1. 개요

자연어처리의 기반 모듈은 C로 작성된 경우가 많습니다. 대표적인 것은 형태소 분석기인데, 최근들어 형태소 분석기를 Java, Python, Perl 등의 다른 언어에서 사용하고자 하는 요청이 늘어나고 있습니다. 특히, Perl의 경우 강력한 문자열 처리가 가능해서 일단 형태소분석기의 perl wrapper를 만들어둔다면 개발속도 등 여러 측면에서 시너지가 날것으로 예상됩니다.

이와 같은 배경에서, 이 글에서는 Perl의 XS(eXternal Subroutine)를 이용해서 C 라이브러리를 Perl에서 사용하는 방법에 대해 기술합니다. 더불어 FCGI와 연동해서 WEB API를 작성하는 방법에 대해서도 살펴보고자 합니다.

2. Extension 만들기

XS를 사용하기 위해서 우선 extension을 만들어야합니다.

$ h2xs -A -n Moran

Moran이라는 이름으로 extension 파일들이 만들어집니다.

$ ls
Changes MANIFEST Makefile.PL README lib Moran.xs ppport.h t

<주요 파일>

- Changes
  : 버그 픽스, 기능 추가 등의 히스토리를 기술
- MANIFEST
  : 'make dist'로 tar.gz 패키지를 만들때 포함되어야할 파일 기술
- Makefile.PL
  : Makefile을 생성하는 perl 프로그램 (automake의 Makefile.am과 비슷)
- Moran.xs
  : C 라이브러리를 호출하는 서브루틴을 작성
- lib/Moran.pm
  : Moran.xs에 기술된 서브루틴을 perl 서브루틴으로 wrapping하는 최종 인터페이스

3. Makefile.PL

예를 들어, 사용할 C 라이브러리 이름이 'moran.so'라고 하면, 대략 아래와 같이 편집합니다.

```
use 5.010001;
use ExtUtils::MakeMaker;

my $MORAN_HOME=q(/home/moran);
my $MORAN_LIB="-L$MORAN_HOME/lib -lmoran";
my $MORAN_INC="-I. -I$MORAN_HOME/include";

# See lib/ExtUtils/MakeMaker.pm for details of how to influence
# the contents of the Makefile that is written.
WriteMakefile(
    NAME              => 'Moran',
    VERSION_FROM      => 'lib/Moran.pm', # finds $VERSION
    PREREQ_PM         => {}, # e.g., Module::Name => 1.1
    ($] >= 5.005 ?     ## Add these new keywords supported since 5.005
      (ABSTRACT_FROM  => 'lib/Moran.pm', # retrieve abstract from module
       AUTHOR         => 'yourname <yourname@gmail.com>') : ()),
    LIBS              => [$MORAN_LIB], # e.g., '-lm'
    DEFINE            => '', # e.g., '-DHAVE_SOMETHING'
    INC               => "$MORAN_INC", # e.g., '-I. -I/usr/include/other'
    # Un-comment this if you add C files to link with later:
    # OBJECT            => '$(O_FILES)', # link all the C files too
);
```

편집을 마친 이후, 아래 명령으로 Makefile을 생성합니다.

$ perl Makefile.PL

$ make

생성된 Makefile로 간단히 'make' 명령을 내리면, 내용이 비어있는 Moran.xs로부터 Moran.c, Moran.so 라이브러리가 생성됩니다.

$ ls Moran.c

$ ls blib/arch/auto/Moran.so

이제 Moran.xs, Moran.pm을 작성하기 위한 준비는 마쳤습니다.

4. Moran.xs

moran.so에서 필요한 header 파일을 적당히 위치시키고 'MODULE = Moran' 아래 쪽에 서브루틴에 대한 코딩을 시작합니다.

우선 moran.so 라이브러리의 헤더 파일(moran.h)은 아래와 같이 있다고 가정합시다. 코딩할 부분은 C 함수와 Perl과의 인터페이스 서브루틴을 만드는 것인데, C의 문법과 상당히 유사해서 한번 해보시면 쉽게 따라해보실 수 있는 수준이라고 생각됩니다.

```
/*
 *  사전 파일을 메모리에 로딩하고 필요한 핸들러를 초기화
 *  핸들러 메모리 주소를 리턴, 실패시 NULL 리턴
 */
void* initialize_moran(char* dictionary_path);

/*
 * 핸들러에 할당된 메모리를 해제
 */
void  finialize_moran(void* dict);

/*
 * string을 입력받아 형태소분석하고 그 결과를 리턴
 * 실패시 NULL 리턴
 * 리턴된 메모리는 사용 후 반드시 free()해줘야한다
 */
char* analyze_moran(void* dict, char* string);
```

---

initialize_moran() 함수의 리턴 값은 메모리의 주소인데, 여기서 고민이 생기게 됩니다. 이 주소값을 실제로 호출할 Perl로 리턴해서 관리할 것인지, 아니면 Moran.xs에서 전역변수를 잡아서 싱글톤(singleton) 구조로 만들것이지 결정해야합니다. 여기서는 간단하게 싱글톤 구조로 접근하겠습니다.

전역변수를 사용한다면, 아래와 같이 전역변수를 선언하고 이를 초기화할때 이미 초기화되어 있는지 확인해봐야합니다.

```
#include "moran.h"
void* dict=NULL;
```

```
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

#include "ppport.h"

MODULE = Moran        PACKAGE = Moran

void
initialize_moran_xs(dictionary_path)
    char* dictionary_path
    CODE:
        if( dict == NULL ) {
            dict = initialize_moran(dictionary_path);
            if(dict == NULL) {
                fprintf(stderr,"initialize_moran_xs() fail \n");
                exit(1);
            }
        }
```

위에서 char* dictionary_path를 파라미터로 받는데, make한 결과로 생성되는 Moran.c 파일을 보시면 아래와 같이 코드가 생성되어 있는 것을 알 수 있습니다.

```
char*   dictionary_path = (char *)SvPV_nolen(ST(0));
```

ST(0) 즉, Perl에서 initialize_moran_xs() 서브루틴을 호출할 때 넘기는 array의 첫번째 값에 대해서 저장되어 있는 실제 값을 (char*)로 넘겨 받는다는 의미입니다. Moran.xs를 코딩할때는 이런 부분을 상세하게 알아야하겠지만, C 코드처럼 작성해도 자동변환이 된다는 것을 알 수 있습니다.

<참고>

* 메모리 주소를 리턴하는 방법은 perlguts 문서를 참조하면 됩니다.
  - IV : signed integer value (integer 뿐만 아니라 메모리 주소를 저장한 만한 충분한 공간)
  - initialize_moran()으로 얻어진 메모리 주소값을 IV에 저장하고 이를 Perl로 리턴

```
SV*
initialize_moran_xs(dictionary_path)
    char* dictionary_path
    CODE:
        IV addr_holder;
        addr_holder = (IV)initialize_moran(dictionary_path);
        RETVAL = newSViv(addr_holder);
    OUTPUT:
        RETVAL
```

이제 초기화 루틴을 완성했습니다. 다음으로 사전을 해제하는 루틴은 아래와 같이 간단히 작성할 수 있습니다.

```
void
finalize_moran_xs()
    CODE:
        if(dict != NULL) {
            finalize_moran(dict);
            dict = NULL;
        }
```

마지막으로 analyze_moran()에 대한 서브루틴을 작성하면 아래와 같습니다.

<참고>

- SvCUR(SV*) : SV에 저장된 스트링의 실제 길이
- XSRETURN() : perlapi 참조
- newSVpvf() : perlguts 참조

```
SV*
analyze_moran_xs(sv_string)
    SV* sv_string
    CODE:
    char* string;
    char* rst;
    if( SvCUR(sv_string) != 0 ) {
        string = (char *)SvPV_nolen(sv_string);
```

```
  } else {
     RETVAL = (SV*)0;  /* return undef value */
     XSRETURN(1);      /* return되는 값은 stack에 저장되는데 몇개의 item이 있는지 명시 */
  }

  rst = analyze_moran(dict, string);
  if( rst == NULL ) {
     RETVAL = (SV*)0;
     XSRETURN(1);
  } else {
     RETVAL = newSVpvf("%s",rst);
     free(rst);
  }
 OUTPUT:
    RETVAL
```

이제 Moran.xs를 저장하고 make해서 정상적으로 컴파일되는지 확인해봅니다.

$ make

문제가 없다면 기본적으로 perl에서 사용할 준비는 마쳤다고 볼 수 있습니다.

```
use ExtUtils::testlib;   # adds blib/* directories to @INC
use Moran;

initialize_moran_xs($dictionary_path);
$rst = analyze_moran_xs($string);
finalize_moran_xs();
```

5. Moran.pm

Moran.xs에 정의된 서브루틴들을 다시한번 Perl에서 사용하기 편하게 패키징을 하면 사용 및 배포가 용이해집니다. lib/Moran.pm 파일을 열어서 대략 아래와 같이 작성합니다.

```
package Moran;

use 5.010001;
use strict;
use warnings;
```

```
require Exporter;

our @ISA = qw(Exporter);

# Items to export into callers namespace by default. Note: do not export
# names by default without a very good reason. Use EXPORT_OK instead.
# Do not simply export all your public functions/methods/constants.

# This allows declaration   use Moran ':all';
# If you do not need this, moving things directly into @EXPORT or @EXPORT_OK
# will save memory.
our %EXPORT_TAGS = ( 'all' => [ qw(

) ] );

our @EXPORT_OK = ( @{ $EXPORT_TAGS{'all'} } );

# export할 서브루틴의 이름을 기술합니다.
our @EXPORT = qw(
    initialize_moran finalize_moran analyze_moran
);

# Moran.pm의 버전 정보를 관리합니다.
our $VERSION = '1.00';

require XSLoader;
XSLoader::load('Moran', $VERSION);

# Preloaded methods go here.

sub initialize_moran {
    my $dictionary_path = shift;
    # …. some error handling
    initialize_moran_xs($dictionary_path);
}

sub finalize_moran {
    finalize_moran_xs();
}

sub analyze_moran {
    my $string = shift;
    # …. some error handling
    return analyze_moran_xs($string);
```

```
}


1;
__END__

=head1 NAME

Moran - Perl extension for Moran analyzer written in C

=head1 SYNOPSIS

  use Moran; # imports initialize_moran, analyze_moran, finalize_moran

  # simple and fast interfaces
  initialize_moran($dictionary_path);
  $rst = analyze_moran($string);
  finalize_moran();
  ……..
```

현재 Moran 디렉토리에서 'make install'을 하면 Moran.pm이 설치되는데, 위와 같이 documentation을 잘 해두면 ‘perldoc Moran’ 명령으로 쉽게 사용법을 찾아볼 수 있게 됩니다.

6. test_moran.pl

설치된 Moran.pm을 사용하는 방법은 아래와 같이 간단합니다.

```
#!/usr/bin/perl -w
use strict;
use warnings;
use utf8;

….

use ExtUtils::testlib;   # adds blib/* directories to @INC
use DHA;
….

initialize_moran($dictionary_path);
$rst = analyze_moran($string);
print $rst , “\n”;
```

finalize_moran();

---

7. FCGI with Perl

형태소분석기와 같이 초기화가 무거운 모듈을 API로 서비스할때, FCGI를 자주 사용하는데, Perl에서는 어떻게 사용하는지 살펴봅시다. 우선 시스템에 apache와 fcgi가 설치되어 있다고 가정합니다.

$ rpm -qa | grep -e 'httpd' -e 'fcgi' -e 'fast'
fcgi-devel-2.4.0-10.el5.x86_64
httpd-tools-2.2.15-15.el6.x86_64
mod_fastcgi-2.4.6-1.el5.rf.x86_64
httpd-2.2.15-15.el6.x86_64
fcgi-2.4.0-10.el5.x86_64

FCGI를 사용하기 위해서는 CGI::Fast 패키지가 설치되어 있어야합니다. 이를 설치한 이후 API는 아래와 같은 형태로 만들어질 수 있습니다.

<참고> CGI::Fast

```
#!/usr/bin/perl
use strict;
use warnings;
use utf8;
use CGI qw(:standard escape escapeHTML);
#use CGI::Carp qw(fatalsToBrowser);
use JSON;
use XML::Bare;
use Encode;
use CGI::Fast qw(:standard);
binmode STDOUT, ":encoding(UTF-8)";

use ExtUtils::testlib;  # adds blib/* directories to @INC
use Moran;

our $dictionary_path = q(/home/dictionary.dict);

initialize_moran($dictionary_path);

my $q;
while( $q = new CGI::Fast ) {
```

```
  CGI->compile();
  proc_cgi($q);
}

finalize_moran();

exit;

1;

sub proc_cgi {
  my $q = shift || (new CGI);
  $q->charset('utf-8');
  my $path_info = $q->path_info;
  my ($cmd, $path, $suffix) = fileparse($path_info, ".xml", ".json", ".txt", ".html", );
  $suffix = ".json" unless $suffix;
  my $callback = param('callback');
  my $mode     = param('mode') || "";
  my $query    = decode(utf8=>param('q')) || "";
  my $tag      = param('tag') || "";
  my $content_type = mime_type($suffix, $callback);
  my $full_url     = $q->url(-full=>1);
  my $absolute_url = $q->url(-absolute=>1);
  ….
  $header = $q->header(-charset=>'utf-8',
     -type=>$content_type,
     -expires=>'+3m',
     -cache_control => q(public, s-maxage=180),
    )
  ….
  print $header;
  ….
  $rst = anaylze_moran($query);
  ….
}
….
```

이렇게 만들어진 API를 FCGI 서버로 띄우기 위해서 httpd.conf에 설정을 해줘야하는데, 그 방법은 아래와 같습니다.

```
# FCGI
LoadModule fastcgi_module modules/mod_fastcgi.so
<IfModule mod_fastcgi.c>
```

```
    Alias /fcgi/ /home/wrapper/perl/www/
    <Directory /home/wrapper/perl/www/>
        SetHandler fastcgi-script
        Options +ExecCGI
        Allow from all
    </Directory>
    AppClass /home/trunk/wrapper/perl/www/fcgi.pl
</IfModule>
```

---

8. 마침

지금까지 C로 작성된 라이브러리를 Perl에서 사용하는 방법과 만들어진 Perl 패키지를 FCGI를 사용해서 API 서비스하는 방법에 대해 간략히 살펴봤습니다. 사실 Perl 개발은 이제 시작하는 단계라 초짜나 다름 없지만 비슷한 니즈가 있는 분들께 작게나마 도움이 되었으면 합니다. 감사합니다.