Extracted from:

Enterprise Recipes with Ruby and Rails

High-Performance Parsing

Problem

Ruby is an interpreted language, which means it's not as fast as its compiled counterparts. By the time you read this, several Ruby compilers will be available, but none of them will automatically solve the problems described in this recipe. Usually, this is not a problem, but it could happen that you create a Ruby program for parsing an XML document that is too slow. Maybe you have developed it in record time, but that doesn't count much if it does not fulfill your customer's needs.

In this recipe, you'll learn how to increase the performance of your XML parsing code tremendously.

Ingredients

- Install the *LibXML*[5] gem:

  ```
  $ gem install libxml-ruby
  ```

  At the time of this writing, installing *LibXML* as a gem does not work out of the box on the Microsoft Windows platform.

Solution

Let's assume you have to create a Rails application that scans through an XML file containing credit card transactions and displays all transactions belonging to a particular credit card. The file might look like this:

Download xml/libxml2/ccdemo/data/cc_xactions/20080729.xml

```
<?xml version='1.0'?>
<cc-xactions date='20080729'>
  <cc-xaction id='100001' cc-ref='2537403' type='credit' amount='12.00'>
    <text>Monthly bill.</text>
  </cc-xaction>
  <!-- ... -->
  <cc-xaction id='400224' cc-ref='95932' type='purchase' amount='19.99'>
    <text>A new book.</text>
  </cc-xaction>
</cc-xactions>
```

5. http://libxml.rubyforge.org/

Each transaction has a unique identifier that can be found in the id= attribute. All credit cards are identified by a reference ID, which is stored in the cc-ref= attribute (using the credit card number to identify a credit card is not allowed, which is why we use an artificial identifier).

If you get money from your customer, the type= attribute is purchase; otherwise, it's credit. amount= tells us how much money has been transferred, and the content of the <*text*> element appears on the customer's credit card bill.

The input files contain several thousand credit card transactions, and you've tried all traditional methods already, but your application is still too slow. You've measured performance and have come to the conclusion that more CPU cycles are needed in the XML parsing code.

To solve this problem, we'll use the *LibXML* library. It is a C extension and embeds the GNOME *libxml2* library[6] into the Ruby interpreter. Like REXML, it uses XPath wherever possible. Our model looks like:

Download xml/libxml2/ccdemo/app/models/credit_card_transaction.rb

```ruby
require 'xml/libxml'

class CreditCardTransaction
  XACTION_DIR = File.join('data', 'cc_xactions')

  attr_reader :xaction_id, :cc_ref, :type, :amount, :text

  def initialize(xaction_id, cc_ref, type, amount, text)
    @xaction_id, @cc_ref, @type = xaction_id, cc_ref, type
    @amount, @text = amount, text
  end

  def self.find_all(cc_ref)
    xactions = []
    input_file = "#{XACTION_DIR}/xactions.xml"
    doc = XML::Document.file(input_file)
    doc.find('//cc-xactions/cc-xaction').each do |node|
      if node['cc-ref'] == cc_ref
        xactions << CreditCardTransaction.new(
          node['id'],
          node['cc-ref'],
          node['type'],
          node['amount'],
          node.find_first('text')
        )
      end
    end
```

6. http://xmlsoft.org/

```
-       xactions
-     end
30  end
```

That does not differ much from our REXML solution, because both libraries have a similar API, and they even use UTF-8 for encoding characters internally.

In line 16, we read and parse our input file in a single step. The result is a tree representation of our XML document. In line 17, we iterate over all <*cc-xaction*> elements using the find() method. As with REXML's iterators, we can use an XPath expression to select the nodes we're interested in (see Recipe 22, *Use XML Files as Models*, on page 146). In line 20, we copy the content of an attribute, and in line 24, we copy an element's content. The controller action for finding all credit card transactions is trivial:

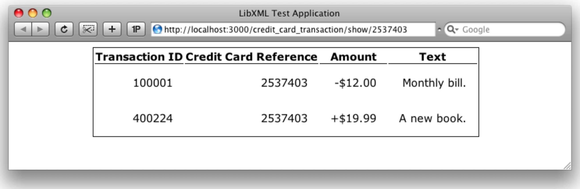
Download xml/libxml2/ccdemo/app/controllers/credit_card_transaction_controller.rb

```
class CreditCardTransactionController < ApplicationController
  def show
    @xactions = CreditCardTransaction.find_all(params[:id])
  end
end
```

The view looks as follows, and its result can be seen in Figure 5.2, on the following page:

Download xml/libxml2/ccdemo/app/views/credit_card_transaction/show.html.erb

```
<% if @xactions.size == 0 %>
  <p>Currently, there are no transactions.</p>
<% else %>
  <table>
    <tr>
      <th>Transaction ID</th>
      <th>Credit Card Reference</th>
      <th>Amount</th>
      <th>Text</th>
    </tr>
    <% for xaction in @xactions %>
      <tr>
        <td><%= xaction.xaction_id %></td>
        <td><%= xaction.cc_ref %></td>
        <% sign = (xaction.type == 'purchase') ? '+' : '-' %>
        <td><%= sign + number_to_currency(xaction.amount) %></td>
        <td><%= xaction.text %></td>
      </tr>
    <% end %>
  </table>
<% end %>
```



Transaction ID	Credit Card Reference	Amount	Text
100001	2537403	-$12.00	Monthly bill.
400224	2537403	+$19.99	A new book.

Figure 5.2: List of credit card transactions

Although we have read model data from an XML file and although we have parsed the files using a C extension, we could still use Rails' regular MVC pattern. By looking at the view, you cannot see where the data came from.

Discussion

So far, so good, but our solution does not differ much from a REXML solution. Why should it be so much faster?

The secret ingredient is the raw power of C, but you cannot determine how much faster your program runs by looking at the code. I have provided a little benchmark that compares three functions that do the same but use different parsers. The first one uses *LibXML*:

Download xml/libxml2/performance_test.rb

```ruby
require 'xml/libxml'
def libxml_parse(xml_string)
  xactions = []
  parser = XML::Parser.new
  parser.string = xml_string
  doc = parser.parse
  doc.find('//cc-xactions/cc-xaction').each do |node|
    xactions << CreditCardTransaction.new(
      node['id'],
      node['cc-ref'],
      node['type'],
      node['amount'],
      node.find('text').to_a.first.content
    )
  end
  xactions
end
```

That looks exactly like the code we used in the CreditCardTransaction class. The only difference starts in line 4. Here we read our input document from a string and not from a file to create fair testing conditions for all approaches. Here's a solution that uses REXML:

Download xml/libxml2/performance_test.rb

```ruby
require 'rexml/document'
def rexml_parse(xml_string)
  xactions = []
  doc = REXML::Document.new(xml_string)
  doc.elements.each('//cc-xactions/cc-xaction') do |node|
    xactions << CreditCardTransaction.new(
      node.attributes['id'],
      node.attributes['cc-ref'],
      node.attributes['type'],
      node.attributes['amount'],
      node.elements['text'].text
    )
  end
  xactions
end
```

This function should not contain any surprises, and for the sake of completeness we'll look at an *Hpricot* version, too (see Recipe 25, *Work with HTML and Microformats*, on page 165 to learn more about *Hpricot*):

Download xml/libxml2/performance_test.rb

```ruby
require 'hpricot'
def hpricot_parse(xml_string)
  xactions = []
  doc = Hpricot.XML(xml_string)
  (doc/'//cc-xactions/cc-xaction').each do |node|
    xactions << CreditCardTransaction.new(
      node['id'],
      node['cc-ref'],
      node['type'],
      node['amount'],
      (node/'text').inner_html
    )
  end
  xactions
end
```

Hpricot was always meant to be an HTML parser, but its XML() method makes it possible to parse XML documents, too.

As you can see, the three solutions differ only in a few characters, and now we use Ruby's *Benchmark* module to compare them.

Download xml/libxml2/performance_test.rb

```ruby
require 'benchmark'
xml_string = IO::read(input_file)
label_width = 8
Benchmark.bm(label_width) do |x|
  x.report('rexml:  ') { rexml_parse(xml_string) }
  x.report('libxml: ') { libxml_parse(xml_string) }
  x.report('hpricot:') { hpricot_parse(xml_string) }
end
```

First, we feed our two functions with an example document containing 1,000 credit card transactions (I've run those tests on an Apple MacBook Pro):

```
mschmidt> ruby performance_test.rb 1000
              user     system      total        real
rexml:    3.110000   0.040000   3.150000 (  3.189711)
libxml:   0.050000   0.010000   0.060000 (  0.060367)
hpricot:  0.510000   0.000000   0.510000 (  0.523038)
```

That's pretty impressive already, but let's see what happens when we parse 10,000 elements:

```
mschmidt> ruby performance_test.rb 10000
              user     system      total        real
rexml:  218.810000   1.640000 220.450000 (222.433778)
libxml:   2.020000   0.110000   2.130000 (  2.168987)
hpricot:  6.600000   0.060000   6.660000 (  6.727825)
```

Wow! As you can see, not only is *LibXML* much faster than REXML, but it is really fast! Regarding this figures, it would be completely impossible to provide a satisfying user experience using REXML, but the performance of *LibXML* is still acceptable. *Hpricot* has excellent performance, too, but when you have to install a separate library anyway, you should install the fastest one. In addition, *LibXML* fully implements the XML standard (and some of its relatives), while *Hpricot* does not.

Despite all this, you have to consider some shortcomings: although *LibXML* is probably one of the most complete XML implementations available, its Ruby binding is still in an early stage of development, and as with all C extensions, you have to test your software intensely. You especially have to check for memory leaks!

REXML is convenient and an adequate solution for small XML documents. But the API of *LibXML* is nice, too, and it's currently the only library that enables you to handle really big documents sufficiently fast.