
Table of Contents

튜토리얼 소개	1.1
0. 웹 크롤링이란?	1.2
주의할 점?	1.2.1
1. 시작하기	1.3
2. 관련 자료모음	1.4
3. 페이지 분석하기	1.5
4. 개발환경 구축하기	1.6
5. 노코기리 튜토리얼	1.7
자주 사용하는 메소드	1.7.1
6. 빌드 및 확인 작업	1.8
7. 다른 공지 파싱 예제	1.9
8. 크롤링 여러번 하기	1.10
9. iframe 데이터 파싱	1.11
10. 인코딩 문제 처리	1.12
11. 대학교 식단 크롤링 예제	1.13
준비하기	1.13.1
설계 및 구현	1.13.2

Web Scraping with Nokogiri

Web Scraping(웹 크롤링) - 노코기리 잼을 이용하여 파싱해보기

이 문서는 웹 크롤링이라는 주제에 대해서 처음 접하는 분들을 위해 제작한 튜토리얼입니다.

작성을 하고 있는 필자도 그저 일반 대학생이기 때문에 튜토리얼을 만들어 배포하는 것이 건방지고 느껴질 수도 있지만, 많은 사람들이 크롤링에 관하여 궁금해 하고 정보를 얻는 것도 쉽지 않기 때문에 작성하게 되었습니다.

처음에 블로그를 통해서 배포를 한 후에 1000명 가까이 되는 분들이 관심을 가져주셔서, Gitbook에 몇몇 부분을 개정하여 다시 배포하려고 합니다. 이 튜토리얼을 통해서 배울 수 있는 것은 다음과 같습니다.

이 튜토리얼에서 배울 수 있는 것들?

1. 웹 크롤링 개념에 대해 알아볼 수 있다.
2. 다양한 언어로 접근할 수 있도록 기초 개념을 다진다.
3. Ruby on Rails의 사용법을 간단히 살펴볼 수 있다.
4. Nokogiri 잼을 이용하여 파싱하는 작업을 체험해볼 수 있다.
5. 개발환경 구축부터 서버 동작까지 모든 과정을 가볍게 다뤄볼 수 있다.
6. Crontab에 job을 등록하는 방법에 대해 배울 수 있다.
7. Nokogiri와 Rails로 파싱 작업을 할 때 자주 발생하는 버그나 이슈를 해결할 수 있다.

참고로 이 튜토리얼의 목적은 특정 언어의 특징이나 프레임워크의 장점을 소개하는 것은 아닙니다. 그러므로 "이런 방법이 있구나." 정도로만 알고 넘어가고 만약 선호하는 개발 언어나 툴이 있다면, 개념을 익히고 다른 방법으로 다시 접근해보시길 바랍니다.

문의 사항이나 관련 질문이 있다면, 아래 이메일로 문의하시면 바로 답변을 드리고 있습니다.

(sangjunpark0203@gmail.com)

그럼 시작합니다~ :D...

Web Scraping(웹 크롤링)이란?

컴퓨터 업계에서 일하는 사람들은 "크롤링", "크롤러", "크롤링 알바" 이런 말을 들어본 적이 있을 겁니다. 친구들이나 선배들이 회사에서 크롤링하는 알바를 했다는 경험담을 들은 적도 있죠. 저도 팀 프로젝트를 하면서 크롤링 할 일이 있었습니다. **하지만! 방법을 몰랐습니다.**

조언을 구하고 구글의 힘을 빌려서 파싱부터 데이터베이스에 넣는 작업까지 성공을 했습니다. 결과물이 잘 나와서 눈물이 앞을 가리더군요. 사람들에게 스크린샷을 찍어서 보내면서 내가 한거라고 $\pi\pi$;; 그 과정에서 학교 홈페이지를 다운 시키기도 했습니다. 죄송합니다... 3일 밤을 샜어요. 고생했어...아 논지를 벗어 났군요.

우리가 흔히 부르는 웹 크롤러 크롤링의 정식명칭은 **'Web Scraping'** 입니다.

외국 자료를 찾아보니까 'Web Crawling'보다는 'Web Scraping'이라는 용어를 자주 사용하더군요.

그래서 웹 크롤링이 뭔데?

"Web scraping is a computer software technique of extracting information from websites."

(웹 크롤링이란 컴퓨터 소프트웨어 기술로 웹 사이트들에서 원하는 정보를 추출하는 것을 의미합니다.)

보통 웹 크롤러란 인터넷에 있는 웹페이지를 방문해서 자료를 수집하는 일을 하는 프로그램을 말합니다. 이때 한 페이지만 방문하는 것이 아니라 그 페이지에 링크되어 있는 또 다른 페이지를 차례대로 방문하고 이처럼 링크를 따라 웹을 돌아다니는 모습이 마치 거미와 비슷하다고 해서 스파이더라고 부르기도 합니다. 엄청난 분량의 웹문서를 사람이 일일 구별해서 모으는 일은 불가능에 가깝습니다. 때문에 웹 문서 검색에서는 사람이 일일이 하는 대신 이를 자동으로 수행해 줍니다.

예를 하나 들어보죠.

멜론(www.melon.co.kr)에서 2016년 07월 27일 12:23분의 음원 차트가 필요하다고 가정해봅시다.

물론 사이트에 들어가서 일일이 확인을 하고 엑셀에 값을 입력하고 정리해서 상사에게 보고하는 사람들도 있을 겁니다.

하지만 다른 곳에 활용하려면 **DB 형태로 저장해 놓을 필요가 있습니다.**

카테고리/순위/발매일/제목/앨범제목/그룹이름 등등 차트에도 여러 정보가 있겠죠.

웹은 기본적으로 HTML 마크업 언어로 보여지는 걸 다들 아실 겁니다. 저희가 결과물을 눈으로 볼 수 있다면 해당 정보가 HTML 형태로 어떻게 구성되어 있는지도 확인할 수 있습니다. '페이지소스 보기' 또는 '개발자 검사'로 말이죠.(우클릭)

이런 소스들은 보통 개발자들이 어떤 정형화된 형태로 작성한 결과물입니다. 때문에 규칙이 생기죠. 이런 규칙을 분석해서 우리가 원하는 정보들만 뽑아오는 것을 **지금부터 하려는 웹 크롤링 작업**이라고 생각하시면 됩니다. 한국에서도 웹 크롤링 할 일이 많기 때문에 알바를 구하는 구인 광고도 본 적이 있고, 네이버, 다음, 구글 등등 여러 포털 사이트 블로그에 크롤링 하는 방법을 치면 나오는 정보들도 많습니다. 그 중 가장 많이 나오는 것이 파이썬으로 크롤링 하는 소스들이 제일 흔하죠.

저도 처음은 파이썬으로 크롤링 개념을 배웠습니다. 하지만 저는 파이썬보다는 현재 루비를 좋아해서 루비로 할 수 있는 방법을 찾아보았습니다. 지금부터 몇 차례에 걸쳐서 **'Ruby'와 'Rails' 그리고 'nokogiri'라는 잼을 이용하여 크롤링 하는 방법**에 대해서 이야기할 예정입니다.

한국에서 노코기리로 스크래핑하는 것에 대한 방법을 기술한 자료는 그렇게 많지 않았습니다. 일주일 내내 찾아도 제대로 된 것이 2개? 1개? 정도 였고 정보도 통틀어 10개를 넘지를 았았습니다. 그래서 루비와 노코기리로 크롤링 하는 방법을 찾는 분들을 위해서 노코기리 잼 분석부터 레일즈로 띄우는 방법까지 자세히 기술해보려고 합니다.

자세한 웹 크롤링에 대한 정의는 아래 위키 피디아를 참조하시기 바랍니다. 위키가 저보다 똑똑하니까요.

(https://en.wikipedia.org/wiki/Web_scraping)

그리고 외국 자료들이 더 정확하고 유용하니까 지금부터라도 '웹 크롤링', '웹 크롤러'로 검색하기 보다는 "Web Scraping with something" 이런 식으로 검색하여 원하는 정보를 찾아 활용하시기 바랍니다.

주의할 점?

아래는 관련 글에서 스크래핑을 할 때 몇 가지 주의할 점에 대해서 기술하여 가져왔습니다.

처음이신 분들은 꼭 읽어보시기 바랍니다.

A few scraping rules(크롤링 규칙?)

- You should check a site's terms and conditions before you scrape them. It's their data and they likely have some rules to govern it.

크롤링을 하기 전에 해당 페이지의 조건등을 살펴볼 것. 데이터는 그들의 것이며, 그 데이터들을 사용하기 위한 룰이 있을 수도 있다.

- Be nice - A computer will send web requests much quicker than a user can. Make sure you space out your requests a bit so that you don't hammer the site's server.

예의를 지켜라. 컴퓨터는 사용자가 하는 것보다 빠른 요청 신호를 웹 페이지로 보냅니다. 그러므로, 요청을 보낼 때에는 어느 정도 간격을 두고 할 것. 아니면, 웹 사이트에 부담을 줄 수 있습니다.

- Scrapers break - Sites change their layout all the time. If that happens, be prepared to rewrite your code.

페이지의 구조가 바뀌면, 해당 페이지를 긁는 코드의 구조 또한 바뀌어야 합니다. 한 번 분석해서 구현했다고 해도 주기적으로 확인을 해야합니다.

- Web pages are inconsistent - There's sometimes some manual clean up that has to happen even after you've gotten your data.

웹 페이지는 고정되어 있지 않습니다. 당신이 데이터를 한 번 가져왔다고 해도, 데이터를 긁은 후에도 구조나 데이터가 변할 수 있습니다.

시작하기

지금부터 루비온레일즈라는 프레임워크를 사용해서 웹 크롤링하는 방법에 대해서 공부해볼 겁니다.

참고로 루비(Ruby) 언어와 Ruby on rails(루비온레일즈)를 사용하지만, 전체적인 크롤링 개념은 다루고 있으므로 다른 언어로 공부하고 싶으신 분들에게도 도움이 될거라고 생각합니다.

Q : "저는 웹 크롤링을 하고 싶습니다. 뭐부터 하면 되나요?"

A : "음... 선택지를 줄게 골라봐."

첫 번째 해야할 일이 무엇일까?

1. "웹 크롤링하는 방법", "Web scraping source file"을 검색해서 본다.
2. 크롤링을 할 줄 아는 친구에게 밥을 사주고 부탁한다.
3. 크롤링 알바를 돈을 주고 고용한다.
4. 웹 크롤링을 하려는 페이지를 분석한다.
5. 문법을 알아야하니까 루비나 파이썬 서적을 사서 언어를 처음부터 공부하고 생각한다.

저도 처음에는 1번을 선택했습니다. 그 다음에는 5번 그리고 2번을 하려다 4번이라는 결론에 도달했습니다.

웹 크롤링이나 스크래핑에 관하여 질문을 던지면 보통 비슷한 답변이 돌아왔습니다.

선배1 : (수집하려는 페이지 소스를 보고...) "야 겁나 쉽네! 이렇게 저렇게 하면 끝! 쉽네! 술이나 먹자!"

=ㅅ=;;; 그렇게 몇 명을 찾아다니다가 이런 말을 들었습니다.

"웹 크롤링이라고 하는 작업은 정형화된 일이 아니고, 페이지 마다 다르기 때문에 먼저 수집하려는 페이지를 분석해봐. 어떤 규칙을 통해 짜여있는지 그리고 원하는 정보는 어떻게 구성되어 있는지를 분석하는 것부터 시작해봐."

그래서 저는 우선 원하는 정보를 나열하고 페이지를 뜯어서 분석하기 시작했습니다. 제가 생각하는 정답은 4번이었고 직접하고 싶다면 페이지 분석이 첫번째입니다. 개념없는 사람이 대충 짠 코드가 아니라면 페이지의 태그 구성은 잘 짜여있어서 추출하기도 편할 겁니다.

그럼 다음 챕터부터는 '동국대학교 공지사항'을 예시로 해서 페이지 분석을 시작해보겠습니다.

관련 자료모음

본격적으로 튜토리얼을 시작하기 전에 아래 자료들을 한 번 공부해보길 추천합니다. 대부분 영어로 된 사이트들이지만, 영어가 더 좋은(?) 사람들에게는 이게 익숙할 수도 있으니까 말입니다. Nokogiri로 Web Scraping하는 방법을 공부할 때 참고했던 사이트 목록입니다. 저는 반 강제적으로 한국 블로그 포스팅이 도움이 안되서 외국 자료를 볼 수 밖에 없었습니다.

1. Data Scraping and More with Ruby Nokogiri Sinatra and ...

Nokogiri gem을 이용해서 콘서트 티켓 예매 사이트를 크롤링하는 예제이며, Sinatra로 간단히 서버를 띄우고 자료를 확인하는 것 까지 자세히 나와있습니다. 저도 따라해 봤는데 예시로 사용하고 있는 사이트 구조가 조금 변해서 현재 상황에 맞게 대처하면서 활용했습니다.

(<https://www.youtube.com/watch?v=PWI1PIvy4A8&list=PLnr-v0JC93835JNvgY6o-aXWNjEGpDqQo>)

2. Parsing HTML with Nokogiri | The Bastard Book of Ruby

Nokogiri로 HTML 태그들을 분석하고 파싱하는 방법에 대해서 간략히 기술해 놓은 자료입니다.

(<http://ruby.bastardsbook.com/chapters/html-parsing/>)

3. Nokogiri Tutorial

노코기리 잼을 만든 팀이 직접 기술한 튜토리얼 자료입니다. 물론 모든 문법에 대해서 다루고 있지는 않습니다. 하지만 차근차근 따라가보면 XML, HTML을 파싱하는 방법과 .css/.xpath 메소드를 어떻게 사용하는지에 대해서 개념을 익힐 수 있을 겁니다. 이런 작업이 익숙한 분들은 이 사이트만 참고해도 노코기리를 잘 사용할 수 있을 겁니다. 원래는 노코기리가 지금 우리가 하려는 작업같은 곳에 사용하려고 만든 건 아닙니다. 여러 잼들과 레일즈 자체에서도 기본으로 취급하고 있는 라이브러리로 광범위하게 사용되고 있다는 점도 알아두시면 좋습니다.

(<http://www.nokogiri.org>)

4. Nokogiri를 이용한 게시판 파싱하기(썸미의 티스토리 블로그)

이 블로그는 한국어로 된 블로그 포스팅인데, JSon 형태로 뽑는 것 까지 설명을 하고 있습니다. 글을 자세히 읽어야하지만 꽤나 도움이 됐던 포스팅입니다. 설치나 환경 설정을 마스터한 사람이라면 이 블로그를 참고하셔도 좋습니다.

(<http://susemi99.tistory.com/1482>)

그 이외의 사이트 목록

(<http://www.58bits.com/blog/2012/06/13/getting-started-nokogiri-xml-ruby>)

(<https://www.distilled.net/resources/web-scraping-with-ruby-and-nokogiri-for-beginners/>)

3. 페이지 분석하기

이번에는 수집 대상 페이지를 분석하는 작업을 진행할겁니다. 여기서 예시로 사용할 페이지는 동국대학교 공식 홈페이지의 공지사항입니다. 우선 동국대학교 홈페이지 (<http://www.dongguk.edu/mbs/kr/index.jsp>)에 접속해 봅시다. 주소를 잠깐 살펴보면 끝이 (.jsp)라고 되어있죠? 이 말은 이 페이지가 JSP를 사용하여 만들어졌다는 걸 암시합니다. 그냥 참고로 알아두세요.

그럼 다음과 같은 페이지가 나옵니다. 첫 페이지에 크게 7개의 카테고리로 공지사항을 제공하고 있는 걸 확인할 수 있습니다. 각각의 페이지들을 살펴보면 구조가 같고 URL과 내용만 다르기 때문에 저는 '학사' 페이지를 분석을 해보겠습니다.

일반	학사	장학	입학	취업	국제	학술
의료기기산업학과 행정조교 모집						01-22
[경영학과] 경영학과 행정조교 모집안내						01-22
[DCTL] 설문조사 추가 당첨자 발표						01-22
[교수학습개발센터] 학생이 생각하는 "좋은...						01-20
[대학미디어센터] 2016년도 1학기 대학원신...						01-20
[경영대학 교학팀] 행정조교모집						01-20
[경영학과] 김현동 교수님 교육조교 모집 공...						01-20
[취업] 2016년 공채대비 자기소개서 작성 ...						01-20

대상 페이지의 URL 분석하기

대상 페이지가 하나라면 상관없지만 2~100 페이지를 분석하려고 한다면 여러 페이지에 접속해야겠죠?

우선 학사 페이지를 클릭해서 들어가 봅시다. 첫 페이지의 URL을 보면 다음과 같습니다.

- "www.dongguk.edu/mbs/kr/jsp/board/list.jsp?boardId=3638&id=kr_010801000000"

그렇다면 다음 페이지는 어떨까요?

- "생략...&command=list&id=kr_010801000000&mcategoryId=0&spage=2"

동국대학교 정도의 학교 공식 홈페이지가 이상하군요. URL이 페이지 하나 달라졌다고 이렇게 달라지면 어찌라구요? 괜찮습니다. 첫 페이지는 짧게 표시가 되지만, 페이지네이션하는 방식에 따라 URL이 다르게 표현될 수 있는 겁니다. 여러 URL이 사실 하나의 같은 페이지로 향하고 있다면 이해가 가는 부분입니다. 이걸 증명하려면 현재 상황에서 다시 1페이지를 클릭해서 돌아가면 확인이 가능합니다.

- "생략...&command=list&id=kr_010801000000&mcategoryId=0&spage=1"

다시 길지만 끝 부분이 "space=1"이 된 걸 확인하실 수 있습니다. 그렇다면 3 페이지로 가려면 끝 부분을 3으로 바꿔주면 되겠다는 생각이 드시나요? 안 든다면, **HTTP 프로토콜 개념에 대해서 공부하고 오시는 걸 추천합니다.** 이런 방법으로 우리는 **N 페이지에 URL GET Request** 방식으로 접근할 수 있다는 지식을 얻었습니다.

추출하려는 데이터 구조 설계하기

학사공지

홈 > 대학안내 > 공지사항 > 학사공지

전체

수업,성적

학적

국내교류

프로그램및특강

계절학기

기타

RSS

제목

검색

번호	제목	작성자	작성일	조회	파일
	2016학년도 1학기 한양대학교 수학안내	탁상민	2016-01-22	105	
	2015학년도 2학기 현장실습 학점인정 결과 안내	김홍희	2016-01-22	197	

위 화면을 살펴보면 공지사항이라는 게시판에서 추출할 수 있는 데이터 목록이 크게 몇 가지로 추려지는 걸 알 수 있습니다. '번호', '제목', '작성자', '작성일', '조회', '파일' 총 6개로 나누어 볼 수 있습니다.




하지만 우리는 좀 다르게 보고 4개만 선정해보겠습니다.

1. 제목
2. 제목에 연결되어 있는 링크(URL)
3. 작성자
4. 작성일

어라? 6개에는 링크는 없는데 어떻게 추출할까요? 다 방법이 있습니다. 바로 페이지 소스 파일에서 그 정보를 얻을 수 있습니다. 페이지 소스는 해당 페이지에서 **우클릭 후 '페이지 소스보기'** 또는 **'검사'**를 통해서 확인 가능합니다.

추출 대상 페이지 분석하기

그런데 시작하기도 전에 문제가 하나 발생합니다. 다음 사진을 자세히 보세요.

	2016학년도 1학기 한양대학교 수학안내	탁상민	2016-01-22	105	
4472	[국내교류] 2016학년도 1학기 한양대학교 수학안내 	탁상민	2016-01-22	105	

두 공지는 같은 페이지에 존재하는 두 글입니다. 하지만 사실상 같은 글이죠. 클릭을 하면 같은 페이지로 이동합니다. 게시판을 보는 입장에서는 그냥 '!' 표시를 하면서 중요 공지 표시를 하려는구나 하고 넘어가면 되지만 이걸 구현하는 개발자 입장에서는 골치아픈 요소 중 하나입니다. 페이지 소스를 열어봐도 역시나 저 두 글이 모두 나와있고 테이블 구조도 비슷해 보입니다. 만약 이대로 진행을 해버리면 DB에서 [국내교류] 라는 것 스트링 차이때문에 다른 글로 인식하고 중복된 데이터가 들어가게 됩니다. 물론 둘 다 수집할 필요가 있다면 상관없지만 우선은 중복이 되면 안된다고 가정하고 진행해보겠습니다. **분명 소스 상에는 이 두 글의 차이가 있을 겁니다.**

자세히 살펴보면 두 글의 차이는... 하나는 '!' 표시가 있고 글 번호가 없습니다. 반면에 다른 하나는 글 번호가 4472. 그리고 [국내교류]가 맨 앞에 붙어있고 우측에 새로운 글 'N' 표시도 있습니다. 이걸로 구분을 하면 될 것 같습니다. 그럼 이러한 차이점을 염두해두고 소스 파일을 살펴보겠습니다. 다시 한번 우클릭 후 페이지 소스파일로 이동합니다.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>...</head>
<body>
  <ul>...</ul>
  <div>...</div>
  ...
  <table id="board_list" summary="...">
    <colgroup>...</colgroup>
    <thead>...</thead>
    <tbody>
      </tbody>
  ...
```

전부 가져올 수 없기 때문에 큰 구조만 가져왔습니다. 뭐 대충 이런식으로 시작합니다. 하지만 추출하고 싶은 내용들은 안에 있었습니다. 그렇다면 우리가 접근해야할 가장 첫 번째 위치는 입니다. 감사하게도 해당 페이지 소스에서 는 단 하나밖에 없습니다. 그러므로 를 찾으라는 명령을 내리면 바로 우리가 원하는 위치로 갈 수 있다는 겁니다. 이런식으로 처음에 할 일은 원하는 내용의 위치를 파악하는 일입니다. 그리고 세부적으로 분석해 나가는 거죠. 이제는 안의 내용을 살펴봅시다. 우선은 '!'이 표시가 되어있는 알림 공지부터 살펴보겠습니다.

```
<tr>
  <td>
    
  </td>
  <td class="title">
    <a href="url주소">2016학년도 1학기 한양대학교 수학안내</a>
  </td>
  <td>탁상민</td>
  <td>2016-01-22</td>
  <td>105</td>
  <td>
    
  </td>
</tr>
```

처음에 '!' 표시는 이미지 태그로 그리고 첨부된 파일도 이미지 태그로 표시되어 있습니다. 단순한 문자가 아니었네요? 참고로 첨부된 파일이 없을 때에는 그냥 "-" 이렇게 표시가 되어 있습니다. 다시 언급하지만 알림 공지(!)는 중복 데이터이므로 수집하면 안됩니다. 이번에는 일반적인 공지의 구조를 살펴보겠습니다.

```
<tr>
  <td>4472</td>
  <td class="title">
    <a href="url주소">2016학년도 1학기 한양대학교 수학안내</a>
    
  </td>
  <td>탁상민</td>
  <td>2016-01-22</td>
  <td>105</td>
  <td>
    
  </td>
</tr>
```

위에서 살펴 보았듯이 두 글에는 몇 가지 차이가 있습니다. 방법은 여러가지가 있겠지만, 여기서 저는 첫 번째 에 있는 값의 형태로 구분을 해보겠습니다. 추출 대상이 되는 글은 글 번호로 시작합니다. 그리고 나머지를 분석한 결과를 한글로 풀어쓰면 다음과 같습니다.

- 제목 : title 클래스(class="title")가 있는 td 태그 안의 a 태그 속 텍스트
- 링크 : title 클래스(class="title")가 있는 td 태그 안의 a 태그안의 href에 할당된 텍스트
- 작성자 : tr 태그 안에서 세 번째 td 태그 안에 있는 텍스트
- 작성일 : tr 태그 안에서 네 번째 td 태그 안에 있는 텍스트

이렇게 페이지 분석이 끝났습니다. **이 걸 가져오겠다! 마음만 먹은 걸로 충분합니다.** 이제 "무엇을?"은 끝났으니 "어떻게?"를 살펴 보겠습니다. 본격적으로 'Nokogiri gem'을 이용하여 정보를 수집하는 방법에 대해서 알아보겠습니다.

4. 개발환경 구축하기

이번에는 본격적으로 Nokogiri Gem을 이용해보도록 하겠습니다. 앞의 내용이 이론같은 느낌이라 필요 없을 거라고 생각하는 사람들도 있겠지만, 저는 개인적으로 무척이나 중요한 개념이라고 생각합니다. Ruby on rails를 구축하는 방법도 가볍게 다루겠지만, 루비온레일즈 튜토리얼이 아니므로 그건 제 블로그(blgo.naver.com/potter777777)나 구글을 참고하시기 바랍니다.

여기서는 노코기리가 무엇인지 그리고 어떻게 설치하고 사용하는지에 대해서 살펴보겠습니다. 참고로 개발 환경은 맥 OS를 기준으로 하겠습니다. 리눅스도 별 차이가 없으므로 UNIX 계열을 사용하고 계신 분들이라면 따라하시면 됩니다.

왜 노코기리(Nokogiri)를 사용하는가?

보통 스크립트 언어마다 파서와 관련된 라이브러리가 존재합니다. Python은 Scrapy, BeautifulSoup 같은 것이 있습니다. 반면에 루비는 Nokogiri를 가장 많이 사용합니다. HTTP 프로토콜 관련해서는 Open-uri나 mechanize 등 여러가지가 있지만 파싱은 결국 노코기리를 사용합니다. Nokogiri 같은 라이브러리는 마크업 언어나 특정 구조를 분석하고 파싱하는데 사용됩니다. 이전에도 언급했지만 원래 이런 식으로 크롤링 하라고 만든 라이브러리는 아닙니다. 자체적으로 루비라는 언어를 설치할 때 함께 깔리는 걸 봤을 때도 컴파일을 할 때나 스트링 관련 문법을 적용할 때에도 활용되고 있습니다.

Nokogiri 공식 Github에 가서 README 파일을 읽어보면 다음과 내용이 쓰여있습니다.

"Nokogiri parses and searches XML/HTML very quickly, and also has correctly implemented CSS3 selector support as well as Xpath 1.0 support." ("노코기리는 XML/HTML 파일을 빠른 속도로 검색 및 파싱할 수 있습니다. 또한 CSS 셀렉터와 Xpath 기능을 지원합니다.")

즉, CSS로 이루어진 HTML 태그들을 잘 분석해서 원하는 내용으로 찾아갈 수 있도록 메소드들이 구성되어있고, 원하는 방식으로 재구성하여 추출 및 수정 활용을 가능하게 해주는 도구라고 생각하시면 됩니다.

Nokogiri Gem 설치

- "http://www.nokogiri.org/tutorials/installing_nokogiri.html"

위 링크는 노코기리 잼에 대한 튜토리얼 사이트입니다. 이 곳에는 libxml2 and libxslt로 설치하는 방법을 가르쳐줍니다. 저는 rvm과 기본 system으로 설치하다가 실패한 경험이 있어서

위 설치방법은 추천하지 않습니다. 하지만 물론 해결 방법은 존재합니다. rvm도 가상 개발환경을 제공한다는 점에서 훌륭한 도구입니다.

대신에 homebrew(rbenv)를 사용해서 잼을 설치하시기 바랍니다.

우선 노코기리 최신 안정 버전을 설치합니다. 아래 명령어를 치면 됩니다. 간단하죠?

```
$ gem install nokogori
```

이렇게 노코기리 잼 설치가 끝났습니다.

Rails 개발환경 구축하기

물론 단순히 Ruby 코드(.rb)와 노코기리만으로도 크롤링을 할 수 있습니다. 하지만 우리에게 MVC 모델 형태로 데이터를 쉽고 단순하게 다룰 수 있도록 도와주는 프레임워크가 있습니다. 서버의 부담면에서 차이가 있지만 우선 그 문제는 넘어가도록 하죠. 그래서 저는 레일즈 프로젝트를 생성하고 데이터베이스에 INSERT 및 COMMIT하는 방법까지 소개하려고 합니다.

```
$ mkdir workspace
#작업을 할 공간을 만듭니다.

$ cd workspace
#해당 디렉토리로 옮깁니다.

$ rails new scraping_notices
#레일즈 프로젝트를 생성합니다.

$ cd scraping_notices/
#레일즈 프로젝트 디렉토리로 옮깁니다.

$ sublime .
#해당 프로젝트를 서브라임 텍스트로 엽니다.
```

"sublime ." 이 명령어는 해당 디렉토리를 서브라임 텍스트에서 연다는 의미입니다. 그냥 설치만 하셨다면, 이런 명령어는 사용하기 어려울 겁니다. Path 설정이 필요합니다. 그건 블로그를 참조하거나 구글링 해서 해결하시기 바랍니다. 서브라임을 사용하지 않는다면 그냥 vi . 혹은 notepad로도 작업이 가능합니다. 그럼 코딩할 준비가 되었습니다. 이제 모델을 하나 만들어 보겠습니다. 저는 여기서는 간단하게 Scaffold 기능을 사용하여 빠르게 만들겠습니다. Scaffold(스캐폴딩)에 관한 설명도 따로 블로그나 구글을 참조하시기 바랍니다.

Notice 모델링

저는 이전에 네 가지를 추출한다고 말씀드렸습니다. 제목, 링크주소, 작성자, 작성일

그럼 위와 맞도록 Rails에 Model(notice.rb)을 만들어 보겠습니다.

*notice : title(String), link(String), writer(String), created_on(date)

여기서는 빠른 진행을 위해서 Scaffold를 사용하겠습니다. 되도록이면 스캐폴딩은 급할 때가 아니라면 사용하지 않는 것을 추천합니다. 이걸 생성하는 명령어는 다음과 같습니다.

```
$ rails generate scaffold notice title:string link:string writer:string created_on:date
#위에 정한 대로 스캐폴딩 기법으로 모델링을 구축합니다.
```

```
$ rake db:migrate
#실제로 테이블을 만들어서 위의 스키마(Migration)를 적용시킵니다.
```

이렇게 모델이 생성되었습니다. 기본적인 틀은 모두 완성되었습니다. 여기서는 Rails 기본 DB인 SQLite를 사용하겠습니다.

잘 따라오셨다면, `scarping_notices/db/migrate/`에서 다음과 같은 소스 파일을 확인하실 수 있을 겁니다.

```
class CreateNotices < ActiveRecord::Migration
  def change
    create_table :notices do |t|
      t.string :title
      t.string :link
      t.string :writer
      t.date :created_on

      t.timestamps null: false
    end
  end
end
```

위에서 `t.timestamps` 부분은 Rails Scaffold에서 튜플 생성 시간과 수정 시간을 나타내기 위해서 자동으로 생성되는 부분이므로 지우고 진행하셔도 무방합니다. 이렇게 개발 준비가 완료되었습니다. 다음 chapter에서는 노코기리 문법에 대해서 살펴보겠습니다.

4. 노코기리 튜토리얼

그럼 이제 노코기리 잼에 대해서 알아보겠습니다.

튜토리얼에 사용될 소스는 실제 동국대학교 학사 공지사항입니다. 챕터 3에서 일반 공지사항 알람 공지(!)가 같은 내용을 담고 있어서 문제가 된다고 했는데, 이 곳에서 실제 소스를 살펴보면 그 문제도 다릅니다.

살펴볼 소스는 다음과 같습니다. 이와 같은 소스 파일을 보고 싶을 때는 원하는 페이지에서 우클릭 후 '소스파일 보기' 혹은 '검사'로 확인 가능합니다. 그리고 이 소스를 다른 텍스트 에디터같은 곳으로 옮겨서 보고 싶을 때는 '검사'에서 'HTML 형태로 복사'같은 기능을 사용해서 복사를 하거나 직접 복사해서 사용하시면 됩니다. 'HTML 형태로 복사' 같은 기능은 특정 태그 안의 내용들만 살펴보고 싶을 때 자주 사용되는 기능입니다.

```
<tr>
  <td>4472</td>
  <td class="title">
    <a href="view.jsp?spage=1&boardId=3638&boardSeq=14745726&id=kr_010801000000&column=&search=&categoryDepth=&mcategoryId=0">[국내교류]&nbsp;2016
    학년도 1학기 한양대학교 수학안내</a>
    
  </td>
  <td>탁상민</td>
  <td>2016-01-22</td>
  <td>126</td>
  <td>
    
  </td>
</tr>
```

위에 가져온 소스 파일은 하나의 게시물에 관련된 것을 가져온 겁니다. 그 것도 일반적인 공지의 경우를 가져온 겁니다. 다른 경우는 따로 다를 예정입니다. 조금만 기다려주세요. 그러면 다음에는 /app/modles/notice.rb/ 파일을 살펴보겠습니다.

```
class Notice < ActiveRecord::Base
end
```

여기서 DB에 입력하는 INSERT 방식을 보면 ORM 방식이라고 해서 다른언어에서 직접 소스에 넣어서 사용하는 SQL 쿼리 문장과 다른 것을 확인하실 수 있습니다. 이에 대한 내용은 아래 문서를 참고하시기 바랍니다.

(<https://www.coffeepowered.net/2009/01/23/mass-inserting-data-in-rails-without-killing-your-performance/>)

위 위의 공지 하나를 크롤링해오는 소스 파일을 구현해보면 다음과 같이 나옵니다.

```
require 'nokogiri'
require 'open-uri'

class Notice < ActiveRecord::Base
  validates :link, :uniqueness => true

  #학사공지
  url = "http://dongguk대학교 학사 공지 1 페이지이므로 생략"
  #노코기리 파서로 HTML 형태로 분류한 형태의 데이터를 data 변수에 저장한다.
  data = Nokogiri::HTML(open(url))
  #.css selector로 tbody 안의 tr 내용들만 가져온다.
  @notices = data.css('tbody tr')

  #각 tr을 for문으로 끝까지 반복하면서 작업을 진행한다.
  @notices.each do |notice|
    #첫 번째 요소가 숫자라면, 우리가 대상으로 하는 공지이므로 추출한다. 아니면 넘어간다.
    if notice.css('td')[0].text.strip =~ /\A\d+\z/
      Notice.create(
        #제목
        :title => notice.css('td.title a').text.strip,
        #링크
        :link => "http://www.dongguk.edu/mbs/kr/jsp/board/" + notice.css('td.title a')
      )[0]['href'].strip,
        #작성자
        :writer => notice.css('td')[2].text.strip,
        #작성일
        :created_on => notice.css('td')[3].text.strip
      )
    else
      next
    end
  end
end
```

직접쳐서 오타가 있을 수도 있지만 하나씩 살펴볼거니까 아래 내용에 집중해주세요.

require 'nokogiri'

```
require 'nokogiri'
require 'open-uri'
```

우선 nokogiri와 open-uri 잼을 사용하려면 이렇게 위에 선언을 해주거나 Gemfile 설정이 필요합니다. 이걸 그냥 보험상 해주고 넘어가시면 됩니다. 가끔 프로젝트에서 잼 설정을 잘못하시면, 잼이 없다고 나오니까 이렇게 미리 사용한다고 해놓은 겁니다. 물론 Gemfile에 포함 시키시고 `$bundle install` 명령을 내리시거나 자체적으로 잼을 설치하시고 그 환경에서 프로젝트를 만들고 나서 생각할 문제입니다. 잼이 뭔지 전혀 모르겠다면 제 블로그의 다른 글을 참고하시거나 구글링으로 해결하시기 바랍니다.

```
validates :link, :uniqueness => true
```

이 부분은 유니크 키 설정으로 중복처리를 검사한다고 선언한것을 의미합니다. 예를 들어서 같은 대상을 여러번 크롤링하는 명령을 내렸을 때 이미 같은 공지가 DB에 있으므로 이 건 무시한다라는 의미입니다.

```
data = Nokogiri::HTML(open(url))
```

여기서 url에는 동국대학교 공지 1페이지에 해당하는 경로가 쓰여있겠죠? 이 페이지에 open-uri가 접속해서 노코기리 파서가 HTML 형태를 기준으로 Tree 구조를 만들어서 data에 담는다는 의미입니다. 무슨 말인지 잘 모르겠다구요? 아래 소스를 보시죠.

```
<html>
  <head></head>
  <body>
    <div class="hello"></div>
  </body>
</html>
```

이 소스를 보면, html tag 안에 head tag 그리고 body tag가 있습니다. 그리고 그 안에 div tag가 있는데, 그 div tag는 "hello"라는 class로 지정되어 있습니다. 만약에 이러한 소스가 한 줄로 표현이 되어 있더라도 노코기리가 속성 하나하나를 파악하고 분석해서 어느 태그 안에 어떤 것들이 있고 어떤 속성이 있는지를 자체적으로 걸러내서 깔끔한 형태로 저장하는 작업을 도와줍니다. 그리고 역으로 노코기리를 활용해서 td tag를 모두 찾아라! class="hello"만 찾아라! 등의 명령을 내릴 수 있습니다.

```
@notices = data.css('tbody tr')
```

여기서는 .css 라는 놈이 보입니다. 메소드로 사용되고 있죠. 이 메소드에 대해 알아보겠습니다.

.css selector

.at_css와 .css가 있는데 .at_css는 하나의 값(보통 첫 번째 값)만 그리고 .css는 해당하는 모든 값을 가져옵니다. 위에 소스를 보면 .css를 사용하였습니다. 그 이유는 .css로 가져온 구조를 여러 방식으로 활용할 예정이기 때문입니다. 계층적인 html 구조가 있다면 위 처럼 띄어쓰기로 상위와 하위 요소를 구분할 수 있습니다. 그 이외에도 다음과 같이 사용이 가능합니다. css를 아시는 분들은 자신이 원하는 방식으로 응용하여 사용하시면 됩니다. 왜냐하면 class, id 등등 기본적인 html과 css에서 사용하고 있는 것들을 그대로 문법으로 활용하고 있기 때문입니다. 문법 예시를 몇 가지 살펴해보도록 하죠.

1. @doc.css('title') : "@doc에 있는 것들 중 모든 title tag를 가져온다."

아래는 위의 명령어에 대한 결과 예시입니다.

```
["<title>example 1</title>", "<title>example 2</title>"]
```

2. @doc.css('div.featured a') : "div tag 중 class가 featured인 곳 중 a tag 요소를 가져온다."
3. doc.css("div#footer_inner strong")[0] : div tag 중 id가 footer_inner인 곳 중 strong으로 감싸고 있는 부분을 추출한다."

자시해 보면 '.', '#', ' ' 이런 문자들이 보입니다. 위에서도 말씀드렸지만, css/stylesheet에서 실제로 사용되는 문법들입니다. 직관적으로 사용하기 위해서 nokogiri에서 만든 형태입니다. 때문에 이름도 css selector입니다.

이와 비슷하게 .xpath 라는 것도 있는데 xpath는 xml 파싱에 사용됩니다. 이 튜토리얼에서는 다루지 않겠습니다.

그럼 이제부터 세부적으로 접근해보겠습니다. tbody 안의 tr에 접근했죠? 위의 모든 값은 계층별로 접근이 가능합니다. 그러므로 for 문으로 각각의 td tag에 접근해보겠습니다. 그 전에 if 문에 있는 /\Ad+\z/라는 문장이 있습니다. 이건 문자를 의미하는데, 문자가 아닌 건 숫자라고 생각하고 숫자 인지를 판별할 때 사용하는 문장입니다. 주석을 보시면 아실 수 있을 겁니다.

```
if notice.css('td')[0].text.strip =~ /\Ad+\z/
  #something
else
  next
end
```

td tag의 첫 번째 요소를 text(string) 형태로 추출하는데, 특수 부호같은 것은 알아서 제거를 해주라는 의미로 .strip 을 사용합니다. 벗었다는 의미죠? 그리고 그렇게 걸러낸 스트링 값이 숫자라면 어떻게 하고 아니면 어떻게 하라는 문장입니다. next는 continue; 같은 의미로 해당 차례의 명령어를 수행하지 말고 다음 차례로 넘어가라는 의미입니다.

- .text는 해당 tag 안의 text값을 가져오는 것을 의미합니다.

- `.strip`은 값에서 `tab`이나 쓸대없는 `space` 등을 제거하는 기능을 합니다. 특히나 `.strip`은 상황에 따라서 확인을 하고 사용하시는게 좋습니다. 노코기리도 모든 상황에 100% 완벽하게 적용되지 않기 때문에 제거하지 않고 넘어가는 경우도 있습니다.

- `Notice.create()` : `Notice`는 모델이고, `create()`는 튜플 하나를 생성한다는 의미입니다. 즉 `row`가 하나 늘어나겠죠?
- `:title => xxx` : `Notice` 테이블에 `col` 값 중 하나인 `title` 속성에 `xxx`를 할당한다는 의미입니다.

그럼 나머지 소스를 살펴보도록 합시다.

```
:title => notice.css('td.title a').text.strip
#<td class="title"><a> 안에 있는 텍스트를 가져옵니다.

:link => "http://www.dongguk.edu/mbs/kr/jsp/board/" + notice.css('td.title a')[0]['href'].strip
#요소 검색으로 저장된 각 URL을 보면 위의 http 부분은 생략되어 있고 나머지 뒷 부분만 저장되어 있습니다.
#즉, 내부 로직으로 앞의 값을 자동으로 붙여서 해당 링크로 들어간다는 의미죠. 그래서 "" 부분을 더하여 저장합니다.
#<td class="title"><a> 안의 첫 번째 요소 중 [href] 값을 순수하게 가져옵니다.

writer => notice.css('td')[2].text.strip
#세 번째 <td> 안의 텍스트를 순수하게 가져옵니다.

created_on => notice.css('td')[3].text.strip
#네 번째 <td> 안의 텍스트를 순수하게 가져옵니다.
```

주석이 곧 설명입니다. 이것으로 노코기리 사용법에 대한 튜토리얼은 마칩니다. 다음 챕터에서는 이렇게 가져온 결과물을 직접 확인해보는 작업을 해보겠습니다. 추가적으로 필요한 사항이나 궁금한 사항들은 이메일나 블로그로 문의주시고 원문 서적을 보고 싶다면, 아래 링크의 내용을 참고하십시오.

(https://classic.scrapewiki.com/docs/ruby/ruby_css_guide/)

자주 사용하는 메소드

제가 지금까지 대상으로 따지면, 대충 20개 페이지 정도 크롤링 작업을 거쳤습니다. 그 중에서는 미흡한 부분도 많죠. 모든 페이지의 구조가 다르고 때로는 비슷한 구조도 보입니다. 하지만 제가 Ruby로 작업을 하다보니, 자주 사용하는 틀과 메소드들이 있어서 한 번 리뷰할 겸 적어보려고 합니다.

Nokogiri Parse

```
data = Nokogiri::HTML(open(url))
```

#url을 Nokogiri API를 이용하여 <HTML> 기준 파싱 작업을 합니다.

Ruby For

```
(0..6).each do |i| end
```

#특정 리스트 혹은 배열에서 index 0~6을 순서대로 접근합니다. i에 0~6이 차례대로 들어갑니다.

```
target.each do |t| end
```

#Iterator 패턴이라고 생각하시면 됩니다. C#의 Foreach 구문이라고 보셔도 됩니다.
#target이라는 리스트 객체를 순서대로 모두 접근합니다. 그 인자는 t로 순서대로 접근합니다.

```
#응용 버전
tbody.css('tr').each do |tr|
  tr.css('td').each do |td|
  end
end
```

#이런식으로 each 안에 each를 두어 사용하기도 합니다.

nil? Null? empty?

```
some.nil?
```

#이것은 irb에서 테스트해봤을 때, nil이 나오는 경우를 걸러냅니다.

```
some.empty?
```

#이것은 리스트 객체가 비어있는 경우를 의미합니다.
#위 두 가지를 헷갈리지 마시기 바랍니다. 하다보면, 익숙해질 겁니다.

" "(not a space)

#위에 쓴 문자가 뭔지 궁금하십니까? 저 문자는 빈칸(Space)일까요? 결론부터 말씀드리자면 아닙니다.
#가끔 컴퓨터 환경에 따라서 Tab이 다르게 표현될 경우가 있고 빈칸도 크기가 다릅니다.
#그러므로 빈칸으로 보이는 것도 제대로 코딩했는데 안되면, irb에서 결과물로 나온 것을 복사 붙여넣기 해서 사용해보기길.

숫자 판별

```
some.scan(/\d/).join('') 그리고 (something =~ /\A\d+\z/)
```

#위 메소드는 특정 텍스트(some)에서 숫자만 추출하는 방법입니다. 각각의 기능을 살펴보면, 결과적으로 숫자만 추출합니다.

#그래서 이 방법으로 텍스트 안에 숫자가 포함되어 있는지 등을 판별하거나 숫자가 한 곳에만 있다면 그 숫자를 추출할 수도 있습니다.

#something 부터는 숫자로만 이루어져있는지를 판별하는 방법입니다.

if elsif else end next break

#경우의 수가 많고, 불규칙할 경우가 있기 때문에 더러워 지더라도 조건문을 사용하는 경우가 생깁니다.
#물론 객체지향적으로 잘 설계한다면, 코드가 많아져도 깔끔한 코드를 유인할 수는 있습니다.
#Ruby에서 조건문의 위치를 사용합니다. end를 습관화 들이시기 바랍니다.
#next는 continue를 의미하고, break는 break입니다.

+=

#보통 i++같은 걸 많이 사용합니다. 하지만 Ruby에서는 for(int i=0;i<10;i++) 같은 형태로 사용하지 않습니다.
#i += 1 형태로 사용하는 것을 기억해두시기 바랍니다.

Json

```
require 'json'
```

```
JSON.generate({:some1 => "", :some2 => ""})
```

#require 'json' 이라고 해야합니다.

```
#JSON.generate({:some1 => "", :some2 => ""}) 형태로 만듭니다.
```

Array

```
@something = Array.new
@something << array_plus
#@something = Array.new 형태로 우선 배열을 하나 만듭니다.
#@something << array_plus 형태로 배열을 하나씩 늘려갑니다.
#이름은 배열이지만 리스트일 수도 있고 다른 형태일 수도 있습니다. 응용 가능합니다.
```

String 교체

```
something.gsub("교체하고 싶은 부분", "교체할 문자")
#위처럼 사용되며, ' '로도 가능하지만 상황에 따라 다르니 해보시기 바랍니다.
```

Split Method

```
something.split(">")
#'>'라는 문자를 기준으로 나눕니다. 이 것은 배열에 순서대로 저장됩니다.
something.split(">")[0]
#위 처럼 나눈 후에 첫 번째 값을 의미합니다.
```

Switch

```
#Ruby에서는 case와 when 형태로 사용합니다.
#case @from when /daum/ when /naver/ else end
#위와같은 형태로 사용합니다.
```

Model.delete_all

```
#위 문법은 해당 모델 테이블에 있는 모든 튜플을 리셋합니다.
```


4. 페이지 빌드 및 확인 작업

지금까지 데이터베이스에 넣는 작업까지 노코기리 튜토리얼에서 살펴보았습니다.

그럼 이제, 실제로 레일즈로 페이지를 띄워서 우리가 얻은 자료들이 잘 추출되었는지 확인해 보겠습니다. 이 곳에 포스팅되는 내용은 루비온레일즈와 관련이 있으므로 자세한 설명은 생략하겠습니다.

먼저 라우터 설정으로 첫 페이지를 #index로 바꿔보겠습니다. /config/routes.rb/로 가서 #root 'welcome#index'의 주석을 풀고 root 'notices#index'로 루트 페이지를 조정합니다. 이유는 그냥 첫 페이지에 올린 내용을 바로 확인하고 싶어서 그렇습니다.

```
Rails.application.routes.draw do
  resources :notices
  root 'notices#index'
  #이하 생략
```

이번에는 화면을 표시해줄 View를 살펴보겠습니다. Scaffold로 기본적인 클래스 및 메소드들이 모두 설정이 되어있고 Views도 모두 설정이 되어있습니다. 그래서 따로 조정하지 않겠습니다. 조금 멋이 없지만 지금은 크롤링 결과물 내용이 더 중요하므로 원래 있던 형태를 최대한 활용하겠습니다. 그럼 페이지를 빌드 해보겠습니다.

```
$ rails s
```

그리고 웹 브라우저에서 <http://localhost:3000/>로 들어가서 페이지를 확인합니다. 그럼 다음과 같은 화면이 나옵니다. 실제 동국대학교 학사 공지사항 1페이지와 비교해 보았을 때 일반 공지사항만 중복없이 잘 들어간 것을 확인할 수 있습니다.

Listing Notices		Link	Writer Created on
[국내교과] 2016학년도 1학기 학업성취도 수확안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14745726&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-22	Show Edit Destroy
[수업,성적] 2015학년도 2학기 학업성취도 학업진행 결과 안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14737850&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-22	Show Edit Destroy
[수업,성적] 공통교과목과 대학별 이수학기 변경 안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14708658&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-21	Show Edit Destroy
[학칙] [학업성취도] 2015-2학기 학업부진 상담 안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14701727&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-21	Show Edit Destroy
[국내교과] 2016학년도 1학기 국어대학교 수확안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14693877&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-21	Show Edit Destroy
[가사] 2016학년도 신입생을 위한 공인영어 Vision 세...	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14657526&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-20	Show Edit Destroy
[학칙] 2016학년도 1학기 복학 신청(2차) 안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14598048&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-19	Show Edit Destroy
[가사] [국제교과] 기원특목(IPP) 시험 안내 (1/29)	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14572768&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-18	Show Edit Destroy
[제학박기] [24차 학업성취도] 2015-겨울학기 학업진행가이드...	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14566447&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-18	Show Edit Destroy
[제학박기] 2016학년도 1학기 경제대학교 수확안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14557959&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-18	Show Edit Destroy
[제학박기] [23차 학업성취도] 2015-겨울학기 학업진행가이드(...	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14554245&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-16	Show Edit Destroy
[수업,성적] 2015학년도 2학기 학업부진학생 상담 안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14505162&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-15	Show Edit Destroy
[제학박기] [22차 학업성취도] 2015-겨울학기 학업진행가이드(...	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14477291&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-15	Show Edit Destroy
[수업,성적] 2016-1학기 희망양자선생원 안내	http://www.dongguk.edu/mba/kr/jsp/board/view.jsp?spage=1&boardid=3638&boardSeq=14470335&id=kr_010801000000&column=&search=&categoryDepth=&mcate...	2016-01-15	Show Edit Destroy

notice.rb 소스에서 DB에 직접 넣는 기능을 구현했기 때문에 모두 DB에 들어가 있습니다. 그리고 Scaffold 만들어서 새로 추가하거나 수정 및 삭제가 가능합니다.

기본적으로 원하는 페이지를 선택하고 분석하여 가져오고 원하는 방식으로 바꿔서 활용하는 것까지 전체 내용을 전부 다루었습니다. 다른 방식으로 하는 방법도 있지만 큰 틀은 비슷하므로 개념을 익히고 넘어간다는 생각으로 접근하시는 것이 좋을 것 같습니다. 제가 처음 크롤링을 하고 싶었을 때 무작정 검색만하다가 포기를 했던 기억이 있습니다. 지금은 여러 공부를 병행하고 있는데 지금 생각해보면 그리 어렵지 않은 내용이었습니다. 다른 언어로도 마찬가지입니다. 이제부터는 크롤링을 하는 친구가 있다면, 대단하다면서 신기하게 보지만 말고 직접 할 수 있는 사람이 되었으면 좋겠습니다. 저와 똑같은 삽질은 더이상 하지 않게 하고 싶어서 이 글을 썼습니다. 이렇게 노코기리로 웹 스크래핑하기 튜토리얼을 마칩니다. 이 다음 챕터부터는 추가적인 이슈나 다른 예제를 만들었을 때 추가하겠습니다.

감사합니다.

다른 공지 파싱 예제

이번에는 조금 더 복잡한 페이지를 파싱해보도록 하겠습니다. 예제로 사용했던 '동국대학교 공지사항'보다 조금 더 복잡한 '동국대학교 각 대학별 공지사항'을 파싱해보겠습니다. 동국대학교 공과대학 홈페이지입니다.

먼저 주소는 (http://engineer.dongguk.edu/bbs/board.php?bo_table=en6_1&page=1)입니다.

테이블 찾기

'페이지 소스 보기'로 원하는 데이터가 어디에 있는지 찾아보겠습니다. 이 페이지도 데이터들은 역시나 테이블에 있습니다. 하지만 일반 공지사항과 다르게 구조가 조금 다릅니다. 그래서 "조금 더 복잡한 소스 파싱하기"라고 이름을 붙였습니다. 보통은 하나의 학교에서 여러 게시판을 만들면 구조를 깔끔하게 통일시키는게 좋은데 말이죠. =ㅅ= 마음에 안듭니다.

table tag를 검색하면 총 세개가 나옵니다. 그리고 마지막에 있는

이 제가 원하던 테이블입니다. 시작점은 이 곳으로 하겠습니다.

```
@notices = data.css('table')[2].css('tr')
```

데이터 찾기

table tag안에 데이터는 tr tag안에 있습니다. 그러나 이번 역시 각각 tr tag에도 공지알림용 공지사항이 있고 일반 공지가 있습니다. 이를 구분하는 방법은 첫 번째 td tag의 span tag안에 있는 값이 숫자인지로 판단할 수 있습니다.

```
if notice.css('td')[0].css('span').text.strip =~ /\A\d+\z/
```

세부 데이터 찾기

이번에 수집할 데이터는 제목, 링크, 작성자, 조회수, 작성일 입니다. 각각을 wr_title, wr_link, wr_writer, wr_hit, wr_created_on 라고 하겠습니다.

제가 이번 예제를 굳이 만든 이유는 tag라고 해서 모두 다 같은 tag가 아니라는 것을 알려드리고 싶어서 입니다. 페이지 소스를 보면 html tag, td tag, tr tag같은 기본 tag도 있지만 strong tag, nobr tag같이 꾸며주는 일시적인 태그들도 있습니다. 하지만 이런 것들까지 노코기리가 구분해주지는

않습니다. 큰 틀만 구조적으로 의미가 있다고 인식을 하는 것 같습니다. 그래서 tag가 아니라 그냥 String으로 처리가 됩니다.

```
<td align="left" style="word-break:break-all;">
  <nobr style="display:block; overflow:hidden;">
    <a href="../bbs/board.php?bo_table=en6_1&wr_id=338&page=1">2015-2학기 학점포
    기안내문</a>
    
  </nobr>
</td>
```

위 소스를 보면 안의 "2015-2학기 학점포기안내문" String을 가져오려면, td >> nobr >> a >> string 이렇게 접근을 해야합니다. 좀 길고 복잡해 보입니다. 하지만 노코기리에게 이 형태는 td >> a, img 이렇게 밖에 안보입니다. 즉, nobr은 무시해도 된다는 의미입니다. 단순히 CSS 형태에서 style을 정의해놓은 것에 불과하기 때문에 노코기리에게는 의미있는 태그가 아닙니다. 그렇기 때문에 위 내용에서 "2015-2학기 학점포기안내문"을 뽑으려면 다음과 같이 하면 됩니다. notice가 td tag까지 접근했다고 했을 때

```
notice.css('td')[1].css('a').text.strip
```

이는 irb로 확인해보면 이유를 알 수 있습니다. 각각을 nobr tag로 tbody tag로 table tag로 img tag로 파싱한 것들을 짝 찾아보면 원하는 내용이 있는 것과 없는 것들이 있습니다. 위의 notice.css('td')[1].css('a').text.strip을 notice.css('td')[1].css('nobr a').text.strip이라고 해서 찾으면, 결과 배열 값이 [] 즉, 아무것도 나타나지 않는 것을 알 수 있습니다. 없는 객체를 찾으라고 명령한 셈이 되는 겁니다.

그 다음은 href의 링크를 가져온다고 해보겠습니다.

```
"http://engineer.dongguk.edu/" + notice.css('td')[1].css('a')[0]['href'].strip.sub(/../, "")
```

좀 복잡해 보이죠? 이 전 예제와 href안의 링크를 뽑는 방법은 같습니다. 하지만 이번에는 전체 링크가 아니라, 상대 경로를 제공하고 있기 때문에 ../를 없애버리고 앞에 "http:xxx" 부분을 추가해서 링크를 뽑았습니다. sub 메소드의 의미는 구글링으로 사용법을 찾길 바랍니다.

전체적으로 이전 예제와 다르게 css 메소드가 여러번 사용된 것을 볼 수 있습니다. 이전에 css 메소드로 뽑은 값을 다시 css 메소드로 파싱하여 가져오고 싶을 때는 계속 위와 같이 뒤에 붙여주면 됩니다. 왼쪽부터 순서대로 파싱이 적용됩니다.

```
//작성자 부분
<td align="center" width="110">
  <nobr style="display:block; overflow:hidden; width:105px;">
    <!---->
    <font class="event">
      <span class="member">공과대학</span>
    </font>
  </nobr>
</td>

//조회수 부분
<td width="50">
  <span style="font:normal 11px tahoma; color:#777777;"> 1047 </span>
</td>

//작성일 부분
<td width="80">
  <span style="font:normal 11px tahoma; color:#888888;">
    <strong> </strong>
    2015-09-01
  </span>
</td>
```

즉 위에 있는 nobr tag나 부분이나 strong tag같은 부분은 무시하고, 큰 그림인 td, tr, span만 보고 접근하면 쉽게 풀립니다. 이런식으로 적용한 소스를 살펴보면 아래와 같습니다.

```
url = "http://engineer.dongguk.edu/bbs/board.php?bo_table=en6_1&page=1"

data = Nokogiri::HTML(open(url))
@notices = data.css('table')[2].css('tr')
@notices.each do |notice|
  if notice.css('td')[0].css('span').text.strip =~ /\A\d+\z/
    Engineer.create(
      :wr_title => notice.css('td')[1].css('a').text.strip,
      :wr_link => "http://engineer.dongguk.edu/" + notice.css('td')[1].css('a')[0]['href'].strip.sub(/\.\/, ""),
      :wr_writer => notice.css('td')[2].css('span').text.strip,
      :wr_hit => notice.css('td')[3].css('span').text.strip,
      :wr_created_on => notice.css('td')[4].css('span').text.strip
    )
  else
    next
  end
end
```

튜토리얼을 잘 보셨다면, 위 소스의 의미가 무엇인지 금방 이해하실 수 있을거라고 생각하여 추가 설명은 생략하겠습니다.

크롤링 여러번 하기

이 제목이 무슨 말인지 이해가 잘 안되시죠? 그런데 사실 위의 예제들을 직접 실행시켜보셨다면 무슨 말인지 이해가 가실겁니다. 지금까지는 모든 크롤링 관련 소스파일을 **Model** 부분에서 처리했습니다. 그렇게 한 이유는 사실 따로 없습니다. 일부러 이 부분을 만들기 위해서 한 것도 있구요.

Controller 부분이 **Scaffold**로 가득차 있어서 그냥 그렇게 한 이유도 있습니다. 사실 크롤링 소스의 위치는 그렇게 중요하지 않습니다. 따로 **.rb** 파일을 만들어서 객체를 생성하고 돌려도 되고 어느 부분에서 돌리던 그건 자유입니다. 하지만 **MVC** 패턴을 위해서 만들어진 **Rails**에서 **Controller** 부분의 역할을 잘 생각해보면 크롤러를 좀 더 효율적으로 만들 수 있습니다. 이 챕터는 어떻게 보면 **Rails**의 **MVC** 패턴에 대해서 설명하는 부분입니다.

Model의 역할은 실제 객체들의 저장소인 **DB**의 내용을 정의하고 처리하는 부분입니다. 그렇기 때문에 중간에 변화가 일어나지 않는 이상 **Model**에서 내린 정의는 바뀌지 않아야 합니다. 만약에 이 프로젝트에서 중복 공지를 허용한다 하지 않는다를 정의하지 않고 시작했다면, 골치가 아파지겠죠? 그렇기 때문에 **Model** 부분은 전체 프로젝트를 빌드했을 때 한 번만 거칩니다. 즉 **이 전에 썼던 대로 소스를 만들어서 넣어버리면, 단 한번만 크롤링 작업이 수행된다는 의미입니다.** **Crontab**에 아무리 해당 **URL**로 **Get** 요청을 보내도, **Model**에서 수행한 작업이기 때문에 첫 페이지 빌드 시에만 작업을 수행하고 다음 부터는 **Controller**에서는 별 말이 없어서 **View**의 내용을 계속 반복해서 보여줄 뿐입니다. 실시간 반복 처리가 안됩니다.

그럼 이제부터는 **Controller**에서 처리하는 방법을 살펴보겠습니다.

```
#notices_controller.rb
def index
  #여기에 이전에 만든 소스를 넣어서 처리합니다.
end
```

왜 위처럼 처리했냐고 물어보신다면 **index** 페이지에 방문했을 때 해당 작업을 수행하고 싶기 때문입니다. 물론 방법은 여러가지 입니다. 하지만 페이지 방문 형태가 이해하기 쉽기 때문에 이렇게 합니다. 여기서 **일정 주기로 크롤링 작업을 수행하고 싶다면, 서버를 Background로 계속 띄워놓고 Get Request와 관련된 명령어를 Crontab Job으로 등록해놓으면 됩니다.** **Rails**에서는 이 또한 **Gem**으로 쉽게 작성할 수 있도록 도와주고 있는데요. **Whenever gem**이 바로 그것입니다. 이에 대한 내용은 블로그 내용을 참조하시거나 필요 시에는 이 곳에도 추가하겠습니다.

iframe 데이터에 접근하고 파싱하기

최근에 여러 대학교 캠퍼스의 식단을 파싱하는 일을 하고 있습니다. 그 이외의 학교도 분석하여 파싱하는 작업을 하고 있었는데, 특이한 케이스를 확인해서 글을 씁니다. 모바일 버전만 지원, GET 방식으로 URL을 지원하지 않는 학교, 사진으로 찍어서 올려놓는 학교, 누군가 직접 일일 식단을 매일 올리시는 학교(경희대 국제)등등... 여러 방식이 있었습니다. 그 중에서 한양대학교 학생 식당 식단표를 예제로 살펴보겠습니다. 우선 아래 사진을 살펴봅시다.

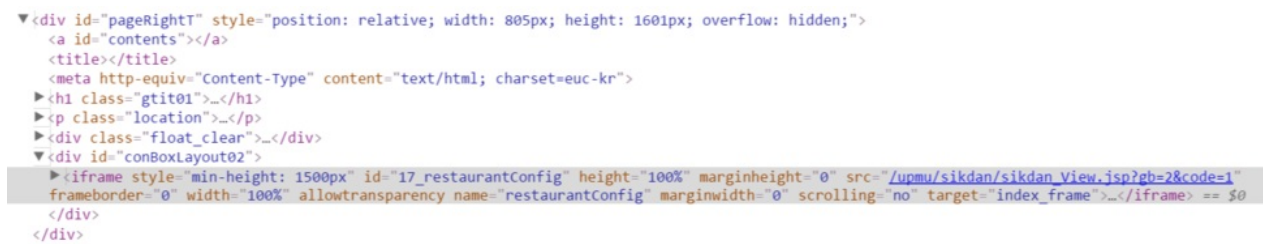
그냥 의뢰를 받고 파싱 작업을 하게되면, "아~ 그냥 소스 파일 보면, 식단 표가 있겠구나~"라고 생각할 수 있습니다. 그래서 페이지 소스를 열어보았습니다. 그러면 다음과 같은 화면을 만나게 됩니다.

[페이지 소스 첫 화면]



다음 사진을 보시면 iframe 이라는 tag가 보입니다. 이것 모르고 무작정 Nokogiri로 파싱을 했다면, 이런 데이터가 없다고 나올겁니다. iframe에 대한 지식이 없다면 당연한 결과입니다.

[iframe tag 확인]



이 iframe tag를 열어서 살펴보면, 웬??? html tag가 하나 더 있습니다. 보통 저희가 만들 때는 html은 처음과 끝 딱 두 곳에만 존재해야하는데 갑자기 html이 시작되고 있습니다.


```

<iframe style="min-height: 1500px" id="17_restaurantConfig" height="100%" marginheight="0" src="/upmu/sikdan/sikdan_View.jsp?gb=2&code=1"
frameborder="0" width="100%" allowtransparency name="restaurantConfig" marginwidth="0" scrolling="no" target="index_frame"> == $0
#document
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>...</head>
<body marginwidth="0" marginheight="0">
  <div style="width:100%; class="bridgeBox">
    <!-- 식당 정보 START -->
    <div class="floatClear marT20">...</div>
    <!-- 식당 정보 END -->
    <h3 class="h3Campus03">...</h3>
    <form name="searchForm" action="./sikdan_View.jsp" method="post">...</form>
    <div id="sikdang" class="tableStyle42Div">
      <!-- 공통, 고정메뉴, 분식 START -->
      <!-- 공통, 고정메뉴, 분식 END -->
      <!-- 주식단 TYPE1 START -->
      <table style="width: 100%; margin-bottom: 10px; float: left; border-top: #700904 1px solid" summary="식단표입니다">
        <colgroup>...</colgroup>
        <tbody>

```

자세히 살펴보면 두 번째 html tag 위에는 #document라고 되어 있습니다. 즉, javascript로 다른 페이지를 이 안에 끼워 넣은 구조입니다. 때문에 실제 소스는 여기서 보이지 않고, 우리 눈에는 링크를 타고 온 결과물만 보여지는 겁니다. 이 것만 보고서는 잘 모를 수 있어서 확대해 보았습니다. 그렇다면 노코기리로 가져올 수 없는 데이터가 있다는 말입니다. 이제는 파싱을 포기해야 할까요? 아닙니다. 웹 구조상 어딘가에는 위 식단으로 이어지는 url이 존재할 거라는 믿음을 갖고 뒤지기 시작합니다.

```


src="/upmu/sikdan/sikdan_View.jsp?gb=2&code=1"
ling="no" target="index_frame"> == $0

```

iframe 안에 src라는 것이 보입니다. 즉, link가 있습니다. src="/????/????/????" 익숙한 것이 보입니다. 이것을 검사가 아닌, 소스 파일 보기에서 열어보면 하이퍼 링크가 걸려져 있어서 클릭하면 해당 페이지로 이동할 수 있습니다. 이를 기존 url에 더해줘도 되지만, RESTFUL을 모른다면 그냥 클릭하면 됩니다. 그러면 다음과 같은 링크로 이동합니다.

(view-source:https://www.hanyang.ac.kr/upmu/sikdan/sikdan_View.jsp?gb=2&code=2)

결과화면은 다음과 같습니다. 완전하지 않지만 이전 한양대 식단에서 비슷하게 작은 페이지를 본 것 같습니다.

www.hanyang.ac.kr/upmu/sikdan/sikdan_View.jsp?gb=2&code=1																																							
		<table border="1"> <tr> <td>식당명</td> <td colspan="9">교직원식당</td> </tr> <tr> <td>위치</td> <td colspan="9">복지관 3층</td> </tr> <tr> <td>운영시간</td> <td colspan="9"> 중식 11:30 ~ 14:00 석식 17:30 ~ 18:50 </td> </tr> </table>								식당명	교직원식당									위치	복지관 3층									운영시간	중식 11:30 ~ 14:00 석식 17:30 ~ 18:50								
식당명	교직원식당																																						
위치	복지관 3층																																						
운영시간	중식 11:30 ~ 14:00 석식 17:30 ~ 18:50																																						
금주의 식단(2016-06-06 ~ 2016-06-10)																																							
월요일 (06/06) 현충일																																							
중식																																							
석식																																							
화요일 (06/07)																																							
중식	소고기우거지해장국/ 두부조림/ 김구이&양념장/ 오이양파무침/ 깍두기	4,000	수제돈까스&오므라이스/ 크림스프/ 샐러드&말기드레싱/ 단무지/ 배추김치	4,000																																			
석식	여육국/ 혼육김치찌개/ 새싹연두부드레싱/ 야채겉절이/ 깍두기	4,000																																					
수요일 (06/08)																																							
중식	얼큰부대찌개/ 메밀면아채무침/ 연두부계란찜/ 숙주들깨무침/ 깍두기&배추김치	4,000	치즈닭갈비볶음밥/ 콩나물국/ 순대두부강정/ 샐러드&말기드레싱/ 단무지/ 배추김치	4,000																																			
석식	순두부국/ 미트볼어묵피망조림/ 김가루죽음/ 미나리무생채/ 배추김치	4,000																																					
목요일 (06/09)																																							
중식	혼육김치찌개/ 브로콜리어묵죽음/ 멸치알콩조림/ 부추생채/ 깍두기	4,000	하이라이스달걀/ 미역국/ 달콤약육이/ 샐러드&망고드레싱/ 단무지무침/ 배추김치	4,000																																			
석식	돈육백김치탕/ 스크램블에그&케첩/ 콩나물무침/ 고추파클지/ 배추김치	4,000																																					

즉, 이 페이지를 기존의 한양대 식단 페이지에 끼워 넣은 겁니다. 여러 식당이 있고 그 식당마다 페이지가 있어서 하나로 뭉치기 위해서 이렇게 구조를 만들었다면 이해할 수 있는 내용입니다. 이처럼 iframe 내의 `#document & inner_html` 구조는 페이지 소스파일에서 `src`를 찾아서 해결하면 됩니다. 하지만 이렇게 복잡하게 하지 않아도 크롬 브라우저에서 "프레임 소스 보기" 클릭 한 번으로 해당 페이지를 얻는 방법도 있습니다. 아래 링크는 공부할 때 본 Stackoverflow 자료입니다.

(<http://stackoverflow.com/questions/28045121/scraping-iframe-data-using-nokogiri-and-ruby>)

Open-uri 인코딩 문제 처리

가끔 예제대로만 크롤링 작업을 하다보면 인코딩 문제가 발생할 수 있습니다.

예를 들면, Ruby의 Open-uri의 경우 Default 설정이 UTF-8 방식이므로, 자동으로 UTF-8 방식으로 변환해서 가져옵니다.

만약에 깔끔하게 UTF-8 변환을 지원하지 않는 데이터를 Open-uri로 열 경우, Nokogiri로 파싱작업을 끝낸 놈에서 "뽕뽕찌!\$#%#\$" 이런 식으로 표시가 되서 DB에 들어가 있는 결과를 만나보게 되고, 아무리 디버깅하면서 irb에서 삽질을 한다고 해도, 본인은 알고리즘 상에서 잘못된게 없다는 결론에 도달하게 됩니다. (물론, 보통은 Open-uri를 지원합니다.)

"보통 이런 형태의 문제가 있는 데이터는 일부는 잘 나오고, 일부는 깨져서 개발자를 당황시킵니다." (예를 들면, 블락 비 - BIC;\$%, BIC;\$% 부분이 Block B입니다.)

여러 삽질을 하다가 Open-uri 문제라는 걸 깨닫고, Open-uri를 대체할 놈을 찾기 시작했습니다. Open-uri 를 대체할 놈으로 Mechanize Gem 을 찾았습니다. Mechanize gem 을 설치하거나 Rails 에서 Gemfile 에 넣는 방법은 알고 있다고 가정하고 글을 씁니다.

해결 방법

```
data_url = "URL 내용을 넣어줍니다."

@agent= Mechanize.new      #Mechanize 객체 하나를 생성
@page = @agent.get data_url  #get 메소드를 사용하여 get 요청을 보낸 값을 가져옵니다.
@data_by_nokogiri = Nokogiri::HTML(@page.content)  #가져온 데이터를 다시한번 Nokogiri 로
변환합니다.
```

위처럼 open-uri를 대체할 것을 찾아서 get 요청 후 Nokogiri로 다시 파싱해서 사용하시면 됩니다.

그런데 위처럼 해도 해결이 안되는 경우가 있습니다. 아래 에러 메시지를 한 번 살펴보죠.

"nokogiri::xml::syntaxerror: input is not proper utf-8, indicate encoding"

Nokogiri 나 Mechanize를 통해서 웹 페이지의 데이터를 파싱하려고 할 때, 글자가 깨져서 나오는 경우가 있습니다. 인코딩 에러 또는 깨짐 현상이라고 말하죠.

특히 한글이 말이죠. 흔히 "뽕뽕뽕뽕뽕뽕" 이런걸 생각하실 수도 있는데 "ʼβ°j½¿»iÀ´,±âÁ±½Ä" 이렇게 나올 때도 있습니다. 아주 멘붕이 오는거죠. 에러 메시지는 "0xED 0x6E 0x2C 0x20" 이런 걸 만나보실 수도 있고 다른 걸 수도 있습니다. 글씨에 따라 다른 에러 메시지가 나오니까요. 요즘

페이지들은 대부분 알아서 UTF-8로 인코딩해서 웹 페이지를 구현합니다. 하지만 일부 짜증나는 것들이 대충 만들어서 크롤링하기 귀찮게 만들어 놓았더라구요. "페이지 자체를 인코딩해주세요." 라고 부탁하는 방법도 있지만 페이지 자체를 저희가 인코딩해서 수정할 수 없다면, 받아올 때 인코딩해서 잠시 소스를 가져오는 방법은 있습니다.

이 현상은 부분적으로 일어날 수도 있고 특정 언어 전반에 걸쳐서 일어날 수도 있습니다. 페이지가 어떻게 구현 되었는지에 따라 다릅니다. 여기서는 Nokogiri와 Open-uri 잼을 이용하여 페이지를 열어서 가져올 때 일어난 것에 대해서 살펴보겠습니다. 예시로 사용할 페이지는 '광운대학교 푸드 코트 메뉴 페이지' 입니다.

```
$irb
> require 'nokogiri'
> require 'open-uri'
> url = "http://foodcourt.kw.ac.kr/menu/menu_chinesefood.html"
> data = Nokogiri::HTML(open(url))

#보통 이렇게 하면, 정상적으로 데이터를 받아서 HTML 구조대로 Nokogiri가 나누어줍니다. 하지만 여기서는 문제가 발생합니다.
```

```
> data = Nokogiri::HTML(open(url)).errors
#이렇게 에러를 살펴보면 다음과 같은 에러 메시지가 나타납니다.
```

```
=> [#<Nokogiri::XML::SyntaxError: Input is not proper UTF-8, indicate encoding !>,
#<Nokogiri::XML::SyntaxError: Input is not proper UTF-8, indicate encoding !
Bytes: 0xEA 0xB8 0xDE 0xB4>, #<Nokogiri::XML::SyntaxError: Input is not proper UTF-8,
indicate encoding
```

#즉 페이지 자체에 문제가 있다는 의미죠. 여러 방법이 있을 수 있지만, Nokogiri 자체에서 인코딩을 처리하는 방법이 있습니다.

해결 방법

```
> Nokogiri::HTML(open(url), nil, 'euc-kr')
#이게 그 방법인데요, 왜 'euc-kr'을 사용했는지 궁금할겁니다.
```

사실 현재 이 상황에서는 노코기리 기본 인코딩인 UTF-8을 사용하는데, EUC-KR이 필요한 상황이기 때문에 발생했습니다. 뭐가 문제인지 알려주는 사이트가 있어서 공유합니다. 아래 사이트에 URL을 입력하고 옵션을 조정하면 여기서 문제가 발생했을 때 왜 인지를 알려줍니다. 위와 같은 상황에서는 다음과 같은 메시지를 남깁니다. (<http://validator.w3.org>)

Character Encoding Override in effect!

The detected character encoding "euc-kr" has been suppressed and "utf-8" used instead.

즉, EUC-KR이 필요한데, UTF-8로 역지로 인코딩 중이었다는 이야기입니다. 이제 EUC-KR로 바꿔주면 문제가 해결될 것 같은 기분이 들죠? 그래서 위처럼 Nokogiri 안에 인코딩을 'euc-kr'로 바꾸어 준 것 입니다. 일본에서도 비슷한 문제가 있어서 포스팅한 내용이 있는데 거기는 'EUC-JP'로 바꿔주었습니다. 어쨌든 이런 식으로 해결해주면 됩니다.

여기서 하나 의문을 가질 수 있습니다. "아 그래? 그럼 한글 사이트는 전부 'euc-kr' 로 옵션을 주어서 넣어주면 되겠구나~" 생각하시는 분도 있을 수 있습니다. 우선 정답은 "아닙니다." 왜냐하면, 한국에서 만들어 졌다고 모두 'EUC-KR'로 인코딩되어야 하는건 아닙니다. 구현 방식과 보고 있는 웹 브라우저에 따라서도 달라지는 것이 인코딩이므로 그 때 그 때 나오는 에러 메시지를 보고 판단하시는 것이 맞습니다. 그러므로 가장 많이 사용되는 UTF-8을 기본으로 두고 접근하시기 바랍니다. 아래는 제가 참고한 페이지들 입니다.

(https://www.youtube.com/watch?v=Pej6T_vYCml)

(<http://insideflag.blogspot.com/2010/06/rubymechanize.html>)

대학교 식단 크롤링 예제

지금부터 간단히 대학교 식단을 크롤링하는 작업에 대해서 살펴보겠습니다. 이번 예제에서 사용할 대학교는 -한양대학교 에리카 캠퍼스_입니다. 딱히 이 예제를 사용해야만 하는 이유는 없지만 이전에 `iframe` 데이터 파싱관련 글을 작성할 때 이와 관련된 내용을 다룬 김에 예제로 사용하기로 마음 먹었습니다. 그럼 시작해보겠습니다. 총 두 단계로 나뉩니다.

1. 준비단계
2. 실제 구현

위 모든 내용은 제 튜토리얼을 모두 읽은 분들이라고 가정하고 진행되므로 자세한 내용은 다루지 않고 큰 솔루션만 제공합니다. 여기에서도 다시 한번 언급하지만 제 솔루션은 **정답이 아니라 방법 중 하나일 뿐**이니 스스로 응용하여 활용하시기 바랍니다.

준비하기

ERICA캠퍼스

Home > 대학생활 > ERICA캠퍼스 > **금주의 메뉴**

I 금주의 메뉴

교직원식당	학생식당	창의인재원식당	푸드코트	창업보육센터
--------------	------	---------	------	--------

에리카 캠퍼스에는 홈페이지에서 제공하고 있는 식당이 총 5곳입니다.

1. 교직원식당
2. 학생식당
3. 창의인재원식당
4. 푸드코트
5. 창업보육센터

크롤링이라는 작업을 많이 해보신 분이라면 알겠지만, 같은 학교라도 구조가 다르게 만들어 놓은 곳이 많습니다. 그래서 결국은 공통적으로 적용해서 깔끔하게 코드가 안나오는 경우가 있죠. 개발자를 탓해야할지... 식당 운영자를 탓해야할지... 하하하...

다음과 같은 구조로 설계하려고 합니다. 예를 들면, 값 하나가 다음과 같이 나옵니다.

[데이터베이스 스키마]

- 대학 코드 : Integer
- 식당 이름 : String
- 식당 위치 : String
- 날짜 : Date
- 조식/중식/석식 : String
- 식단내용 : Text(JSON)
- 그 이외 내용 : Text

[예상 개별 튜플 결과값]

- 대학 코드 : 01
- 식당 이름 : "학생식당"
- 식당 위치 : "복지관 2층"
- 날짜 : "2016-06-06"
- 조식/중식/석식 : "lunch"
- 식단내용 : "순대감자탕, 숙주맛살볶음, 연두부&오리엔탈드레싱, 배추김치", "3,000"
- 그 이외의 내용 : ""

위 내용을 모두 홈페이지에서 제공해줄까요? 그렇다면, 정말 좋겠죠. 하지만 **원하는 내용을 전부 제공하지 않을 경우가 있습니다.** 그럴 때는 뭐... 어쩔 수 없이 수기로 작성해야죠.

그럼 시작해볼까요?

설계 및 구현

개발 환경 구축하기

```
$ rails new HanyangMenu -d mysql
#MySQL로 프로젝트를 생성합니다.

$ rails generate scaffold diet univ_id:integer name:string location:string date:time:string diet:text extra:text
#전 포스팅에서 작성한 내용대로 테이블 구조를 생성합니다.

$ rake db:create
#database.yml에 설정한대로 데이터베이스를 생성합니다.

$ rake db:migrate
#Migration 파일과 맞는 형태로 스키마 작성 및 테이블 실제 생성합니다.
```

Controller, Model Unique key, Mysql password

```
#diets_controller.rb
hanyang = HANYANG.new
hanyang.scarpe
```

```
#diet.rb
class Diet < ActiveRecord::Base
  validates :univ_id, :uniqueness => { :scope => [:name, :date, :time, :diet] }
end
```

```
#database.yml
default: &default
adapter: mysql2
encoding: utf8
pool: 5
username: root
password: ***** //알아서 은닉화 시켜서 처리하시기 바랍니다.
socket: /tmp/mysql.sock
```

Hanyang.rb

이 부분은 직접 해당 페이지의 소스를 켜서 분석하면서 보시기 바랍니다. 지금까지 튜토리얼을 모두 보셨다면 하나하나 적을 필요는 없다고 판단합니다. 그리고 이건 정답이 아니고, 이런 식으로 하는 방법이 있다는 것을 알려드릴 뿐입니다.

```
#app/controllers/HANYANG.rb
class HANYANG
  def initialize
    @default_dates = Array.new #날짜에 대한 내용을 저장할 배열.
  end
  def scrape
    #이 부분에 Web Scraping 내용을 넣을 예정입니다.
  end
end
```

```
#scrape method part#

#한양대학교 에리카캠퍼스 식단 URL
hanyang_erica_url = "https://www.hanyang.ac.kr/upmu/sikdan/sikdan_View.jsp?gb=2&code=2"

#Nokogiri로 <html> 기준으로 파싱
hanyang_erica_data = Nokogiri::HTML(open(hanyang_erica_url))

#Mon to Sun
(0..6).each do |i|
  #대부분은 월~금까지 제공하지만, 한양대학교는 일요일까지 식단을 제공하는 경우도 있어서 0~6로 처리하였습니다.
  @default_dates << ((Date.parse hanyang_erica_data.css('h3.h3Campus03').text.strip.split('(')[1].split(' ')[0]) + i).to_s
  #2016-06-01 형태로 제공하므로 이걸 Date 형태로 바꾸어 +1씩 하면서, 일요일까지 처리하고 다시 String으로 바꿉니다.
end

#1차 분류(식단을 관리하는 부분으로 갑니다.)
target = hanyang_erica_data.css('div#sikdang table')

content = ""
i = 0 #날짜별 즉, 테이블
target.each do |t|
  p = 0 #각 메뉴별 즉, <td>
  t.css('td').each do |part|
    if (part.nil? || part.text == " ")
      puts "nothing"
    elsif (part.text == "중식" || part.text == "석식")
      puts "nothing"
    end
  end

  #각 식단 콘텐츠를 살펴보면, 2개만 있을 때도 있고, 세 가지 이상이 있을 경우가 있습니다. 처음인 요일(날짜)를 제외하고, 그리고 그 다음 조식/중식/석식을 제외하고는 내용 -> 가격 -> 내용 -> 가격 -> 내용 -> 가격 순으로 표가 구성되어 있습니다. 그러므로 홀수는 content, 짝수는 price에 넣습니다. 그리고 price가 채워지는 순간, 하나의 콘텐츠가 완성되므로 바로 튜플 하나를 생성합니다.
  elsif (p % 2 == 1) #홀수
```

```

content = part.text
elsif (p % 2 == 0) #짝수
  price = part.text.scan(/\d/).join('') #price가 완료되면, 객체 생성
  Diet.create(
    :univ_id => 01,
    :name => "학생식당",
    :location => "복지관 2층",
    :date => @default_dates[i],
    :time => 'lunch',
    #diet는 JSON 형태로 제공하므로, Ruby에서 제공하는 json gem을 이용해서 json처리를 다음과 같
    이 합니다. 사용하는 gem의 버전과 rails 버전에 따라 방식이 달라지므로, 사용시 주의하시기 바랍니다.
    :diet => JSON.generate({:name => content, :price => price}),
    :extra => ''
  )
else
  puts "nothing"
  #이 부분은 예외의 상황이나 nil 또는 빈칸으로 컨텐츠가 없을 경우 command 라인에 표시됩니다.
end
p += 1
end
i += 1
#금요일까지 했으면 끝, 보통 컨텐츠가 들어있는 부분 이외에도 테이블을 생성해서 유지하고 있는 깔끔하지
않은 페이지들이 있습니다. 그래서 이렇게 하나하나 처리를 해줘야 하는 불편함이 있죠... 생각이 있는 개발자라
면, 더 깔끔하게 처리 했을 텐데 아쉽습니다. 다른 학교는 더 가관인 경우가 있습니다. 그래도 한양대학교가 깔
끔하게 만든 편은 아니죠... 동덕여대나 덕성여대가 깔끔한 것 같습니다.
if (i == 5)
  break
else
  end
end
end

```

이해가 안되거나 이상한 점 혹은 질문이 있다면 댓글로 남겨주시면됩니다. 그럼 도움이 되셨기를...
 다른 학교 식단을 크롤링하고 싶은데, 궁금한 점이나 어려운 점이 있다면 연락 주시면 아는데까지
 설명해 드리겠습니다.

추가로 제가 크롤링 한 식단 중 보고 싶은 거기 있다면, 알려주시면 열어드리겠습니다.

(동덕여대, 덕성여대, 한성대, 한양대, 동아대, 삼육대, 광운대, 인하대, 명지대)까지 구현 완료했습니다.