Perl Mongers im Ruhrgebiet

Ruhr.pm

XML-Daten verarbeiten mit XML::LibXML

Autor: Simon Wilper

EMail: simon AT ruhr.pm.org

Datum: 20. November 2007

http://ruhr.pm.org/

Grundlagen

- Perl-Binding für libxml2 (aus GNOME-Projekt)
- Klassischer SAX-Parser
- DOM-Parser
- XPATH-Schnittstelle für XPATH-API in libxml2

Ruhr.pm

Unterteilung in XML::LibXML::~

Parser
DOM
SAX
Reader
Document
Node
Element
Comment
CDATASection
Attr
DocumentFragment
Namespace
PI
Dtd
RelaxNG
Schema
XPathContext

SAX-Parser (Simple API for XML)

- Basiert auf der XML::SAX API
- Direkter Zugriff auf die Parser-Engine – kein DOM-Tree
- Es werden SAX-Events erzeugt, auf die mit Handlern reagiert wird
- Sinnvoll bei großen Dokumenten

Beispiel

```
#!/usr/bin/perl

use warnings;
use strict;

use XML::SAX;
use MyHandler;    # siehe MyHandler.pm in den
                  # Codebeispielen

my $parser = XML::SAX::ParserFactory->parser(
        Handler => MyHandler->new
);

$parser->parse_string( ... );
```

Parser wird von der ParserFactory bereitgestellt

Eigener Handler wird zugewiesen

Parser kann beginnen

MyHandler.pm

```
package MyHandler;
use base qw(XML::SAX::Base);

# Handler fuer Beginn eines Elements
sub start_element {
        my( $self, $element ) = @_;

        # object-Elemente filtern
        if ( $element->{'LocalName'} eq 'object' ) {
                print "Element started: " .
                        $element->{'LocalName'} . "\n";

                # Da $element->{'Attributes'} eine Hashreferenz
                # ist, hier dereferenzieren, um darueber zu
                # iterieren
                foreach my $attr (
                        keys %{$element->{'Attributes'}}
                                ) {
                        printf(
                        "%10s : [%s]\n",
                        $element->{'Attributes'}->{$attr}->{'LocalName'},
                        $element->{'Attributes'}->{$attr}->{'Value'}
                        );
                }
        }
        print "\n";
}

sub end_element { ... }
sub characters { ... }
1;
```

LocalName:
Elementname ohne Namespace

Attributes:
Hashref mit Element-Attributen

Value:
Attributwert

Ruhr.pm

DOM-Parser (Document Object Model)

- DOM Level 3 Parser
- Funktionen nach DOM Spezifikation (W3C)

DOM-Beispiel

```
#!/usr/bin/perl

use warnings;
use strict;

use XML::LibXML;

my $parser = XML::LibXML->new();
my $doc    = $parser->parse_string( ... );
my $root   = $doc->documentElement();
print $root->nodeName, "\n";
```

DOM-Parser durch new() erzeugen

parse_*(): Parsen, Erzeugen eines DOM-Trees, resultiert in ein DOM-Document

documentElement / getDocumentElement
gibt das Wurzelelement zurück

nodeName
Name des Nodes (“xml”)

DOM-Beispiele ctd.

```
foreach my $node ( $root->childNodes ) {
        print "Node: ", $node->nodeName, "\n";
}
```

childNodes
gibt eine Liste der Kindelemente zurück

```
if ( $node->hasAttributes ) {
      foreach( $node->attributes ) {
         printf( " %10s: %-20s\n",
           $_->name, $_->value );
      }
}
```

hasAttributes
prüft auf Präsenz von Attributen

attributes
gibt eine Liste von Attributen zurück

name, value
gibt den Namen / den Wert des Attributs zurück

Validieren von Dokumenten

```
#!/usr/bin/perl

use warnings;
use strict;

use XML::LibXML;

my $parser = XML::LibXML->new;
my $dtd    = XML::LibXML::Dtd->new(
 'SOME // Public / ID / 1.0', 'html.dtd'
);

my $doc;
eval { $doc = $parser->parse_file( 'test.html' ) };
eval { $doc->validate( $dtd ) };

if ( !$doc->is_valid ) {
        print "Invalid Document: $@\n";
} else {
        print "Document successfully validated!\n";
}
```

Neues DTD erstellen
Konstruktor nimmt den Public Identifier und den Pfad zum DTD

eval-Blöcke zur Fehlerbehandlung
Die Fehlermeldung steht anschließend in der reservierten Variable $@

validate
Validiert Dokument

is_valid
Prüft auf Validität des Dokuments

XPath

```
my $parser = XML::LibXML->new;
my $doc = $parser->parse_string(<<EOT);
<?xml version="1.0"?>
<xml>
        Text im Dokument
        <element id="myID" name="myname" style="old" />
        Text kurz vorm Ende
        <object objid="001" objname="Object1" />
        <element id="002" name="myname2" />
</xml>
EOT

my $root  = $doc->documentElement();
my $xc    = XML::LibXML::XPathContext->new( $root );
my $nodes = $xc->findnodes( '/xml/element[@id=002]' );

foreach my $node ( $nodes->get_nodelist ) {
        print "Node: ", $node->nodeName, "\n";
        dumpAttribs( $node );
}
```

XpathContext erzeugen
Hier vom Rootelement ausgehend

findnodes($xpath)
gibt eine NodeList mit Matches zurück

get_nodelist
gibt eine Perl-Liste der Nodes zurück

Vielen Dank für Eure Aufmerksamkeit