데이터 수집

정적 스크래핑(크롤링)

x-path, dom 객체 등에 대한 정보도 알아야 한다. 웹클라이언트에 대한 어느정도 지식이 있어야 한다.

스크래핑-> 원하는 부분에 위치한 정보를 컨텐츠로 추출하는것
웹크롤링->서버에 요청해서 가져오는 것
가져오는것이 크롤링 가져온것을 가지고 컨텐츠로 추출하는 것이 크롤링

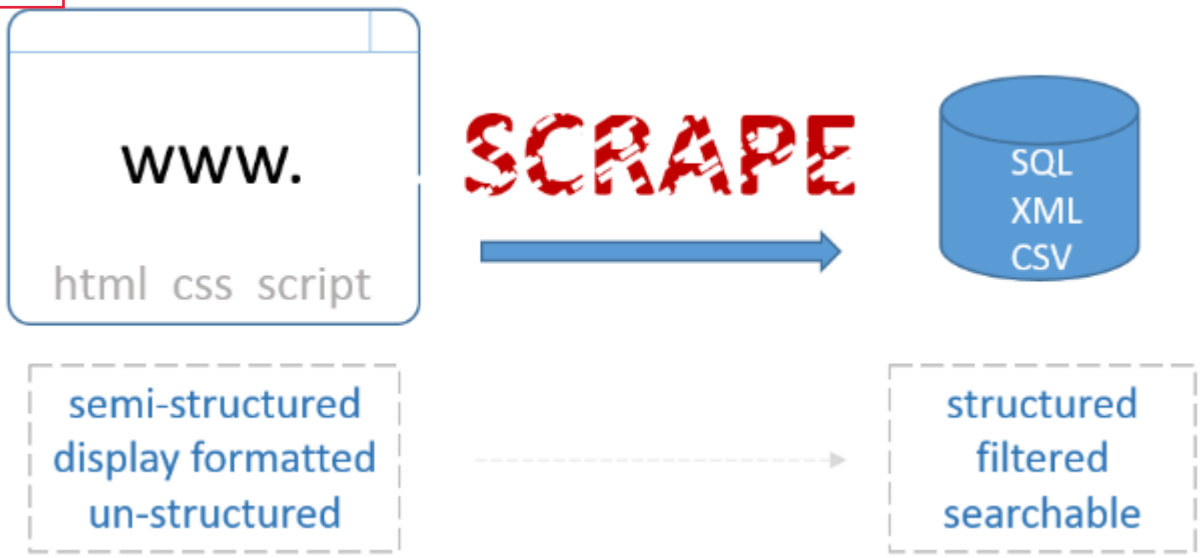
[웹 스크래핑(web scraping)]

웹 사이트 상에서 원하는 부분에 위치한 정보를 컴퓨터로 하여금 자동으로 추출하여 수집하는 기술

[웹 크롤링(web crawling)]

자동화 봇(bot)인 웹 크롤러가 정해진 규칙에 따라 복수 개의 웹 페이지를 브라우징 하는 행위

HTTP 프로토콜에 대해 알면된다.



데이터 수집

정적 스크래핑(크롤링)

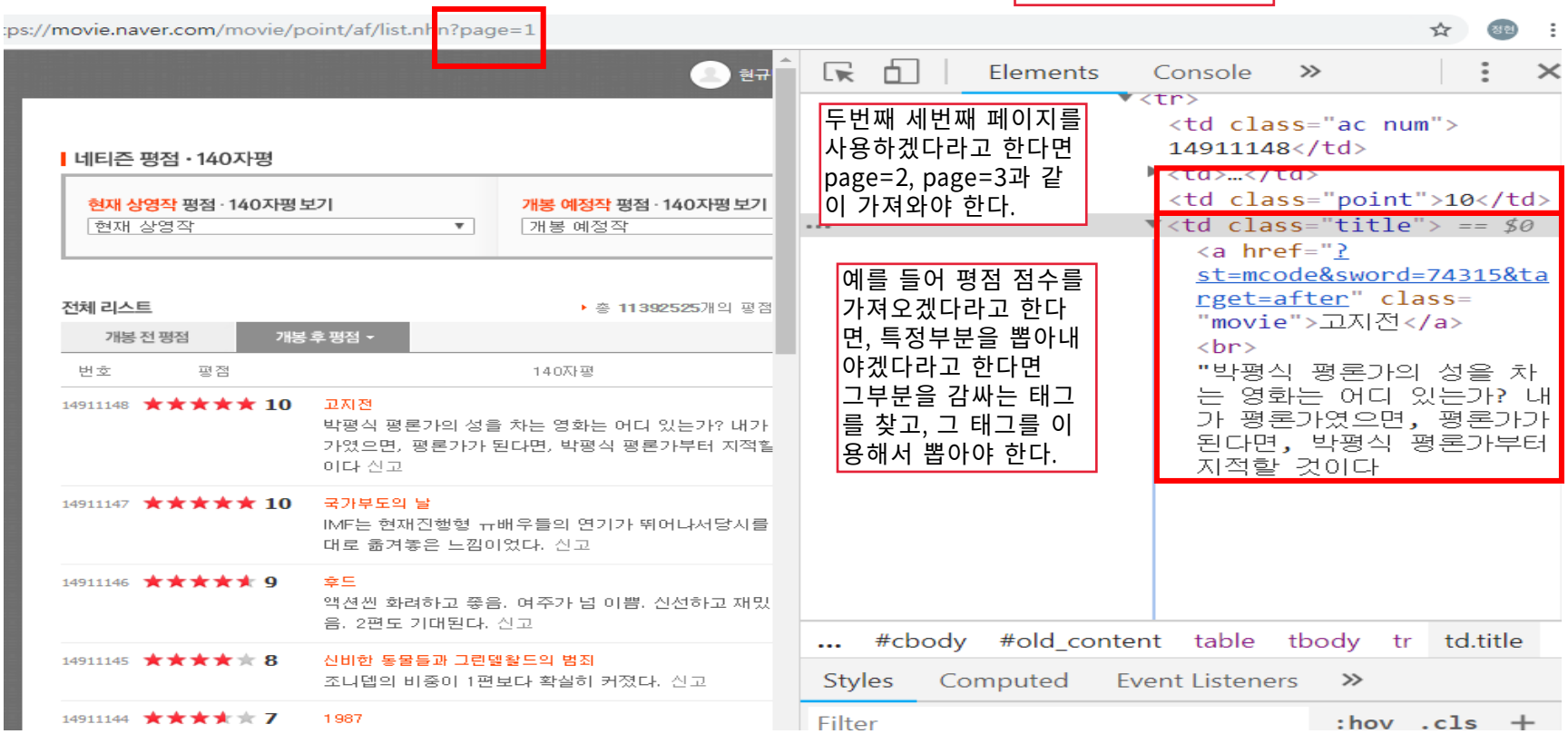
네이버 영화 사이트 댓글정보 스크래핑

네이버 영화 사이트의 데이터 중 영화제목, 평점, 리뷰만을 추출하여 CSV 파일의 정형화된 형식으로 저장한다.

(1) 스크래핑하려는 웹페이지의 URL 구조와 문서 구조를 파악해야 한다.

- URL 구조 : http://movie.naver.com/movie/point/af/list.nhn?page=1

쿼리문자열을 같이 사용해야 한다.



데이터 수집

서버에서 웹페이지를 끌어온다.
x-path/ css선택자를 가지고 원하는 변수를 가지고 온다.

정적 스크래핑(크롤링)

- 문서 구조

  영화 제목 class="movie"

  영화 평점 class="point"

  영화 리뷰 class="title"

nodes 여러개 node 한 개

node에서 css는 바로 써도 되지만 xpath같은 경우에는 xpath= 이라고 지정해야 한다.

[rvest 패키지의 주요 함수]

html_nodes(x, css, xpath), html_node(x, css, xpath)

html_text(x, trim=FALSE)

trim-> 공백 제거
trim=true 하게 되면 쓸때 없는 공백을 제거해 준다.

html_attrs(x)

html_attr(x, name, default = "")

그런속성이없다면 NA가 리턴되었는데 여기 default""값을 넣어주면 name 속성값이 없을 때 default값을 리턴해 준다.

[1페이지 스크래핑]

```
install.packages("rvest");
library(rvest)
text<- NULL
url<- "http://movie.naver.com/movie/point/af/list.nhn?page=1"
text <- read_html(url, encoding="CP949")
# 영화제목
nodes <- html_nodes(text, ".movie")
title <- html_text(nodes)
# 영화평점
nodes <- html_nodes(text, ".title em")
point <- html_text(nodes)
# 영화리뷰
nodes <- html_nodes(text,
xpath="//*[@id='old_content']/table/tbody/tr/td[2]/text()")
imsi <- html_text(nodes, trim=TRUE)
review <- imsi[nchar(imsi) > 0]
if(length(review) == 10) {
  page <- cbind(title, point)
  page <- cbind(page, review)
  write.csv(page, "movie_reviews.csv")
} else {
  cat("리뷰글이 생략된 데이터가 있네요ㅜㅜ₩n")
}
```

서버에서 끌어오는 것 euc-kr일 경우 CP949를 넣어줘야 한다.
UTF-8인경우 생략가능하나, UTF-8로 해도 된다.

스크랩핑 하는 부분

데이터 수집

정적 스크래핑(크롤링)

[여러 페이지 스크래핑]

```
# 여러 페이지
site<- "http://movie.naver.com/movie/point/af/list.nhn?page="
text <- NULL
movie.review <- NULL
for(i in 1: 100) {
  url <- paste(site, i, sep="")
  text <- read_html(url,  encoding="CP949")
  nodes <- html_nodes(text, ".movie")
  title <- html_text(nodes)
  nodes <- html_nodes(text, ".title em")
  point <- html_text(nodes)
  nodes <- html_nodes(text, xpath="//*[@id='old_content']/table/tbody/tr/td[2]/text()")
  imsi <- html_text(nodes, trim=TRUE)
  review <- imsi[nchar(imsi) > 0]
  if(length(review) == 10) {
    page <- cbind(title, point)
    page <- cbind(page, review)
    movie.review <- rbind(movie.review, page)
  } else {
    cat(paste(i," 페이지에는 리뷰글이 생략된 데이터가 있어서 수집하지 않습니다.ㅜㅜ₩n"))
  }
}
write.csv(movie.review, "movie_reviews2.csv")
```

데이터 수집

정적 스크래핑(크롤링)

한국일보 헤드라인
기사 스크래핑

정적 스크래핑(크롤링)

[XML 패키지의 주요 함수]

htmlParse (file, encoding="...")

xpathSApply(doc, path, fun)

fun : **xmlValue, xmlGetAttr, xmlAttrs**

```
library(XML)
imsi <- read_html("http://hankookilbo.com")
t <- htmlParse(imsi)
content<- xpathSApply(t,"//p[@class='title']", xmlValue);
content
content <- gsub("[[:punct:][:cntrl:]]", "", content)
content
content <- trimws(content)
content
```

정적 스크래핑(크롤링)

그 외의 웹 스크래핑시 알고 있으면 도움되는 내용들

[R에서 GET으로 사이트 내용 가져오기 : httr 패키지 사용]

```
install.packages("httr")
library(httr)
http.standard <- GET('http://www.w3.org/Protocols/rfc2616/rfc2616.html')
title2 = html_nodes(read_html(http.standard), 'div.toc h2')
title2 = html_text(title2)
Title2
```

[R에서 POST로 사이트 내용 가져오기 : httr 패키지 사용]

```
game = POST('http://www.gevolution.co.kr/score/gamescore.asp?t=3&m=0&d=week',
            encode = 'form', body=list(txtPeriodW = '2020-03-15'))
title2 = html_nodes(read_html(game), 'a.tracktitle')
title2 = html_text(title2)
title2[1:10]
```

정적 스크래핑(크롤링)

[뉴스, 게시판 등 글 목록에서 글의 URL만 뽑아내기]

```
res = GET('https://news.naver.com/main/list.nhn?mode=LSD&mid=sec&sid1=001')
htxt = read_html(res)
link = html_nodes(htxt, 'div.list_body a'); length(link)
article.href = unique(html_attr(link, 'href'))
article.href
```

[이미지, 첨부파일 다운 받기]

```
# pdf
res = GET('http://cran.r-project.org/web/packages/httr/httr.pdf')
writeBin(content(res, 'raw'), 'c:/Temp/httr.pdf')
# jpg
h = read_html('http://unico2013.dothome.co.kr/productlog.html')
imgs = html_nodes(h, 'img')
img.src = html_attr(imgs, 'src')
for(i in 1:length(img.src)){
  res = GET(paste('http://unico2013.dothome.co.kr/',img.src[i], sep=""))
  writeBin(content(res, 'raw'), paste('c:/Temp/', img.src[i], sep=""))
}
```

데이터 수집

[정규표현식 활용]

```
word <- "JAVA javascript Aa 가나다 AAaAaA123 %^&*"
gsub("A", "", word)
gsub("a", "", word)
gsub("Aa", "", word)
gsub("(Aa){2}", "", word)
gsub("[Aa]", "", word)
gsub("[가-힣]", "", word)
gsub("[^가-힣]", "", word)
gsub("[&^%*]", "", word)
gsub("[[:punct:]]", "", word)
gsub("[[:alnum:]]", "", word)
gsub("[1234567890]", "", word)
gsub("[[:digit:]]", "", word)
gsub("[^[:alnum:]]", "", word)
gsub("[[:space:]]", "", word)
```

.	Any character except \n
|	Or, e.g. (a|b)
[...]	List permitted characters, e.g. [abc]
[a-z]	Specify character ranges
[^...]	List excluded characters
(...)	Grouping, enables back referencing using \\N where N is an integer

*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{,n}	Matches at most n times
{n,m}	Matches between n and m times

[[:digit:]] or \\d	Digits; `[0-9]`
\\D	Non-digits; `[^0-9]`
[[:lower:]]	Lower-case letters; `[a-z]`
[[:upper:]]	Upper-case letters; `[A-Z]`
[[:alpha:]]	Alphabetic characters; `[A-z]`
[[:alnum:]]	Alphanumeric characters `[A-z0-9]`
\\w	Word characters; `[A-z0-9_]`
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; `[0-9A-Fa-f]`
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; `[^[:space:]]`
[[:punct:]]	Punctuation characters; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~`
[[:graph:]]	Graphical char.; `[[:alnum:][:punct:]]`
[[:print:]]	Printable characters; `[[:alnum:][:punct:]\\s]`
[[:cntrl:]] or \\c	Control characters; `\n`, `\r` etc.

데이터 수집

◈ Open API를 활용한 공공 DB 수집

- 공공 데이터 포털 사이트에서 서울시 버스 정보와 버스 위치 정보 읽어오기

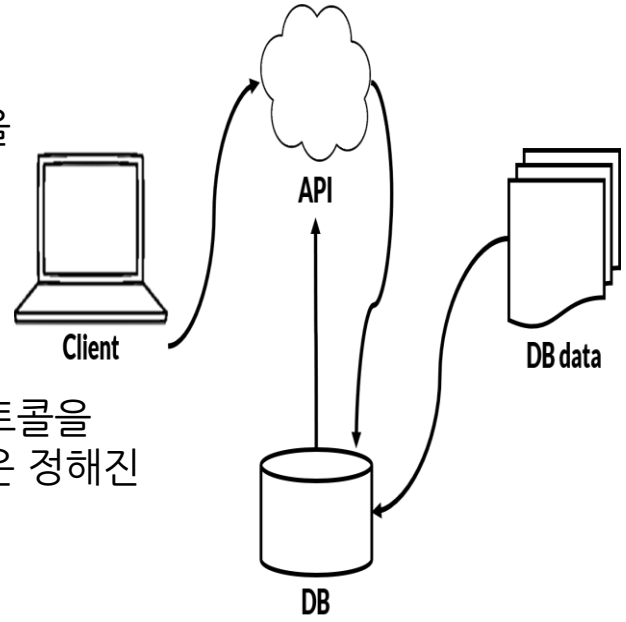
[Open API와 Rest API]

API란 Application Programming Interface의 약자로, 특정 프로그램을 만들기 위해 제공되는 모듈(함수 등)을 의미한다.

Open API : 공개 API 라고도 불리우며, 누구나 사용할 수 있도록 공개된 API이다. 주로 Rest API 기술을 많이 사용한다.

Rest API : Representational State Transfer API의 약자로, HTTP프로토콜을 통해서 정보를 제공하는 함수를 의미하며 실질적인 API 사용은 정해진 구조의 URL 문자열을 사용한다.

일반적으로 XML, JSON의 형태로 응답을 출력한다.

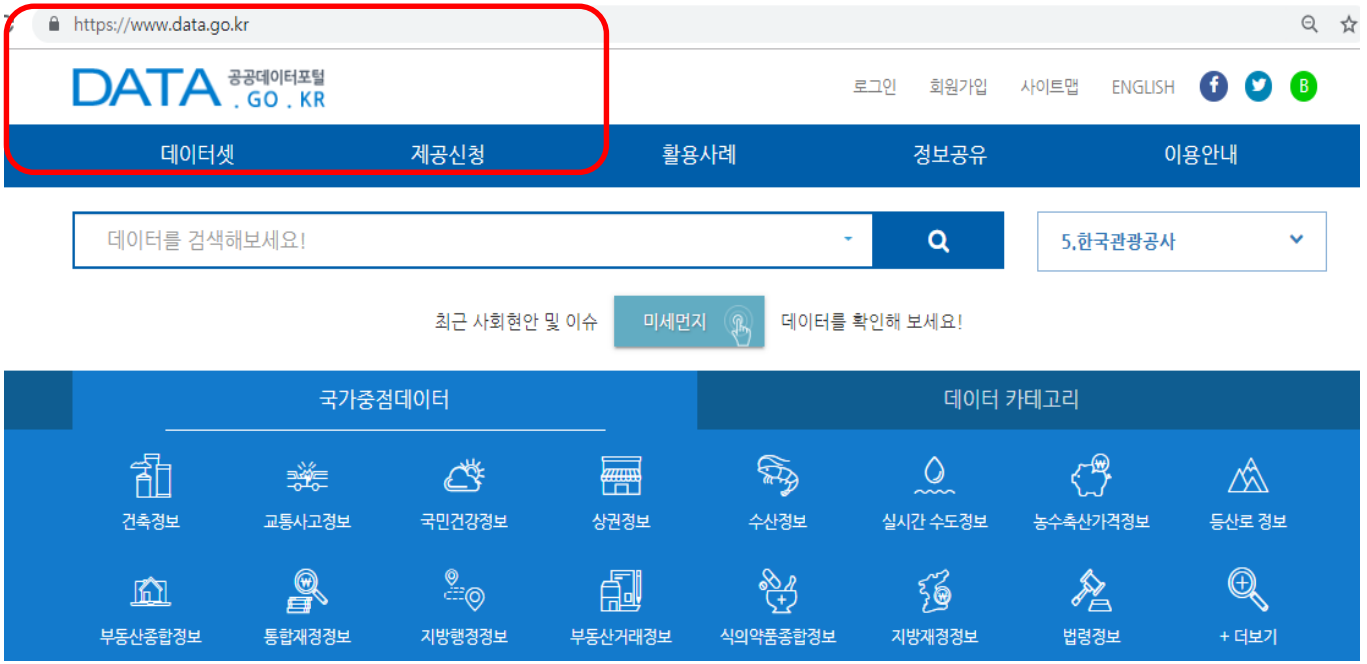


10

데이터 수집

◈ Open API를 활용한 공공 DB 수집

- 공공 데이터 포털 사이트에서 서울시 버스 정보와 버스 위치 정보 읽어오기



옆의 화면은 공공 데이터 포털 사이트의 화면이다.

다음 URL로 방문한다.

http://data.seoul.go.kr/

11

데이터 수집

◈ Open API를 활용한 공공 DB 수집

- 공공 데이터 포털 사이트에서 서울시 버스 정보와 버스 위치 정보 읽어오기

[승인] 버스위치정보조회 서비스 신청일:2019-01-09 [연장신청] 만료예정 :2021-01-09

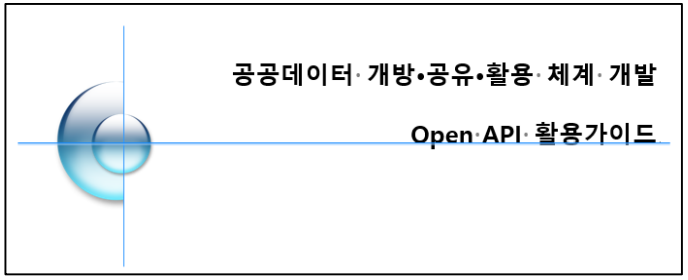
서비스유형 :REST 분류 :교통및물류 > 도로 제공기관 :서울특별시

[승인] 노선정보조회 서비스 신청일:2019-01-09 [연장신청] 만료예정 :2021-01-09

서비스유형 :REST 분류 :교통및물류 > 도로 제공기관 :서울특별시

공공 데이터 포털 사이트에 로그인 한 후에 버스위치정보 조회 서비스와 노선정보조회 서비스의 데이터 사용을 요청하고 인증키를 받습니다.

Open API 활용 가이드가 제공되므로 이 문서를 참조하여 구현한다.



기본정보

서비스명	노선정보조회 서비스 상세설명				
서비스 유형	REST	일일트래픽	#######	평균응답속도(초)	53.599777777777774
심의여부	자동승인	신청유형	개발계정 | 연장신청	처리상태	승인
활용기간	2014-11-11 ~ 2021-01-09				

서비스정보

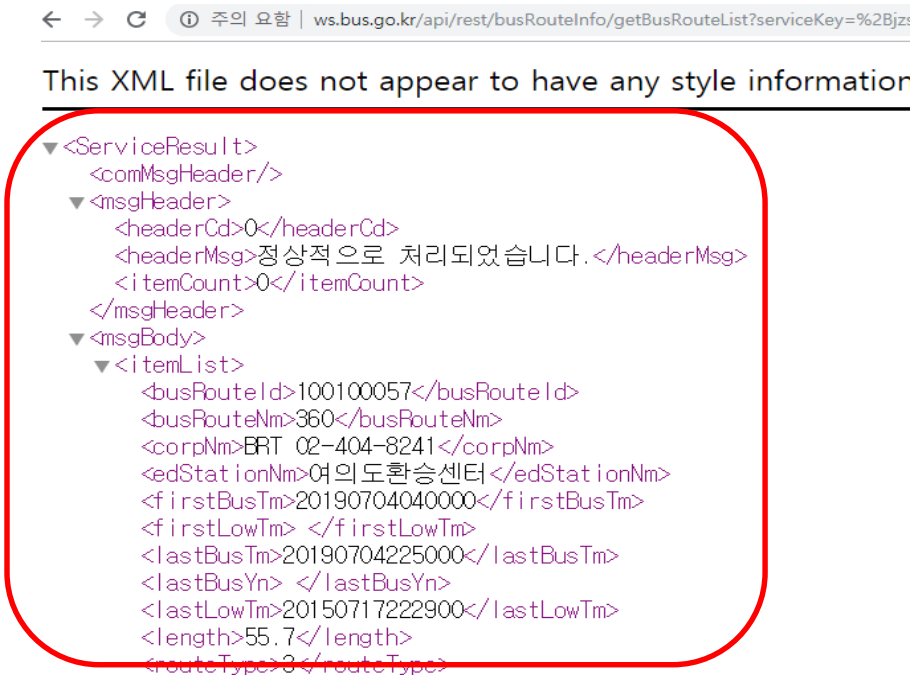
일반 인증키 (UTF-8)	복사
End Point	
데이터포맷	XML
참고문서	서울특별시_노선정보조회_서비스_활용가이드_20190110.docx

12

데이터 수집

◈ Open API를 활용한 공공 DB 수집

- 공공 데이터 포털 사이트에서 서울시 버스 정보와 버스 위치 정보 읽어오기



버스의 위치 정보를 받아오기 위해서는 버스의 노선 정보를 요청하여 버스의 라우트아이디<busRouteId>를 알아야 한다.

부여받은 인증키와 정보를 얻고자 하는 버스 번호를 입력하여 다음과 같이 URL을 구성한 다음 버스 번호에 대한 노선 정보를 요청한다.

http://ws.bus.go.kr/api/rest/busRouteInfo/getBusRouteList?serviceKey=인증키&strSrch=버스번호

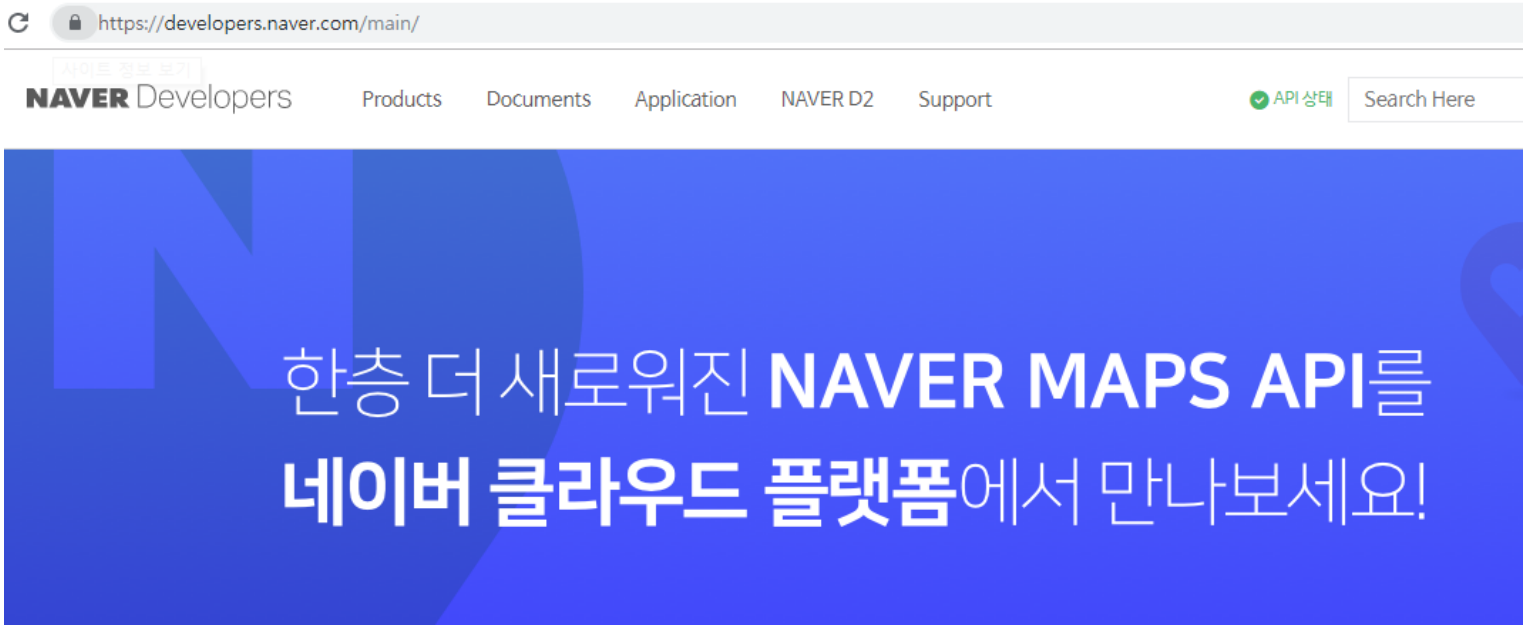
13

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 블로그에 올려진 글 검색하여 읽어오기

다음과 같이 네이버 개발자 메인 웹 페이지에 방문하고 로그인한다. : https://developers.naver.com/main/



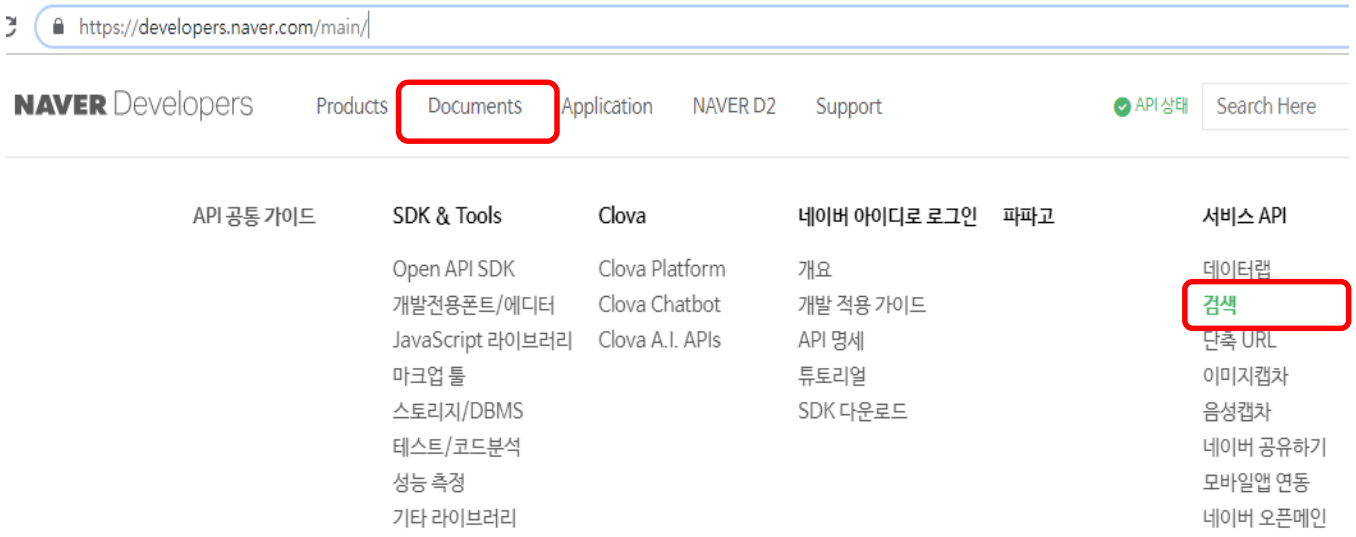
14

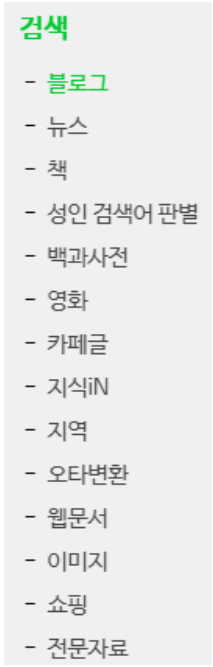
데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 블로그에 올려진 글 검색하여 읽어오기

Documents 메뉴를 선택한 다음 검색 메뉴를 선택한다. 네이버의 검색 Open API 서비스는 블로그 뉴스, 책, 백과사전 등 다양한 분야의 내용을 검색할 수 있게 지원한다.





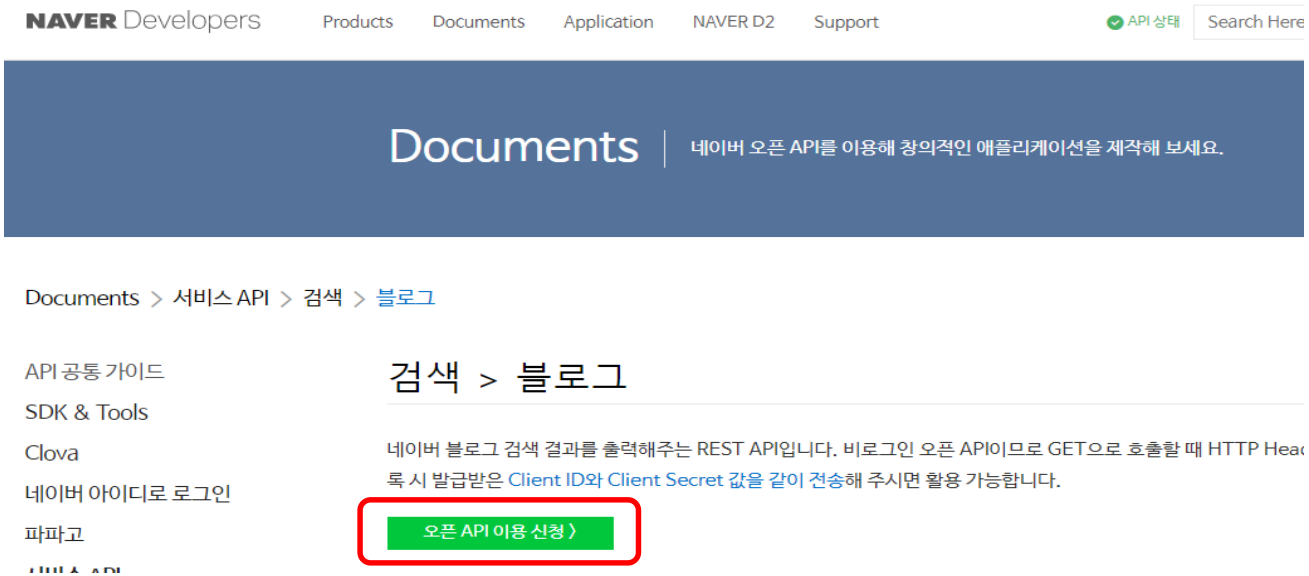
15

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 블로그에 올려진 글 검색하여 읽어오기

다음은 네이버 블로그 검색 결과를 출력해 주는 REST API의 웹 화면이다. 연두색의 오픈 API 이용 신청 링크를 클릭한다.
(이 페이지에 직접 방문할 수 있는 URL은 https://developers.naver.com/docs/search/blog/ 이다.)



16

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 블로그에 올려진 글 검색하여 읽어오기

오픈 API 이용 신청 링크를 클릭하면 다음 화면으로 이어진다. **애플리케이션 이름**, 사용 API 그리고 **비 로그인 오픈 API 신청**에 대한 정보를 입력한 후에 Client ID와 Client Secret를 부여받습니다.

17

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 블로그에 올려진 글 검색하여 읽어오기

이 과정을 위해서 내용 집필자가 오픈API테스트라는 이름으로 애플리케이션 이름을 등록하면서 Client ID와 Client Secret를 부여받은 화면이다.

18

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 블로그에 올려진 글 검색하여 읽어오기

네이버 블로그 검색 Rest API는 다음과 같은 URL 문자열과 Query 문자열 구성 정보를 사용한다.

메서드	인증	요청 URL	출력 포맷
GET	-	https://openapi.naver.com/v1/search/blog.xml	XML
GET	-	https://openapi.naver.com/v1/search/blog.json	JSON

요청 변수	타입	필수 여부	기본값	설명
query	string	Y	-	검색을 원하는 문자열로서 UTF-8로 인코딩
display	integer	N	10(기본값),100(최대)	검색 결과 출력 건수 지정

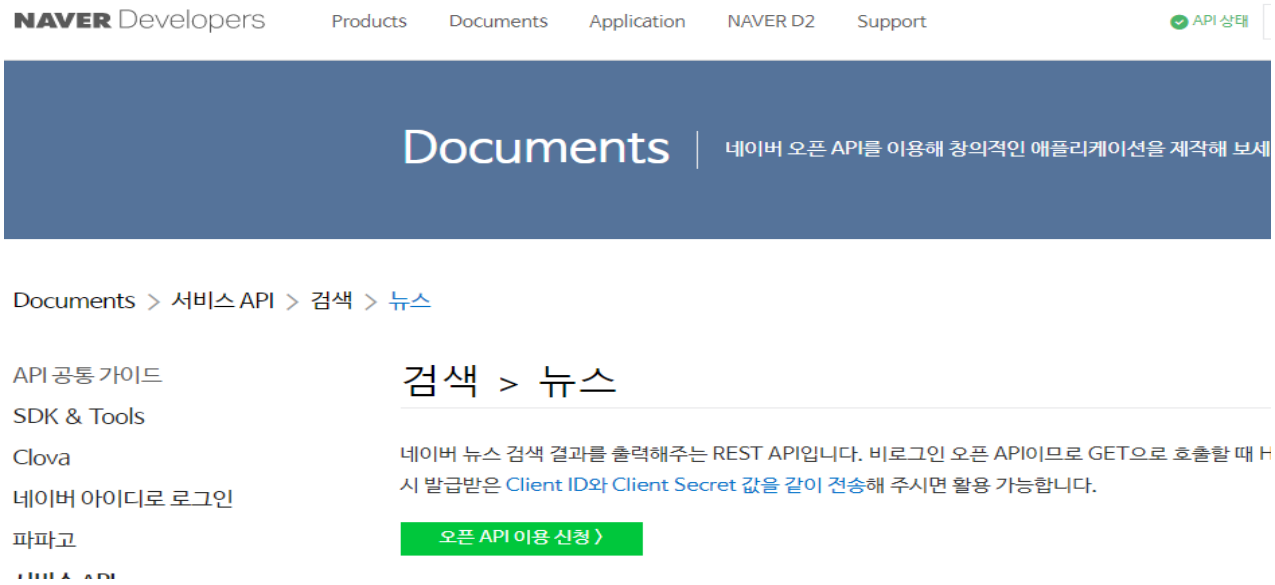
19

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 뉴스에 올려진 글 읽어오기

다음은 네이버 뉴스 검색 결과를 출력해주는 REST API의 웹 화면이다.
블로그 검색에서 부여받은 Client ID와 Client Secret를 공통으로 사용 가능한다.



20

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 네이버 뉴스에 올려진 글 검색하여 읽어오기

네이버 뉴스 검색 Rest API는 다음과 같은 URL 문자열과 Query 문자열 구성 정보를 사용한다.

메서드	인증	요청 URL	출력 포맷
GET	-	https://openapi.naver.com/v1/search/news.xml	XML
GET	-	https://openapi.naver.com/v1/search/news.json	JSON

요청 변수	타입	필수 여부	기본값	설명
query	string	Y	-	검색을 원하는 문자열로서 UTF-8로 인코딩
display	integer	N	10(기본값),100(최대)	검색 결과 출력 건수 지정

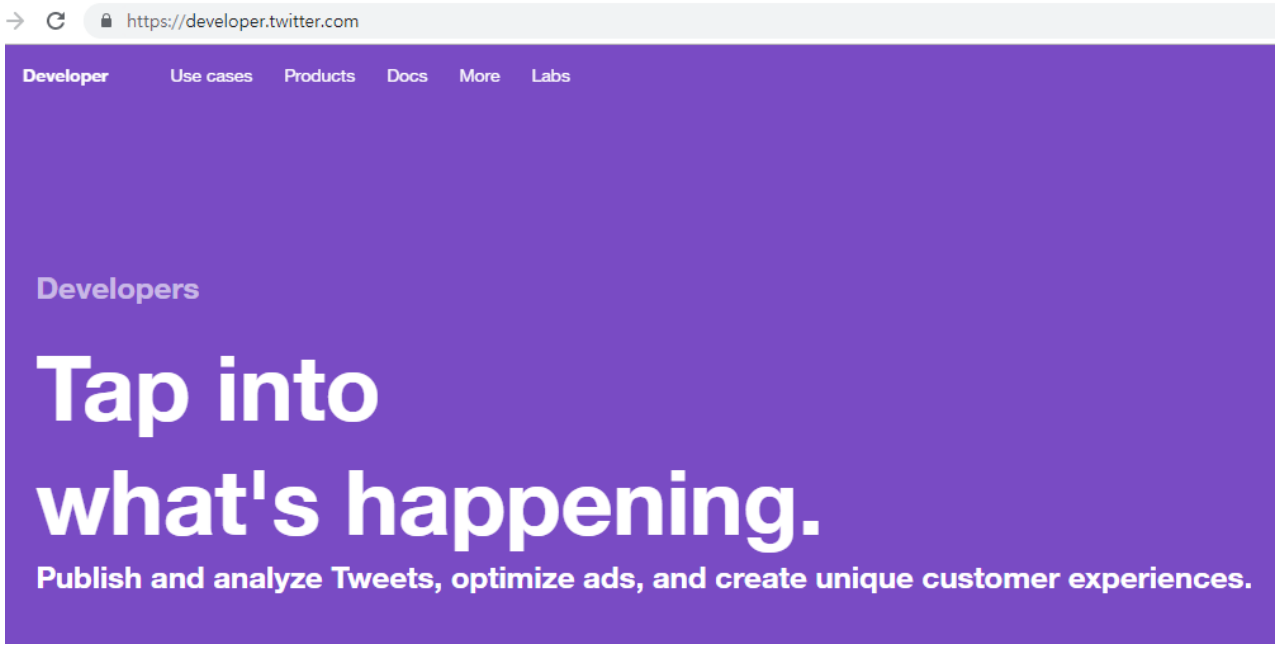
21

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 트위터에 올려진 글에서 검색하여 읽어오기

다음과 같이 트위터 개발자 메인 웹 페이지에 방문한다. : https://developer.twitter.com/



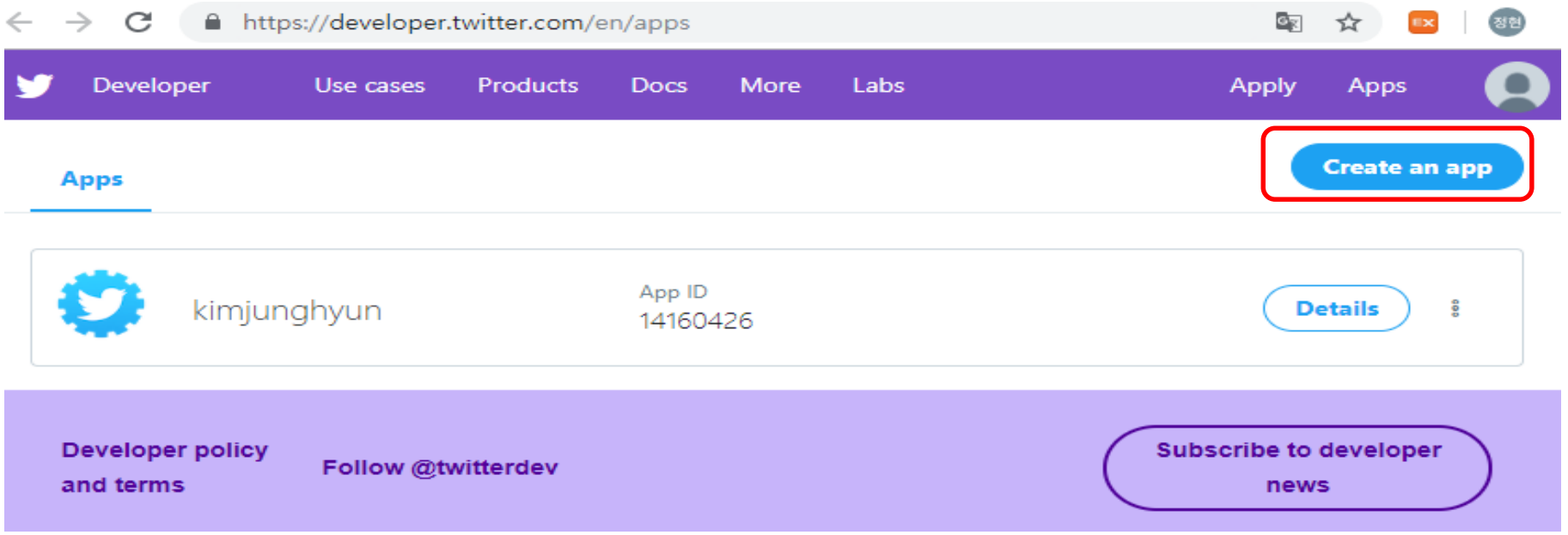
22

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 트위터에 올려진 글에서 검색하여 읽어오기

트위터에 개발자로서 회원 가입하고 로그인한 다음 Create an app 링크를 클릭하여 app(애플리케이션)을 생성한다.



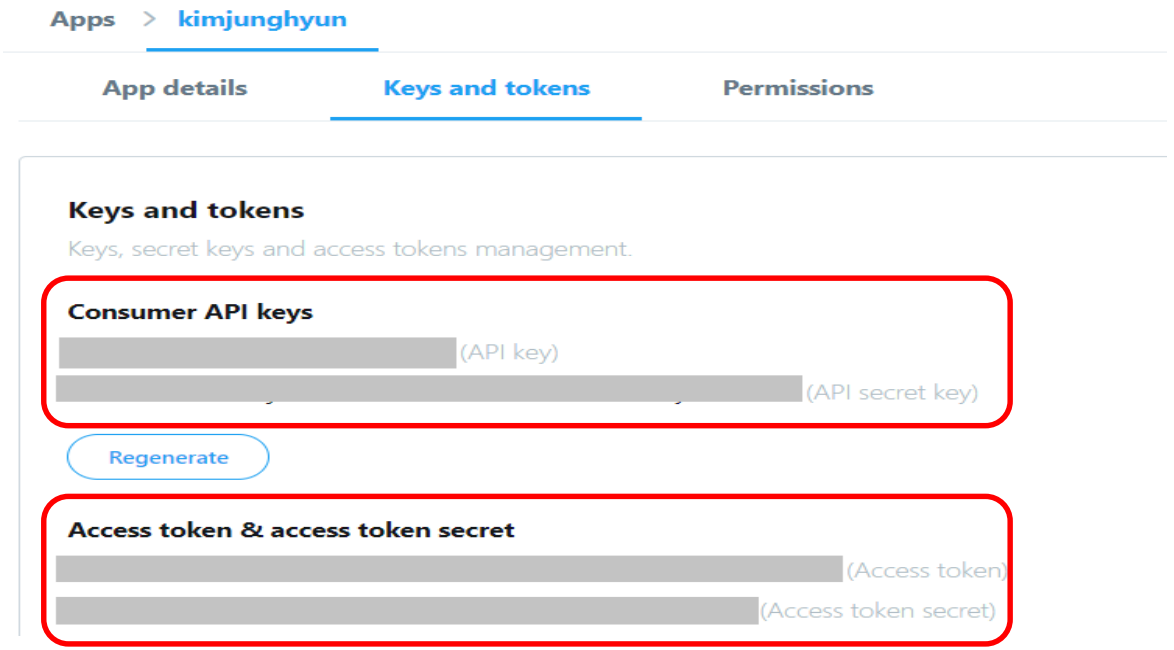
23

데이터 수집

◈ Rest API 그리고 SNS API를 활용한 SNS 글 수집

- 트위터에 올려진 글에서 검색하여 읽어오기

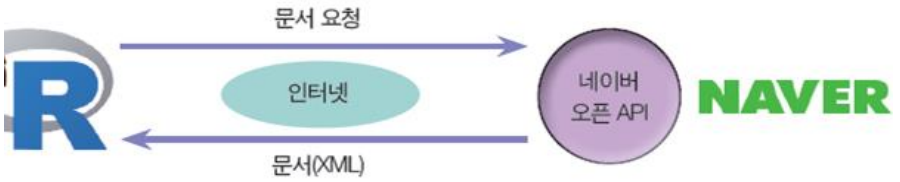
App을 생성하고 등록하면 다음과 같이 Consumer API Key와 Access token&Access token secret를 부여받게 됩니다.



24

데이터 수집

SNS 수집 : 네이버의 뉴스와 블로그 글 읽어오기



https://developers.naver.com/docs/search/blog/ 에서 내용 검토

API	요청	출력 포맷
뉴스	https://openapi.naver.com/v1/search/news.xml	XML
블로그	https://openapi.naver.com/v1/search/blog.xml	XML

요청 변수	값	설명
query	문자(필수)	검색을 원하는 질의, UTF-8 인코딩
display	정수: 기본값 10, 최대 100	검색 결과의 출력 건수(최대 100까지 가능)
start	정수: 기본값 1, 최대 1000	검색의 시작 위치(최대 1000까지 가능)
sort	문자: date(기본값), sim	정렬 옵션 • date : 날짜순(기본값) • sim : 유사도순

데이터 수집

네이버의 뉴스와 블로그 글 읽어오기

네이버 블로그에서 "여름추천요리"를 검색하여 블로그에 올려진 데이터를 수집해본다.

```
searchUrl<- "https://openapi.naver.com/v1/search/blog.xml"
Client_ID <- "....................."
Client_Secret <- "..............................."
# URLencode("여름추천요리")
query <- URLencode(iconv("여름추천요리","euc-kr","UTF-8"))
query
url<- paste(searchUrl, "?query=", query, "&display=20", sep="")
doc<-GET(url, add_headers('Content-Type' = "application/xml",
                          'X-Naver-Client-Id' = Client_ID,'X-Naver-Client-Secret' = Client_Secret))
doc1 <- htmlParse(doc, encoding="UTF-8")
text1<- xpathSApply(doc1, "//item/description", xmlValue)
text1
doc2 <- html_nodes(read_html(doc, encoding="UTF-8"), xpath='//item/description')
text2 <- html_text(doc2)
text2
doc3 <- html_nodes(content(doc, encoding="UTF-8"), xpath='//item/description')
text3 <- html_text(doc3)
text3
```

데이터 수집

네이버의 뉴스와 블로그 글 읽어오기

네이버 뉴스에서 "검찰"로 검색하여 뉴스에 올려진 데이터를 수집해본다.

```
searchUrl<- "https://openapi.naver.com/v1/search/news.xml"
Client_ID <- "izGsqP2exeThwwEUVU3x"
Client_Secret <- "WrwbQ1l6ZI"
query <- URLencode(iconv("검찰","euc-kr","UTF-8"))
url<- paste(searchUrl, "?query=", query, "&display=20", sep="")
doc<-GET(url, add_headers('Content-Type' = "application/xml",
                          'X-Naver-Client-Id' = Client_ID,
                          'X-Naver-Client-Secret' = Client_Secret))

paringData <- htmlParse(doc, encoding="UTF-8")
text<- xpathSApply(paringData, "//item/description", xmlValue);
text
xpathSApply(paringData, "//item/title", xmlValue)
```

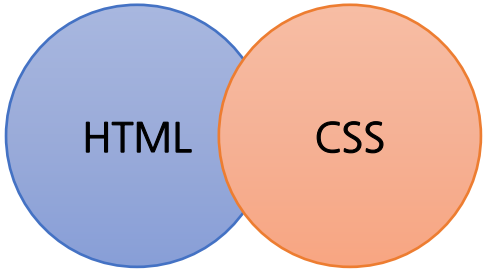
데이터 수집

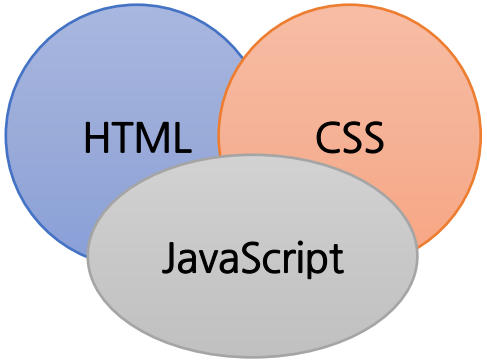
◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지

웹 서버에서 전송된 웹 페이지의 소스에서 화면에 랜더링된 내용을 모두 찾을 수 있는 경우 정적 웹 페이지라 하며 HTML만으로 작성되거나 HTML과 CSS 기술 등으로 구현된 경우가 많습니다.

웹 서버에서 전송된 웹 페이지의 소스에서 화면에 랜더링된 내용을 일부 찾을 수 없는 경우 동적 웹 페이지라 하며 HTML과 CSS 기술 외에 JavaScript 프로그래밍 언어로 브라우저에서 실행시킨 코드에 의해 웹 페이지의 내용이 랜더링 시 자동으로 생성되는 페이지이다.



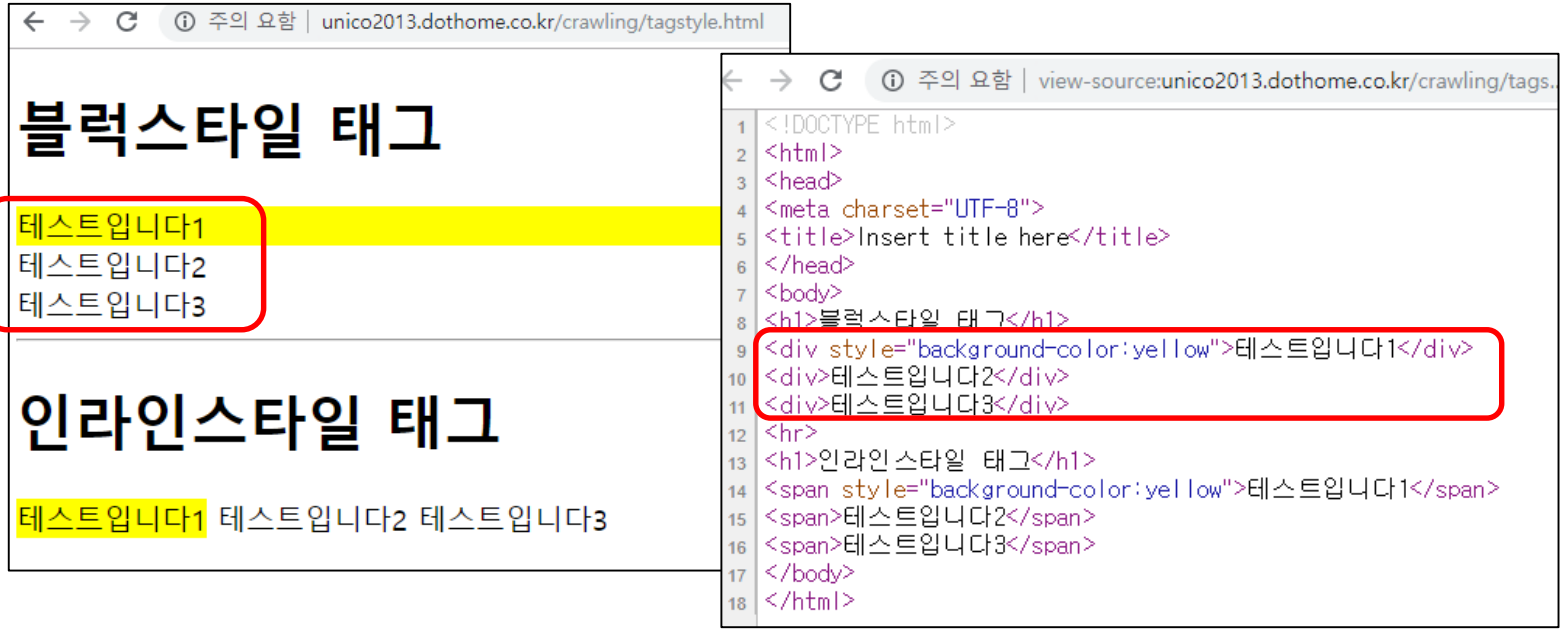


28

데이터 수집

◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지
  다음은 정적 웹 페이지의 예이다. 화면에 랜더링된 각 태그들의 컨텐트가 페이지의 소스에서도 모두 보여진다.

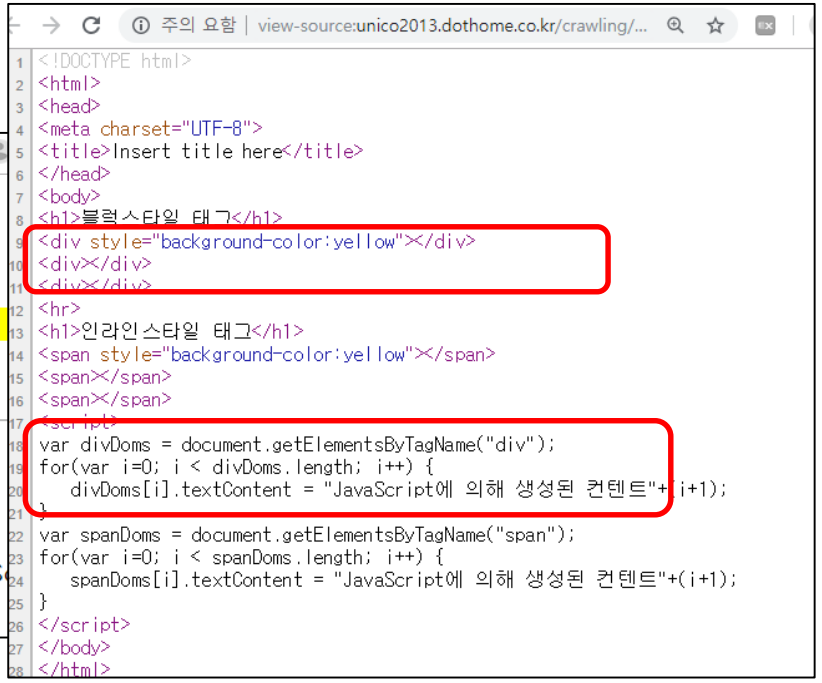


29

데이터 수집

◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지
  다음은 동적 웹 페이지의 예로서 화면에 랜더링된 일부 태그들의 컨텐트를 페이지의 소스에서 찾아볼 수 없다. 태그나 태그처럼 소스코드에서 그 내용을 찾아볼 수 없는 태그의 컨텐트는 동적 컨텐트이다.



30

데이터 수집

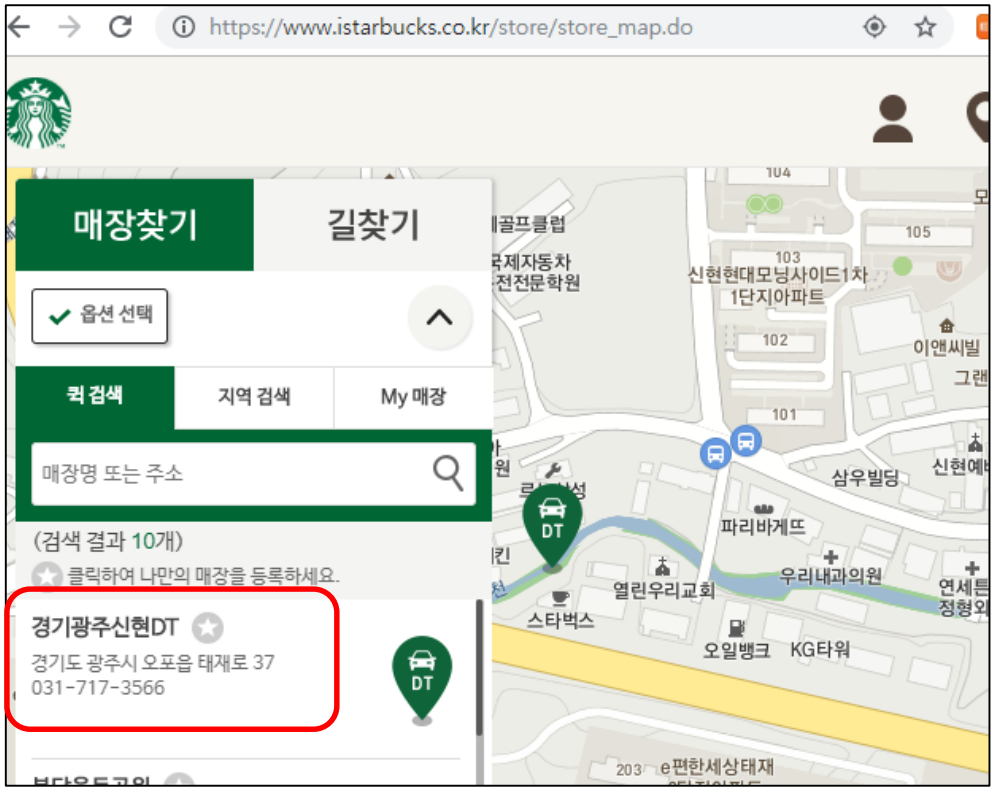
◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지

[정적 컨텐트인지 동적 컨텐트인지 채크]

다음은 스타벅스 웹 사이트의 매장 정보를 제공하는 웹 페이지이다. 이 페이지에서 “경기광주신현DT”라는 매장명과 전화번호 그리고 주소 등을 스크래핑하려고 한다.

우선 이 내용들이 정적 정적 컨텐트인지 채크한다.



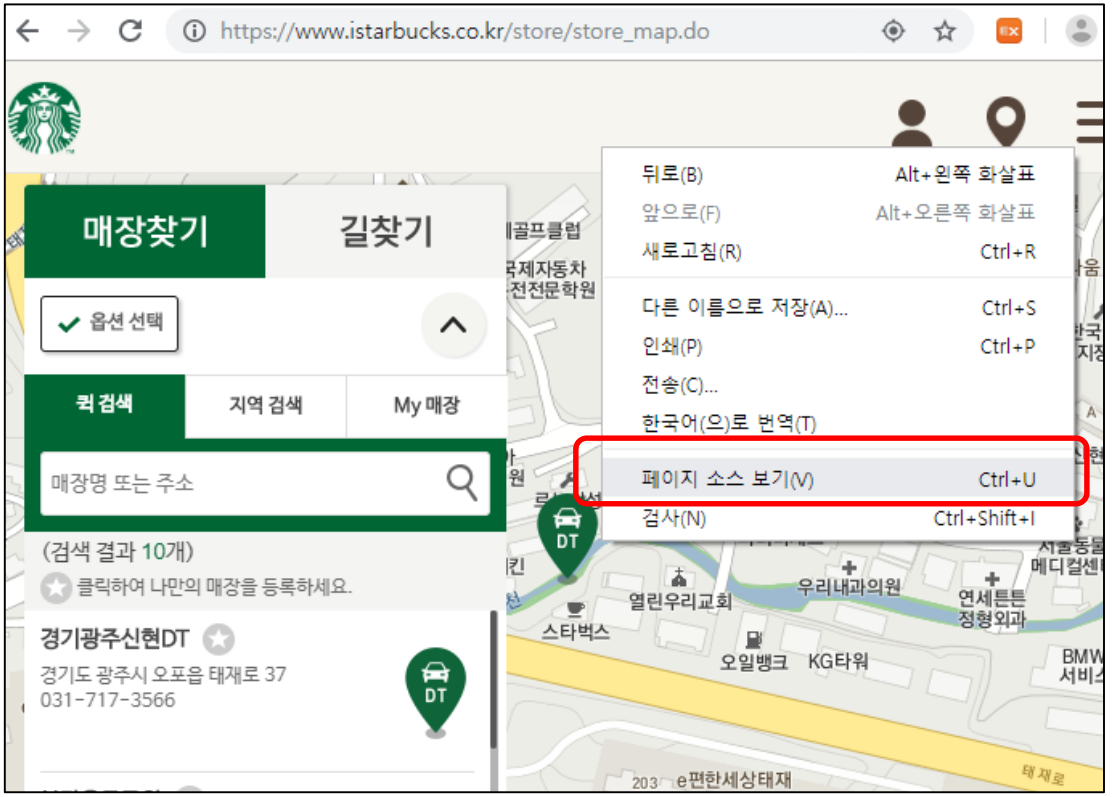
31

데이터 수집

◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지

정적 컨텐트인지 확인하려면 소스 코드를 점검해야 하므로 페이지의 적당한 위치에서 마우스 오른쪽 버튼을 클릭하면 옆의 그림과 같이 팝업메뉴가 출력됩니다. 여기에서 페이지 소스 보기 메뉴를 클릭한다.



32

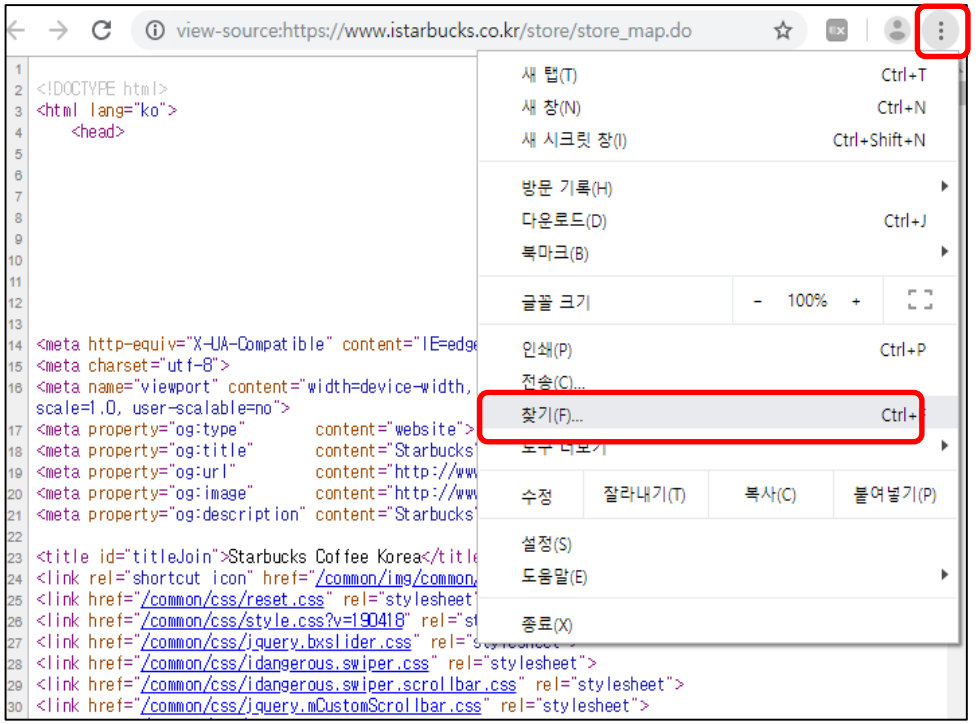
데이터 수집

◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지

그러면 오른쪽 그림과 같이 소스창이 출력됩니다.

오른쪽 상단의 “Chrome 맞춤설정 및 제어“ 메뉴를 클릭하고 풀다운메뉴가 출력되면 찾기 메뉴를 선택한다.



33

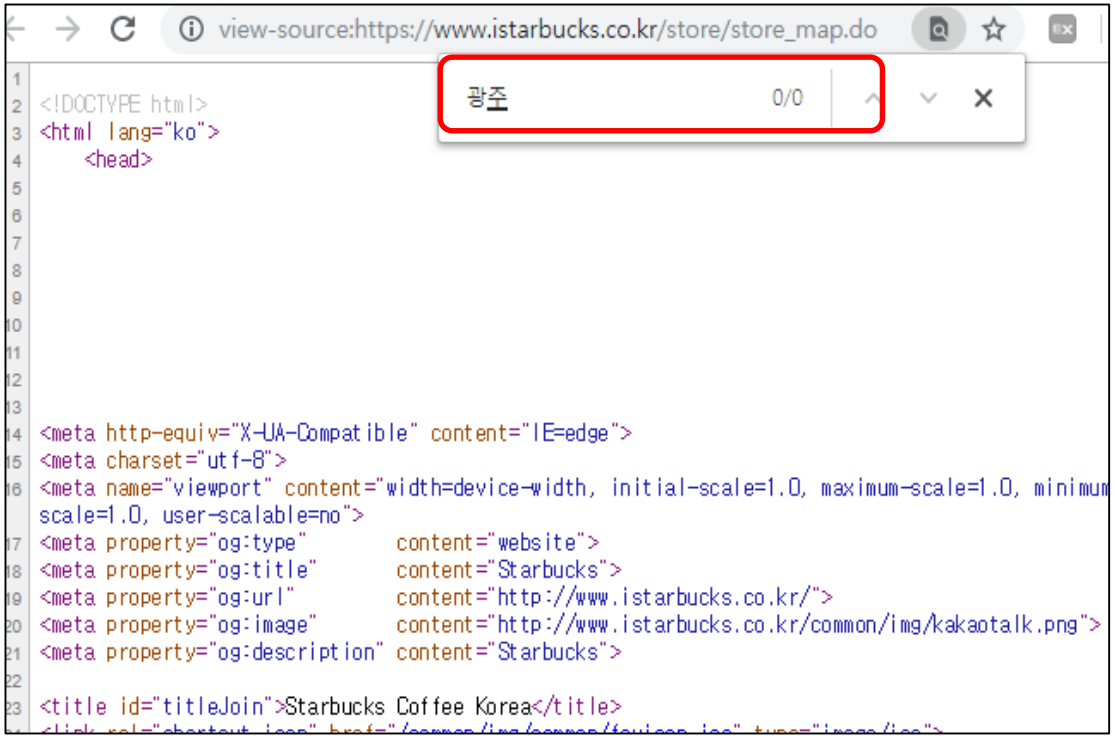
데이터 수집

◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지

광주를 입력해보면 0/0 을 출력하여
이 단어가 소스 코드에 존재하지
않음을 알려준다.

이 내용을 소스코드에서 찾을 수 없으니
JavaScript 코드에 의해서 생성되는
동적 컨텐트이다.



34

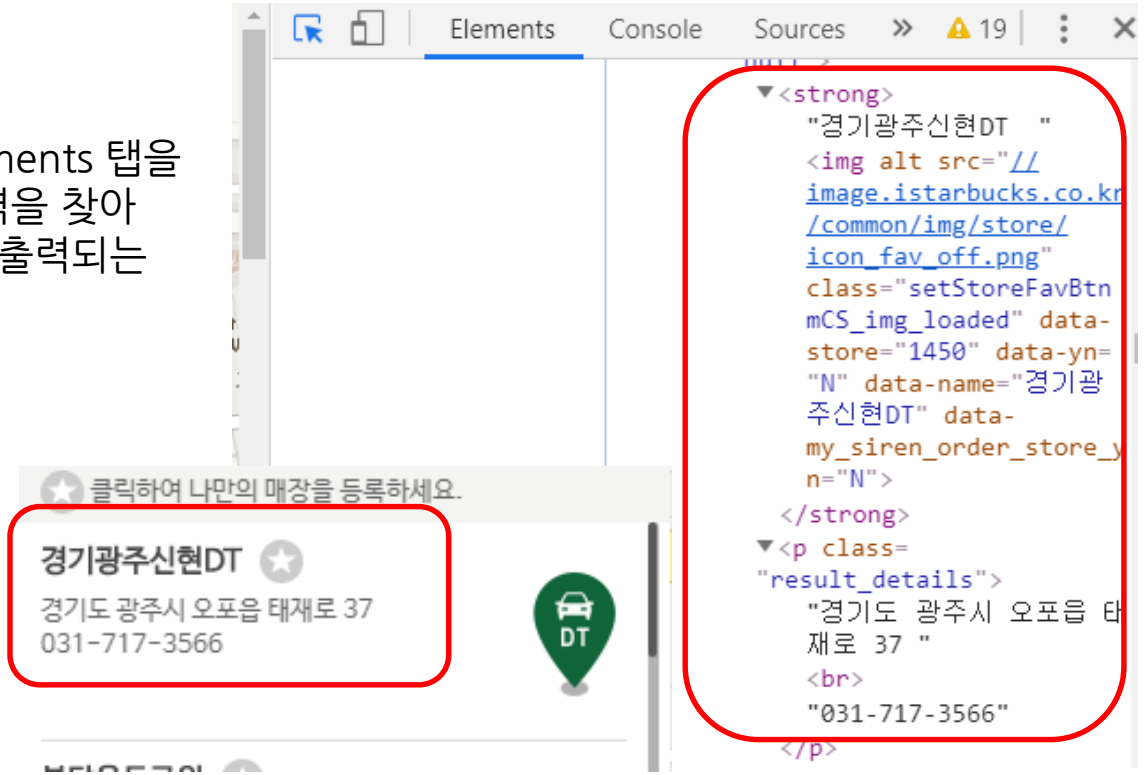
데이터 수집

◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지

크롬 브라우저의 개발자 도구에서 Elements 탭을 선택한 후에 랜더링된 내용의 태그 영역을 찾아 보면 오른쪽 결과와 같이 태그 정보가 출력되는 것을 볼 수 있다.

서버로 부터 전송된 소스에는 없는데
랜더링된 내용에는 있는거 이것이
바로 동적 컨텐트이며 이런 컨텐트를
포함하고 있는 페이지는
동적 웹 페이지라고 한다.



35

데이터 수집

◈ Selenium의 개요

- 정적 웹 페이지와 동적 웹 페이지

동적 웹 페이지에 의해 랜더링된 동적 컨텐트를 어떻게 스크래핑 할 것인가?

Selenium 이라는 웹 브라우저를 자동화 하는 도구 모음을 사용한다.

Selenium은 다양한 플랫폼과 언어를 지원하는 이용하는 브라우저 자동화 도구 모음이다. WebDriver라는 API를 통해 운영체제에 설치된 크롬이나 파이어폭스 등의 브라우저를 기동시키고 웹 페이지를 로드하고 제어한다. 브라우저를 직접 동작시킨다는 것은 JavaScript에 의해 생성되는 컨테트와 Ajax 통신등을 통해서 뒤늦게 불러와지는 컨텐트들을 처리할 수 있다는 것을 의미한다.

36

데이터 수집

다음의 경우에는 웹 페이지의 내용이 동적으로 생성되는 경우에는 지금까지의 방법으로 스크래핑 할 수 없다.
- 사용자의 선택과 같은 이벤트에 의해서 자바스크립트의 수행 결과로 컨텐츠를 생성한다.
- 페이지의 랜더링을 끝낸 후에 Ajax 기술을 이용하여 서버로 부터 컨텐트의 일부를 전송받아 동적으로 구성한다.

이러한 경우에는 Selenium 을 사용하면 제어되는 브라우저에 페이지를 랜더링 해놓고 랜더링된 결과에서 컨텐츠를 읽어올 수 있다. 뿐만 아니라 컨텐츠내에서 클릭이벤트를 발생할 수도 있으며 로그인과 같은 데이터를 입력하는 것도 가능하다.

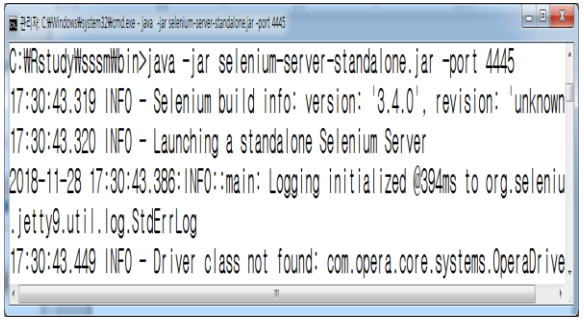
[Selenium 서버 기동 과정]

(1) selenium-server-standalone-master.zip, chromedriver.exe 를 복사한다.

(2) 적당한 디렉토리에 selenium-server-standalone-master.zip 파일의 압축을 해제한다

(3) bin 디렉토리 안에 chromedriver.exe 를 복사한다.

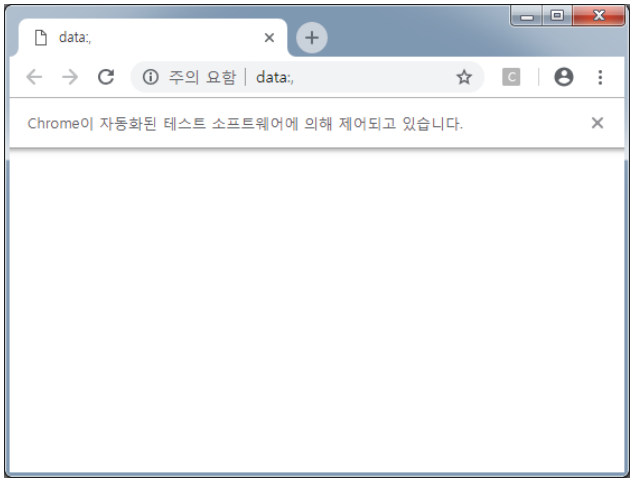
(4) Selenium 을 기동시킨다. (박스속의 명령을 CMD 창에서 실행시켜야 한다.)



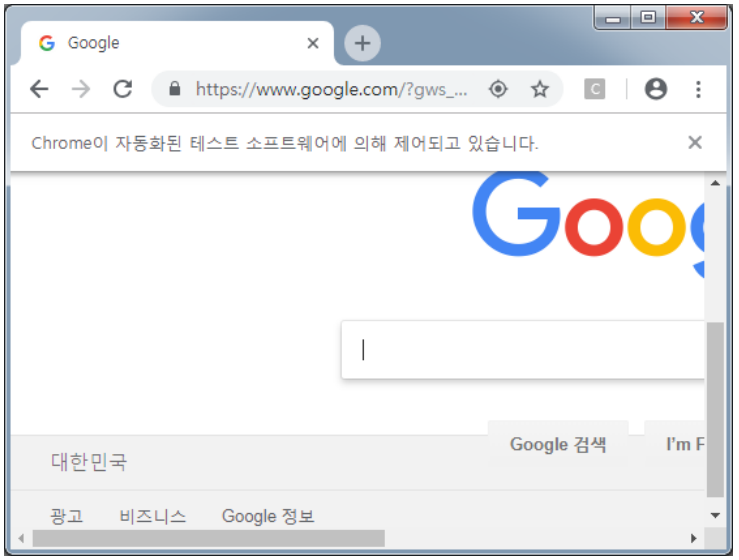
java -jar selenium-server-standalone.jar -port 4445

데이터 수집

```
install.packages("RSelenium")
library(RSelenium)
remDr <- remoteDriver(remoteServerAddr = "localhost" ,
                      port = 4445, browserName = "chrome")
remDr$open()
```

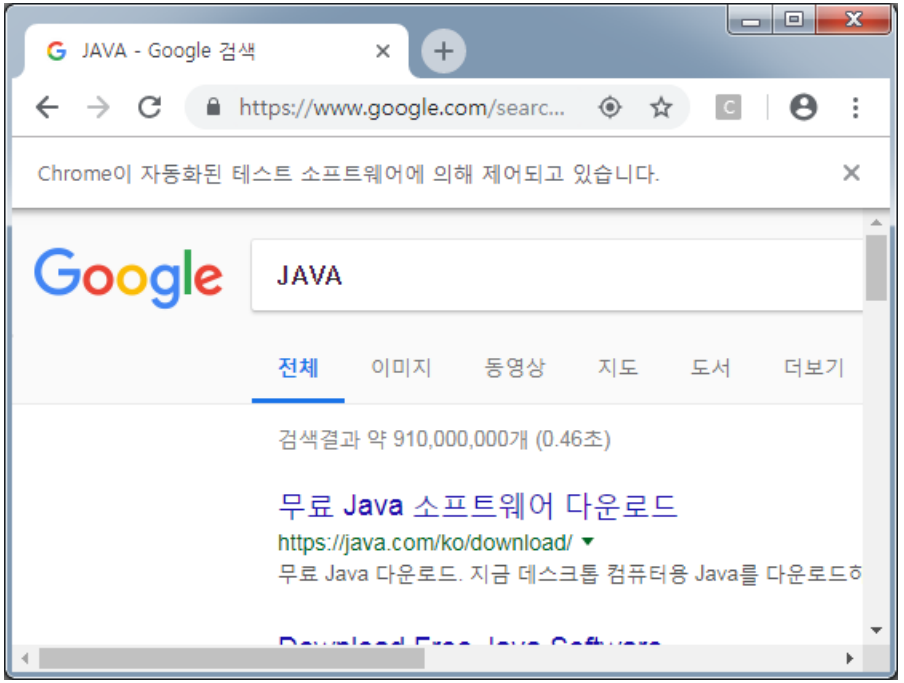



```
remDr$navigate("http://www.google.com/")
```



데이터 수집

```
webElem <- remDr$findElement(using = "css", "[name = 'q']")
webElem$sendKeysToElement(list("JAVA", key = "enter"))
```



데이터 수집

[API 소개]

remDr <- remoteDriver(remoteServerAddr="localhost",port=4445,browserName="chrome")	Selenium 서버에 접속하고 remoteDriver 객체 리턴
remDr$open()	크롬브라우저 오픈
remDr$navigate(url)	페이지 랜더링
doms <- remDr$findElements(using ="css selector", "컨텐트를추출하려는태그의선택자")	태그를 찾자
sapply(doms,function(x){x$getElementText()})	찾아진 태그들의 컨텐트 추출
more <- remDr$findElements(using='css selector', '클릭이벤트를강제발생시키려는태그의선택자')	태그를 찾자
sapply(more,function(x){x$clickElement()})	찾아진 태그에 클릭 이벤트 발생
webElem <- remDr$findElement("css", "body") remDr$executeScript("scrollTo(0, document.body.scrollHeight)", args = list(webElem))	페이지를 아래로 내리는(스크롤) 효과