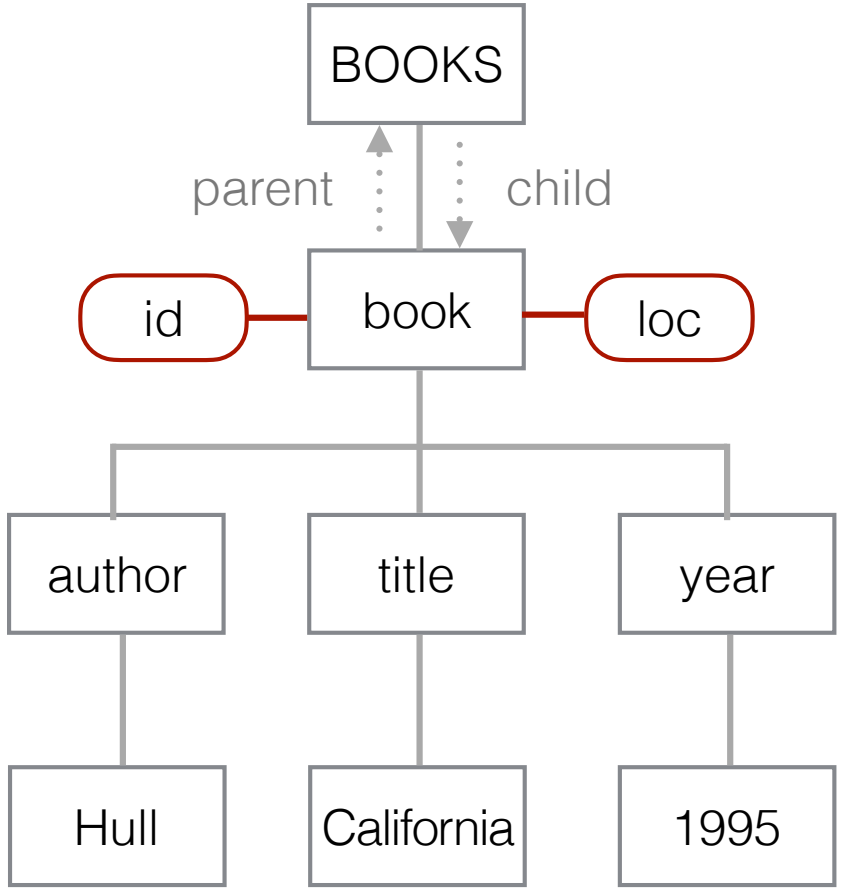
- More Information:

https://docs.python.org/3/library/xml.dom.html

XML DOM

- A DOM implementation presents an XML document as a tree structure

```
<?xml version="1.0" ?>

<BOOKS>

<book id="123" loc="library">

   <author>Hull</author>

   <title>California</title>

   <year> 1995 </year>

</book>

</BOOKS>
```



OBJECT IN THE DOM

- For every object in DOM, different methods and attributes are defined

remember class?

Interface	Section	Purpose
`Node`	Node Objects	Base interface for most objects in a document.
`NodeList`	NodeList Objects	Interface for a sequence of nodes.
`Document`	Document Objects	Object which represents an entire document.
`Element`	Element Objects	Element nodes in the document hierarchy.
`Attr`	Attr Objects	Attribute value nodes on element nodes.
`Text`	Text and CDATASection Objects	Nodes containing textual content from the document.

NODE AND NODELIST

- Node Objects:
  - **node.childNodes:**
    A list of nodes contained within this node.
  - **node.parentNodes:**
    The parent of the current node, or None for the document node. The value is always a Node object or None.
- NodeLists Objects:
  - **NodeList.item(i):**
    Return the i'th item from the sequence
  - **NodeList.length:**
    The number of nodes in the sequence.

DOCUMENT AND ELEMENT

- Document Objects:
  - **Document.documentElement:**
    The one and only root element of the document.
  - **Document.getElementsByTagName(*TagName*):**
    Search for all descendants (direct children, children's children, etc.) with a particular element type name.
- Element Objects:
  - **Element.getAttribute(*name*):**
    Return the value of the attribute named by *name* as a string.
  - **Element.getElementsByTagName(*TagName*):**
    Search for all descendants (direct children, children's children, etc.) with a particular element type name.

same as documents

ATTR AND TEXT

- Attr Objects:
  - **Attr.name:**
    The attribute name. In a namespace-using document it may include a colon.
  - **Attr.value:**
    The text value of the attribute. This is a synonym for the nodeValue attribute.

- Text Objects:
  - **Text.data:**
    The content of the text node as a string.

ELEMENT OBJECT

- element objects:

- **Element.getAttribute(*name*):**
  Return the value of the attribute named by *name* as a string.
- **Element.getElementsByTagName(*TagName*):**
  Search for all descendants (direct children, children's children, etc.) with a particular element type name.

EXAMPLE

- use a simple XML file **movies.xml** as an input!

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
    <type>War, Thriller</type>
    <format>DVD</format>
    <year>2003</year>
    <rating>PG</rating>
    <stars>10</stars>
    <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
    <type>Anime, Science Fiction</type>
    <format>DVD</format>
    <year>1989</year>
    <rating>R</rating>
    <stars>8</stars>
    <description>A schientific fiction</description>
</movie>
<movie title="Trigun">
    <type>Anime, Action</type>
    <format>DVD</format>
    <episodes>4</episodes>
    <rating>PG</rating>
    <stars>10</stars>
    <description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
    <type>Com edy</type>
    <format>VHS</format>
    <rating>PG</rating>
    <stars>2</stars>
    <description>Viewable boredom</description>
</movie>
</collection>
```

MOVIES.XML

- To get all the element of movie.xml

```
from xml.dom.minidom import parse

import xml.dom.minidom
```

parse the XML file into a DOM document object

```
DOMTree = xml.dom.minidom.parse("movies.xml")
```

```
collection = DOMTree.documentElement
```

Get the root element of the document

a list of 'movie' elements

```
movies = collection.getElementsByTagName("movie")
```

MOVIES.XML

- print details of the movies

```
for movie in movies:
    print('*****Movie*****')
    if movie.hasAttribute('title'):
        print('Title: ',movie.getAttribute('title'))
```

a list of elements

Returns `true` if the element has an attribute named by 'title'.

```
…
<movie title="Enemy Behind">
   <type>War, Thriller</type>
   <format>DVD</format>
   <year>2003</year>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Talk about a US-Japan war</description>
</movie>
…
```

part of movies.xml

MOVIES.XML

- print the type of the movies

```
for movie in movies:
    print('*****Movie*****')
    if movie.hasAttribute('title'):
        print('Title: ',movie.getAttribute('title'))
    type = movie.getElementsByTagName('type')[0]
    print('Type: ',type.childNodes[0].data)
```

return a NodeList

* type is an Element
* type.childNodes returns a DOM Text node

```
...
<movie title="Enemy Behind">
   <type>War, Thriller</type>
   <format>DVD</format>
   <year>2003</year>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Talk about a US-Japan war</description>
</movie>
...
```

part of movies.xml

MOVIES.XML

- print other informations of the movies …

```
for movie in movies:
    print('*****Movie*****')
    if movie.hasAttribute('title'):
        print('Title: ',movie.getAttribute('title'))
    type = movie.getElementsByTagName('type')[0]
    print('Type: ',type.childNodes[0].data)
    format = movie.getElementsByTagName('format')[0]
    print('Format: ',format.childNodes[0].data)
    rating = movie.getElementsByTagName('rating')[0]
    print('Rating: ',rating.childNodes[0].data)
    description = movie.getElementsByTagName('description')[0]
    print('Description: ',description.childNodes[0].data)
```

MOVIES.XML

```
from xml.dom.minidom import parse
import xml.dom.minidom

DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
movies = collection.getElementsByTagName("movie")

for movie in movies:
    print('*****Movie*****')
    if movie.hasAttribute('title'):
        print('Title: ',movie.getAttribute('title'))
    type = movie.getElementsByTagName('type')[0]
    print('Type: ',type.childNodes[0].data)
    format = movie.getElementsByTagName('format')[0]
    print('Format: ',format.childNodes[0].data)
    rating = movie.getElementsByTagName('rating')[0]
    print('Rating: ',rating.childNodes[0].data)
    description = movie.getElementsByTagName('description')[0]
    print('Description: ',description.childNodes[0].data)
```

module

a list of elements

get the text information

Output

```
*****Movie*****
Title:  Enemy Behind
Type:  War, Thriller
Format:  DVD
Rating:  PG
Description:  Talk about a US-Japan war
*****Movie*****
Title:  Transformers
Type:  Anime, Science Fiction
Format:  DVD
Rating:  R
Description:  A schientific fiction
*****Movie*****
Title:  Trigun
Type:  Anime, Action
Format:  DVD
Rating:  PG
Description:  Vash the Stampede!
*****Movie*****
Title:  Ishtar
Type:  Com edy
Format:  VHS
Rating:  PG
Description:  Viewable boredom
```

Parsing XML with SAX APIs

- SAX is a standard interface for event-driven XML parsing.

The **xml.sax** package provides a number of modules which implement the Simple API for XML (SAX) interface for Python.

```
import xml.sax
```

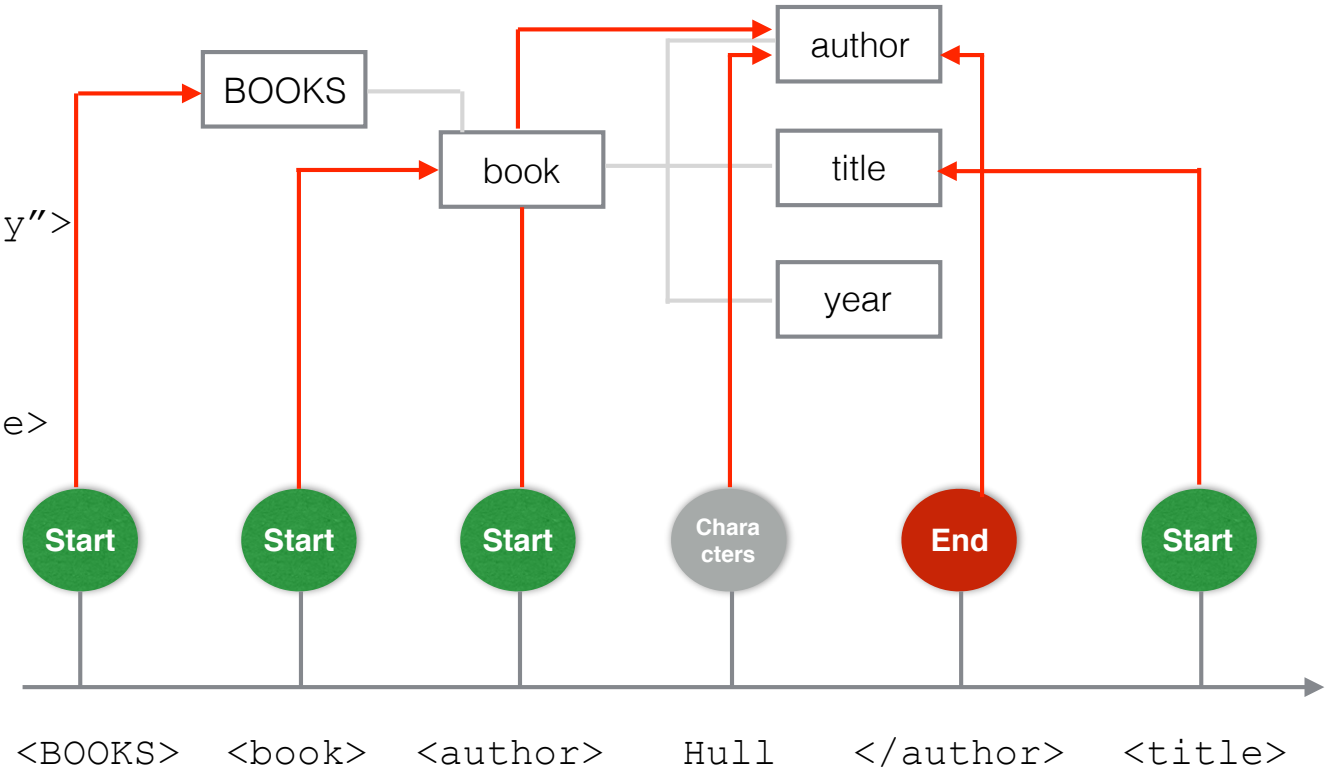
- More Information:

https://docs.python.org/3/library/xml.sax.html

SAX API

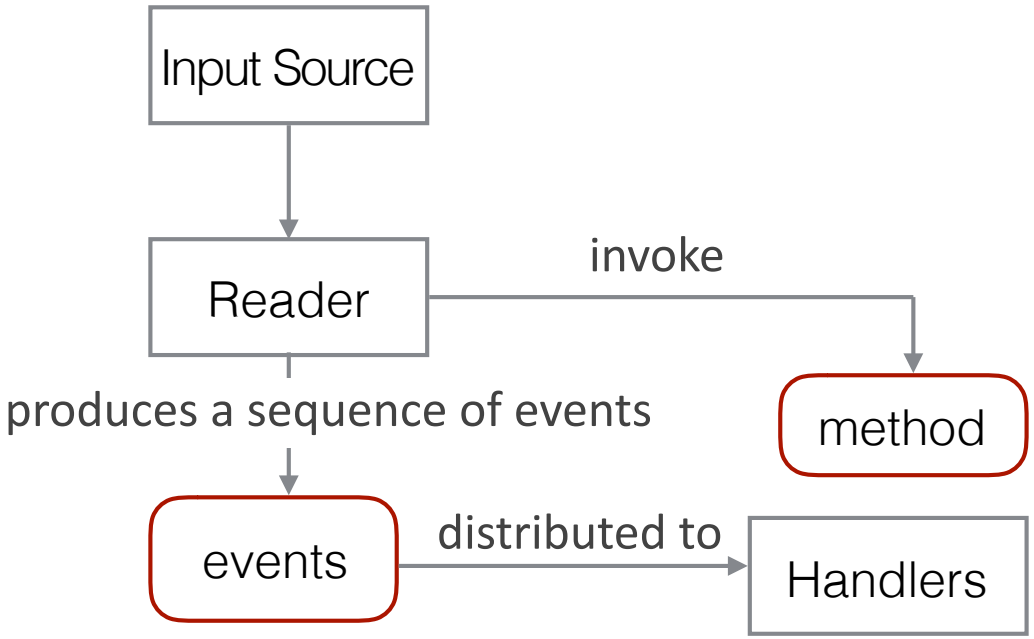
- A similarly fast but much simpler way to extract information from an XML document in an **event-drive** method

```
<?xml version="1.0" ?>

<BOOKS>

<book id="123" loc="library">

   <author>Hull</author>

   <title>California</title>

   <year> 1995 </year>

</book>

</BOOKS>
```



SAX API

- A typical SAX application uses three kinds of objects: **readers**(another term for parser), **handlers** and **input sources.**



- obtain a reader object and the handlers
- create or open the input sources
- the reader is called to parse the input
- During parsing, methods on the handler objects are called based on structural and syntactic events from the input data.

MAKE_PARSE

- Create and return a SAX XML **Reader** object. The parser object created will be of the first parser type the system finds.

```
xml.sax.make_parser( [parser_list] )
```

- **parser_list**: The optional argument consisting of a list of parsers to use which must all implement the make_parser method.

XML.SAX.XMLREADER

- SAX parsers implement the XMLReader interface.
  - **XMLReader.setFeature(featurename, value):**
    Set the featurename to value.
  - **XMLReader.parse(source):**
    Process an input source, producing SAX events.
  - **XMLReader.setContentHandler(handler):**
    Set the current ContentHandler.

- More Information:

https://docs.python.org/3/library/xml.sax.reader.html

CONTENT HANDLER

- Parsing XML with SAX generally requires you to create your own **ContentHandler** by subclassing xml.sax.ContentHandler.

```
class MovieHandler( xml.sax.ContentHandler ):
```

- ContentHandler is called at the start and end of each element.
  - **ContentHandler.startElement(name, attrs):**
    Signals the start of an element in **non-namespace** mode.
  - **ContentHandler.endElement(name):**
    Signals the end of an element in non-namespace mode.
  - **ContentHandler.characters(content):**
    Receive notification of character data.

EXAMPLE

- use a simple XML file **movies.xml** as an input!

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
    <type>War, Thriller</type>
    <format>DVD</format>
    <year>2003</year>
    <rating>PG</rating>
    <stars>10</stars>
    <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
    <type>Anime, Science Fiction</type>
    <format>DVD</format>
    <year>1989</year>
    <rating>R</rating>
    <stars>8</stars>
    <description>A schientific fiction</description>
</movie>
<movie title="Trigun">
    <type>Anime, Action</type>
    <format>DVD</format>
    <episodes>4</episodes>
    <rating>PG</rating>
    <stars>10</stars>
    <description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
    <type>Com edy</type>
    <format>VHS</format>
    <rating>PG</rating>
    <stars>2</stars>
    <description>Viewable boredom</description>
</movie>
</collection>
```

MOVIES.XML

- create **ContentHandler** by subclassing xml.sax.ContentHandler.

```
class MovieHandler( xml.sax.ContentHandler ):
   def __init__(self):
      self.CurrentData = ""
      self.type = ""
      self.format = ""
      self.year = ""
      self.rating = ""
      self.stars = ""
      self.description = ""
   # Call when an element starts
   def startElement(self, tag, attributes):
         self.CurrentData = tag
         if tag == "movie":
            print("*****Movie*****")
            title = attributes["title"]
            print("Title:", title)
```

MOVIES.XML

- create **ContentHandler** by subclassing xml.sax.ContentHandler.

```
# Call when an elements ends
    def endElement(self, tag):
        if self.CurrentData == "type":
            print(”Type:”, self.type)
        elif self.CurrentData == "format":
            print(“Format:", self.format)
        elif self.CurrentData == "year":
            print(“Year:", self.year)
        elif self.CurrentData == "rating":
            print(“Rating:", self.rating)
        elif self.CurrentData == "stars":
            print(“Stars:", self.stars)
        elif self.CurrentData == "description":
            print(“Description:", self.description)
        self.CurrentData = ""
```

MOVIES.XML

- create **ContentHandler** by subclassing xml.sax.ContentHandler.

```
# Call when a character is read
    def characters(self, content):
        if self.CurrentData == "type":
            self.type = content
        elif self.CurrentData == "format":
            self.format = content
        elif self.CurrentData == "year":
            self.year = content
        elif self.CurrentData == "rating":
            self.rating = content
        elif self.CurrentData == "stars":
            self.stars = content
        elif self.CurrentData == "description":
            self.description = content
```

MOVIES.XML

- create **ContentHandler** by subclassing xml.sax.ContentHandler.

```
import xml.sax
# create an XMLReader
parser = xml.sax.make_parser()
# turn off namespaces
parser.setFeature(xml.sax.handler.feature_namespaces, 0)
# override the default ContextHandler
Handler = MovieHandler()
parser.setContentHandler( Handler )
parser.parse("movies.xml")
```

Your defined ContentHandler handles

Process the input

SUMMARY

SAX

- SAX is **read-only**
- the entire file is **never stored** in memory
- can process information **faster** than DOM can when working with large files.
- must write codes to build some sort of tree yourself in SAX events

DOM

- DOM allows **changes** to the XML file
- using DOM exclusively can really **kill resources**
- cannot process information fast when working with large files
- the DOM provide a standard tree representation for XML data