

# R로 웹 데이터를 가져오는 4가지 방법(은 크롤링)

<https://mrchypark.github.io/getWebR>

[github] [pdf버전] [문의하기] [의견 및 오류 신고]

박찬엽

2017년 10월 18일

# 발표자소개

질문 / 상담 / 잡담 대환영!

## 박찬엽



- 서울도시가스 선행연구팀 연구원
- 패스트 캠퍼스 **중급R프로그래밍** 강의
- R 네이버 뉴스 크롤러 **N2H4** 관리자
- **KAKAO**@알코홀릭 R 질문방 운영
- **FACEBOOK**@mrchypark
- **GITHUB**@mrchypark

R로 웹 데이터를 가져오는 4가지 방법

# 웹에 있는 데이터를 가져오는 단계

요청 - 추출 - 저장

반복 - 예외처리 - 최적화

# 관련 R 패키지 및 함수

- 요청 : curl, httr, rvest, RSelenium
- 정리 : 정규표현식, jsonlite, rvest
- 저장 : write\*()
- 반복 : for, parallel
- 예외처리 : try, if
- 최적화 : profvis, microbenchmark

# 관련 R 패키지 및 함수

- 요청 : curl, httr, rvest, RSelenium
- 정리 : 정규표현식, jsonlite, rvest
- 저장 : write\*()
- 반복 : for, parallel
- 예외처리 : try, if
- 최적화 : profvis, microbenchmark

오늘 이야기 할 것

요청(4가지)과 정리

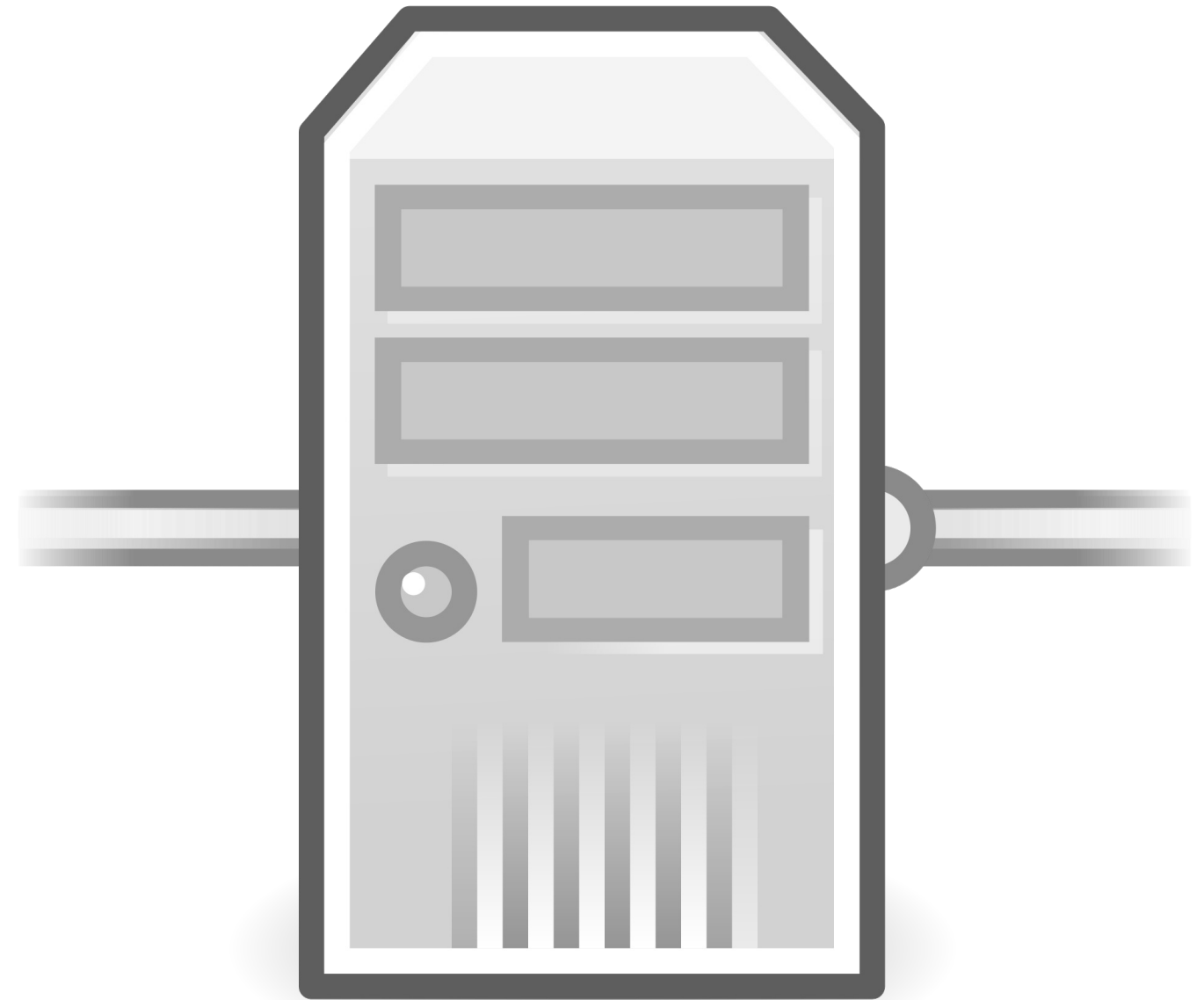
메인과 에피타이저

그럼 에피타이저 먼저!



# 서버가 하는 것

외부에서 요청하면 규칙대로 정보를 제공하는 것



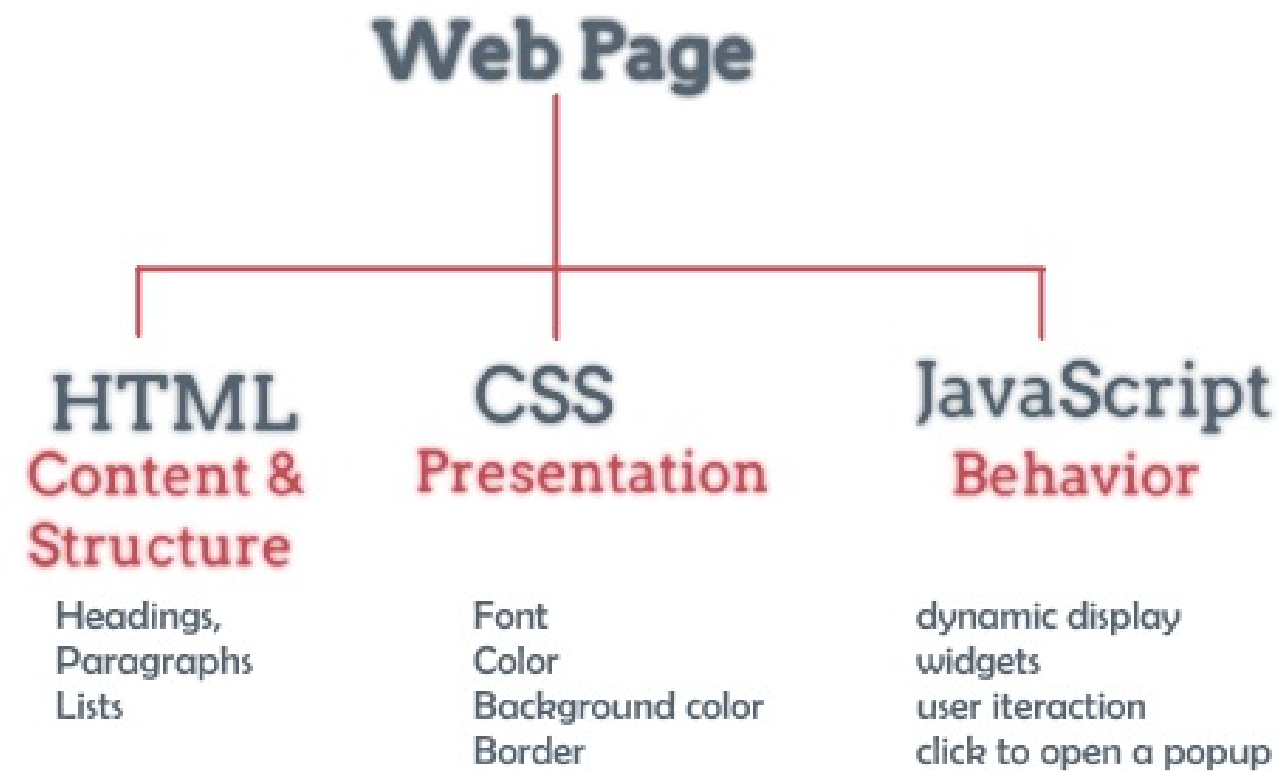
# 브라우저가 하는 것

서버가 주는 것들을 사용자에게 보여주는 것



# 웹 서버가 우리에게 주는 것

text(html, css, js, etc), image. 브라우저가 약속된 모양으로 우리에게 보여줌.



# 실제로 브라우저가 받는 파일들

The screenshot shows a web browser window displaying a news article on Naver. The article is titled "[밀착카메라] 세금 쏟아 지어놓고... 마 세트장" and is from JTBC. The browser's Network tab is open, showing a list of resources loaded by the browser. The resources include the main document, CSS, JavaScript, and various images. A waterfall chart at the top of the Network tab shows the timing of these requests.

Name	Status	Type	Initiator	Size	Time	Waterfall
read.nhn?mode=LSD&...	200	docu...	Other	127 KB	48 ms	
common.css	200	styles...	read.nhn?m...	(from...	16 ms	
news.jindo.js	200	script	read.nhn?m...	(from...	0 ms	
news.service.js	200	script	read.nhn?m...	(from...	0 ms	
dic.tooltip.js	200	script	read.nhn?m...	(from...	0 ms	
nil.news.js	200	script	read.nhn?m...	(from...	0 ms	
snb_h_entertain.png	200	png	read.nhn?m...	(from...	0 ms	
snb_h_sports.png	200	png	read.nhn?m...	(from...	0 ms	
snb_h_newsstand.png	200	png	read.nhn?m...	(from...	0 ms	
snb_h_weather.png	200	png	read.nhn?m...	(from...	0 ms	

124 requests | 3.9 MB transferred | Finish: 4.01 s | DOMContentLoaded: 818 ms | Load: 987 ms

## \* 그럼 web api는 뭔가?

web으로 제공되는 Application Programming Interface

함수인데 외부 서버에서 동작하여 웹 기술로 결과를 받는 것

# 우리가 필요한 것

text(html) 중 일부만(ex>제목)

The image shows a web browser window displaying a news article on Naver. The article title is "전공 개방하고 학과 장벽 낮춰 통섭" (Opening majors and lowering barriers for interdisciplinary integration). The browser's developer tools are open, showing the HTML structure of the article header. The selected element is an 

### tag with the class "font1 tts\_head" and the id "articleTitle". The text of the article is visible in the background, discussing the importance of interdisciplinary education and the need to break down barriers between different fields of study.

News article title: 전공 개방하고 학과 장벽 낮춰 통섭

Article content snippet: 김창수 중앙대 총장  
전공만으로 안 돼 폭넓은 교양교육  
우버처럼 공유 교육 모델 만들 것  
취업·창업·학업 다 잡는 '트리플 업'  
인구절벽 등 난제 해결에도 앞장

Developer tools HTML snippet:

```
<script type="text/javascript">...</script>
<script type="text/javascript">...</script>
<!-- test -->
<div class="article_ctrl">...</div>
<!-- 기사 헤더 -->
<div class="article_header">
  <div class="press_logo">...</div>
  <div class="article_info">
    ...
    <h3 class="font1 tts_head" id="articleTitle">전공 개방하고 학과
    장벽 낮춰 통섭 인재 키우겠다</h3> == $0
    <!-- 영문뉴스 듣기 -->
    <div class="tts_svc">...</div>
    <div id="tts_player_div" style="display: block;"></div>
    <script type="text/javascript">...</script>
  </div>
</div>
```

Selected element: #wrap table tbody tr td #main\_content div div h3#articleTitle.font1.tts\_head

Styles panel: element.style { }

# 그럼 이제 정리(에피타이저)를 설명

html 문서안에 글자 중 필요한 것만 가져오기

1번 글자를 다루는 강력한 방법 : 정규표현식 하지만 어려움

2번 xml의 node를 다루는 패키지 : rvest



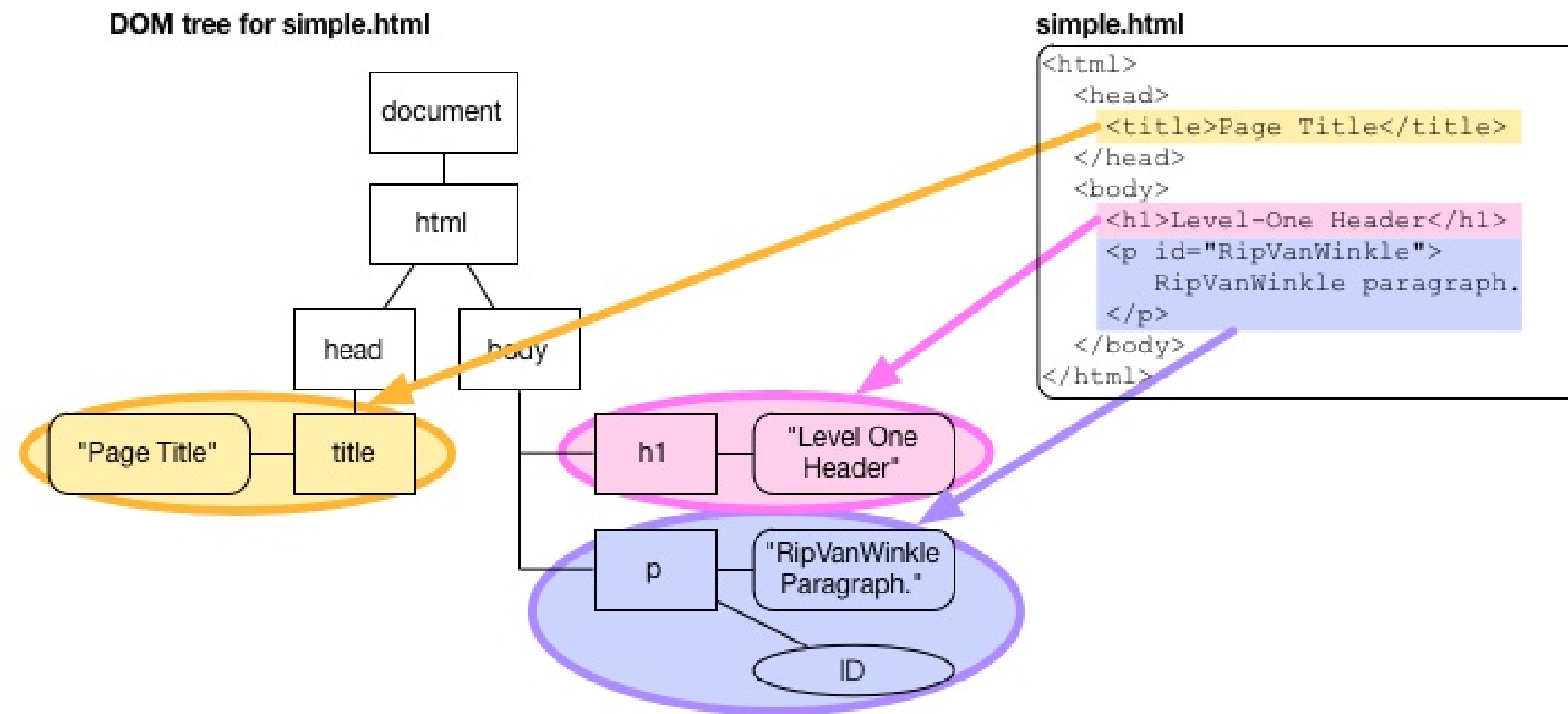
정규표현식은 검색해보세요. r에서는 `stringr`이라는 서포트 패키지가 있음.

rvest

node, attr, text만 기억하면 됨.

# node란

html에서 tag라고 불리는 것.



# 그럼 html이란

xml양식으로 작성된 웹 브라우저들이 이해하는 표준 문서

simple.html

```
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Level-One Header</h1>
    <p id="RipVanWinkle">
      RipVanWinkle paragraph.
    </p>
  </body>
</html>
```

A Browser Window



# attr 이란

attr은 attribute의 줄임으로 아래 예시로 tag의 attr1은 example1 임

```
<tag attr1="example1" attr2="example2"> 안녕하세요 </tag>
```

원하는 글자가 있는 노드를 지정하기 위해서

css 선택자를 공부해야 함.

원하는 글자가 있는 노드를 지정하기 위해서

css 선택자를 공부해야 함.

**css 선택자가 동작하는 방식**

1. tag 이름
2. tag의 id 속성
3. tag의 class 속성
4. tag의 custom 속성

원하는 글자가 있는 노드를 지정하기 위해서

css 선택자를 공부해야 함.

**css 선택자가 동작하는 방식**

1. tag 이름
2. tag의 id 속성
3. tag의 class 속성
4. tag의 custom 속성

**css 선택자로 node를 선택하기 위해서**

1. tag
2. #id
3. .class
4. [attr="val"]
5. tag#id
6. tag.class
7. tag[attr="val"]



# text 이란

text은 시작 태그와 종료 태그 사이에 있는 글자로, 아래 예시 기준 "안녕하세요" 를 뜻함

```
<tag attr1="example1" attr2="example2"> 안녕하세요 </tag>
```

# rvest의 동작 순서(text 가져오기)

1. html 문서 데이터 가져오기
2. 필요한 노드 선택하기
3. 노드내에 text를 가져오기

## rvest 함수

```
read_html(url)
read_html(url) %>% html_nodes("tag.class")
read_html(url) %>% html_nodes("tag.class") %>% html_text
```

# rvest의 동작 순서(attr 가져오기)

1. html 문서 데이터 가져오기
2. 필요한 노드 선택하기
3. 노드내에 attr 중에 "attr1"값을 가져오기

## rvest 함수

```
read_html(url)
read_html(url) %>% html_nodes("tag.class")
read_html(url) %>% html_nodes("tag.class") %>% html_attr("attr1")
```

# 예시

```
library(rvest)
url<-"http://news.naver.com/main/read.nhn?mode=LS2D&mid=shm&sid1=102&sid2=250&oid=025&aid=00027"
(nv<-read_html(url))
```

< >

```
## {xml_document}
## <html lang="ko">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset= ...
## [2] <body>\r\n<div id="wrap">\r\n\t\r\n\t<div id="da_base"></div>\r\n\t<
```

```
(nvns<-html_nodes(nv, "h3#articleTitle"))
```

```
## {xml_nodeset (1)}
## [1] <h3 class="font1" id="articleTitle">전공 개방하고 학과 장벽 낮춰 통섭 인재 키우겠다</h3>
```

```
(title<-html_text(nvns))
```

```
(class<-html_attr(nvns, "class"))
```

```
## [1] "전공 개방하고 학과 장벽 낮춰 통섭 인재 키우겠다" ## [1] "font1"
```

rvest는 html 문서로 되어 있는 웹에서의 텍스트 데이터를 가져와서 처리하는 패키지

이제 메인! 요청하기  
(read\_html이 방법 1 ㅋㅋ)

그래서 우리가 알아야 할 것

GET과 POST

방금 read\_html 함수는 GET 방식으로 서버에 요청하여 html 문서를 받은 것.



http 표준 요청을 수행해 주는 **httr** 패키지

# GET 요청

`read_html(url) == content(GET(url))` 인걸로 GET 요청으로 html 문서를 가져올 때는 `read_html()` 함수가 함께 처리해줌.

```
library(httr)
library(rvest)
```

```
url<-"http://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=102&oid=437&aid=0000165410"
dat<-GET(url)
content(dat)
```

```
## {xml_document}
## <html lang="ko">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset= ...
## [2] <body>\r\n<div id="wrap">\r\n\t\r\n\t<div id="da_base"></div>\r\n\t< ...
```

```
read_html(url)
```

```
## {xml_document}
## <html lang="ko">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset= ...
## [2] <body>\r\n<div id="wrap">\r\n\t\r\n\t<div id="da_base"></div>\r\n\t< ...
```

# 왜 GET 함수를 써야 하는가

header, cookie 등 다양한 옵션을 사용할 수 있음.

ex> 크롤러가 아니라고 속여야 한다던가....

그럼 POST 란?

# POST 란

body에 사용자가 필요한 값을 추가해서 요청을 할 수 있는 방법

# 사용할 api

## 한글 띄어쓰기 api

띄어쓰기가 불문명한 텍스트를 전달하면 맞는 띄어쓰기 결과를 돌려줌.

# 실행

```
library(httr)

body<-list(sent="아래와같은방식으로API를사용할수있으며,호출건수에대해서별도의제한은없으나,1회 호출에200글자로 글자
res<-PUT(url='http://35.201.156.140:8080/spacing', body=body)

content(res)$sent
```

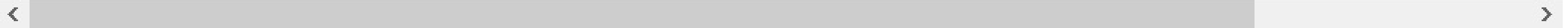
< >

## [1] "아래와 같은 방식으로 API를 사용할 수 있으며, 호출 건수에 대해서 별도의 제한은 없으나, 1회 호출에 200글자로 글자"

# body 란

list() 자료형으로 되어 있고 이름이 있는 데이터로 만든 요청에 추가할 수 있는 값

```
body<-list(sent="아래와같은방식으로API를사용할수있으며 , 호출건수에대해서별도의제한은없으나 ,1회 호출에200글자로글자수를제한하고있다."  
body
```



```
## $sent
```

```
## [1] "아래와같은방식으로API를사용할수있으며 , 호출건수에대해서별도의제한은없으나 ,1회 호출에200글자로글자수를제한하고있다."
```



# 여긴 PUT 이라고 되어 있는데?

PUT은 POST와 매우 유사한 요청 방식으로 서버에서 요구하는 방식을 지켜줘야함. 예시로 PUT을 사용했지만, 다른 POST가 필요한 곳에서는 POST 명령으로 작성하면 됨. 서비스 제공자가 설명서를 작성하게 되어있음.

```
(res<-PUT(url='http://35.201.156.140:8080/spacing', body=body))
```

```
## Response [http://35.201.156.140:8080/spacing]
##   Date: 2017-10-18 06:48
##   Status: 200
##   Content-Type: application/json
##   Size: 348 B
## {"sent": "\uc544\ub798\uc640 \uac19\uc740 \ubc29\uc2dd\uc73c\ub85c API\u...
```

# api 명세 예시

NAVER Developers - Op x

손님 — □ ×

← → ↻ 안전함 | <https://developers.naver.com/docs/common/openapiguide/#/apistlist.md#네이버-아이디로-로그인> ⋮

NAVER Developers Products Documents Application NAVER D2 Support API 상태 V Search Here 🔍 로그인

API 공통 가이드

네이버 오픈API 종류

사전 준비 사항

내 애플리케이션 관리

용어 정리

샘플 코드

오류 코드

https://openapi.naver.com/v1/nid/me

GET

JSON

네이버 회원의 프로필을 조회합니다.

블로그 ↗

다음은 블로그 API에서 사용하는 주요 요청 URL과 메서드, 응답 형식입니다.

요청 URL	메서드	응답 형식	설명
https://openapi.naver.com/blog/writePost.json	POST	JSON	네이버 블로그에 포스트를 등록합니다.
https://openapi.naver.com/blog/listCategory.json	GET	JSON	네이버 블로그의 카테고리를 조회합니다.

↑

카페 ↗

다음은 카페 API에서 사용하는 주요 요청 URL과 메서드, 응답 형식입니다.

요청 URL	메서드	응답 형식	설명
https://openapi.naver.com/v1/cafe/{clubid}/members	POST	JSON	특정 네이버 카페에 가입합니다.

# content() 함수

httr 패키지의 요청들은 read\_html() 함수와 달리 header 나 cookie 등의 모든 정보를 다 확인할 수 있음. 그래서 응답으로 서버가 데이터로써 준 내용을 확인하기 위해서 content() 함수를 제공함.

```
content(res)
```

```
## $sent
```

```
## [1] "아래와 같은 방식으로 API를 사용할 수 있으며, 호출 건수에 대해서 별도의 제한은 없으나, 1회 호출에 200글자로 글자
```

여기까지가 http 요청을 따라하는 httr 패키지(방법 2)

# 중간 정산

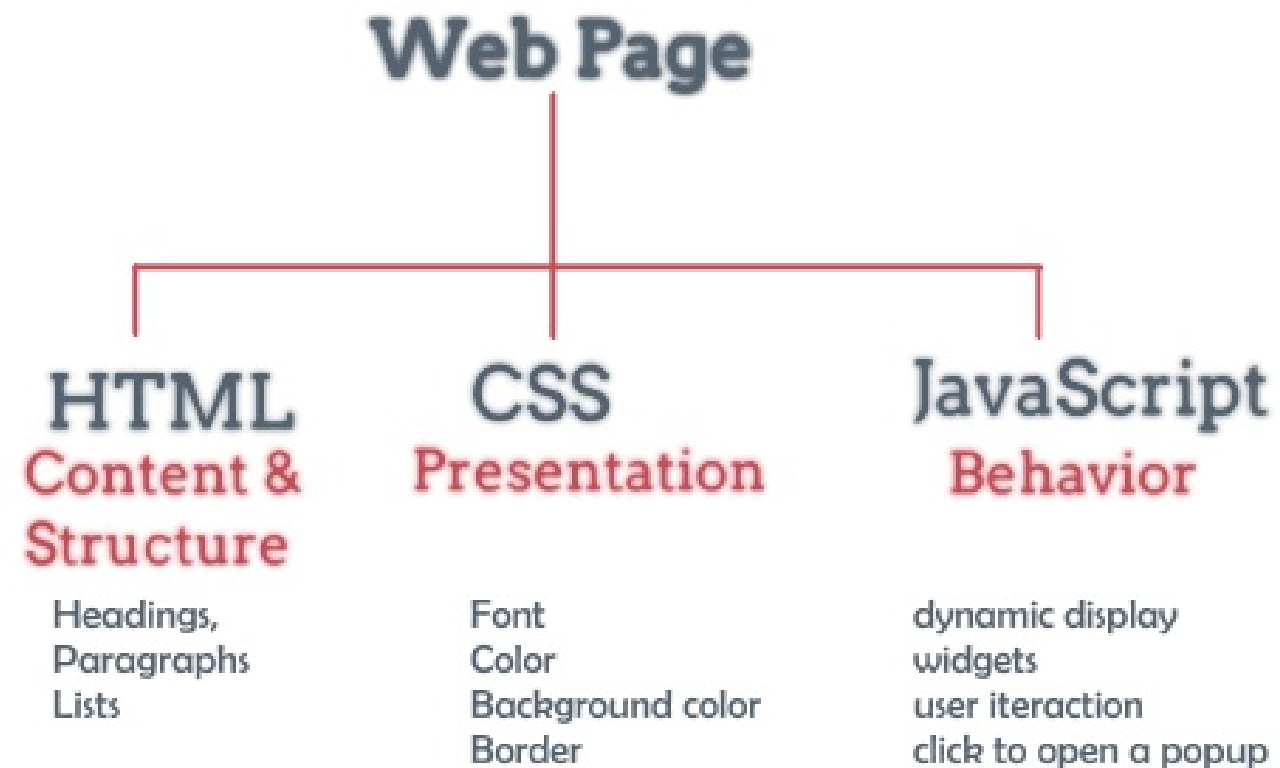
이제 정적 웹 서비스내의 글자와 web api 데이터를 가져오는 2가지 방법을 알게 됨.

이제 필요한 것

동적 웹 서비스에서 가져오기

# 동적 웹 서비스란

javascript<sup>1</sup>가 웹페이지 동작을 조절하는 것.



[1] javascript : 브라우저에서 동작하는 개발 언어로 동적 웹 서비스를 개발하는 목적으로 많이 사용함.

# 그래서 브라우저가 필요함

동적 웹 서비스내의 데이터는 브라우저가 처리해준 결과이기 때문에 브라우저와 같이 처리하는 방법을 찾던지, 브라우저를 사용하던지 해야 함.



# RSelenium

# Selenium 이란

Selenium은 코드로 브라우저를 컨트롤하는 패키지. 그래서 브라우저를 움직일 수 있다! 그걸 R에서 사용하는 **RSelenium** 패키지를 사용할 예정



어떤 브라우저를 사용할까?

phantomjs

# phantonjs 란

phantomjs는 headless 브라우저로 headless란 사람이 보는 부분이 없는 것



- 최근 chrome도 headless를 자체적으로 지원하기 시작함.

# 실행

```
library(RSelenium)
pJS <- wdman::phantomjs(port = 4567L)
```

```
## checking phantomjs versions:
```

```
## BEGIN: PREDOWNLOAD
```

```
## BEGIN: DOWNLOAD
```

```
## BEGIN: POSTDOWNLOAD
```

```
remDr <- remoteDriver(port=4567L, browserName = 'phantomjs')
remDr$open()
```

```
## [1] "Connecting to remote server"
```

```
## $browserName
```

```
## [1] "phantomjs"
```

```
##
```

```
## $version
```

```
## [1] "2.1.1"
```

```
##
```

```
## $driverName
```

```
## [1] "ghostdriver"
```

```
##
```

```
## $driverVersion
```

```
## [1] "1.2.0"
```

# 실행

```
remDr$navigate("http://www.google.com/ncr")  
remDr$getTitle()[[1]]
```

```
## [1] "Google"
```

```
remDr$close()  
pJS$stop()
```

```
## [1] TRUE
```

# 시연

키 입력과 마우스 클릭을 확인

시연코드 가기

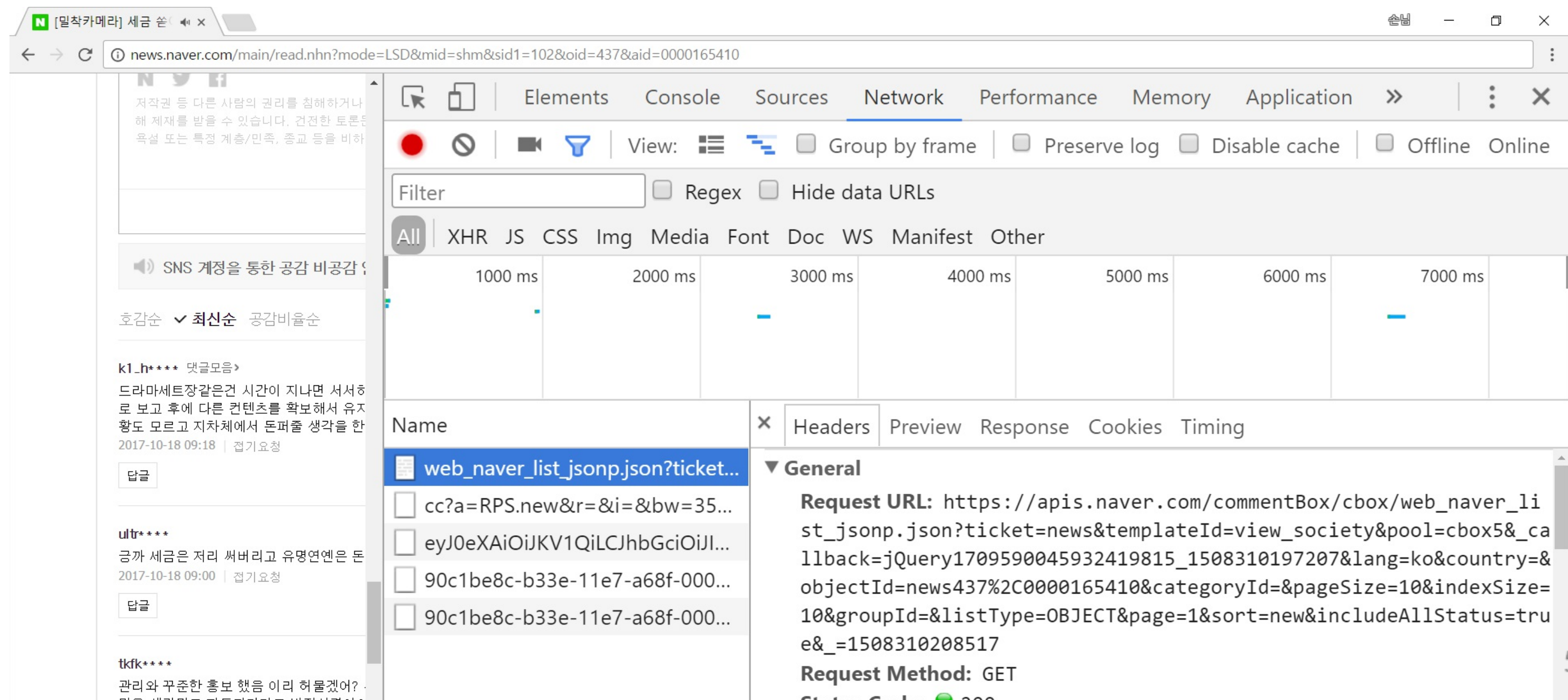
RSelenium이란 브라우저를 컨트롤하는 패키지(방법 3)



마지막으로...

# +고급

크롬 개발자 도구를 이용하면 js가 가져오는 api를 찾아서 더 빠르게 정보를 가져올 수 있음



The screenshot shows a web browser window with the URL `news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=102&oid=437&aid=0000165410`. The Chrome DevTools Network tab is open, displaying a list of requests. The first request, `web_naver_list_jsonp.json?ticket=...`, is selected. The details pane on the right shows the following information:

- Request URL:** `https://apis.naver.com/commentBox/cbox/web_naver_list_jsonp.json?ticket=news&templateId=view_society&pool=cbox5&callback=jQuery1709590045932419815_1508310197207&lang=ko&country=&objectId=news437%2C0000165410&categoryId=&pageSize=10&indexSize=10&groupId=&listType=OBJECT&page=1&sort=new&includeAllStatus=true&_=1508310208517`
- Request Method:** GET

The left pane of the Network tab shows a timeline of requests with a duration of 1000 ms. The right pane shows the details of the selected request, including the Request URL, Request Method, and other information.

# GET 요청

찾아낸 주소에 GET() 요청을 시도함(은 실패). 이런 웹 서비스 내에서 사용하는 api의 경우 OpenAPI가 아니기 때문에 설명서 없고, 다양한 시도를 통해서 사용법을 찾아야 함.

```
url<-"https://apis.naver.com/commentBox/cbox/web_naver_list_jsonp.json?ticket=news&templateId=v  
con <- httr::GET(url)  
tt <- httr::content(con, "text")  
tt
```

```
## [1] "jQuery1707377572341505474_1508264183038({\"success\":false,\"code\":\"3999\",\"message\":\"잘못된
```

# 추가 정보 제공

네이버 뉴스 댓글의 경우, referer라는 정보가 요청 header에 있어야만 정상 동작함

```
url<-"https://apis.naver.com/commentBox/cbox/web_naver_list_jsonp.json?ticket=news&templateId=v  
ref<-"http://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=102&oid=437&aid=0000165410"  
con <- httr::GET(url,  
                  httr::add_headers(Referer = ref))  
tt <- httr::content(con, "text")  
tt
```

```
## [1] "jQuery1707377572341505474_1508264183038({\"success\":true,\"code\":\"1000\", \"message\":\"요청을
```

# json 파싱

표준 json의 경우는 httr 패키지의 content() 함수가 자동으로 list 자료형으로 변환해주나 네이버 댓글의 경우 표준과 모양이 달라서 fromJSON() 함수가 에러 발생.

```
url<-"https://apis.naver.com/commentBox/cbox/web_naver_list_jsonp.json?ticket=news&templateId=v
ref<-"http://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=102&oid=437&aid=0000165410"
con <- httr::GET(url,
                  httr::add_headers(Referer = ref))
tt <- httr::content(con, "text")
jsonlite::fromJSON(tt)
```

```
Error: lexical error: invalid char in json text.
                                jQuery1707377572341505474_15082
                                (right here) -----^
```

```

url<-"https://apis.naver.com/commentBox/cbox/web_naver_list_jsonp.json?ticket=news&templateId=v
ref<-"http://news.naver.com/main/read.nhn?mode=LSD&mid=shm&sid1=102&oid=437&aid=0000165410"
con <- httr::GET(url,
                httr::add_headers(Referer = ref))
tt <- httr::content(con, "text")

tt <- gsub("(;|\n|_callback|jQuery1707377572341505474_1508264183038)", "", tt)
tt <- gsub("\\(", "[", tt)
tt <- gsub("\\)", "]", tt)

data <- jsonlite::fromJSON(tt)
data$result$commentList[[1]]$contents

```

```

## [1] "드라마세트장같은건 시간이 지나면 서서히 시들게 되있음 그럼 거기다 돈을 투자하려면 장기적으로 보고 후에 다른 컨텐츠
## [2] "금까 세금은 저리 써버리고 유명연엔은 돈 쓸어가고..방송국서 외국도 꿈으로 보내주고..."
## [3] "관리와 꾸준한 홍보 했음 이리 허물겠어? 주민들도 그래 지들이 관광객 받을 생각이였음 누워서 떡먹을 생각말고 지
## [4] "드라마에서 못벗어나는 한심한 민족!"
## [5] "임기내에 뭔가 보이는 업적 쌓아보겠다고 이것저것 많이 쳐 만드는데 막상 시간지나면 쓸데없어진게 대다수고 그런데서
## [6] "그래서 다스는 누구꺼가요?"
## [7] "영화찍을때 화재 연기좀 만나게 합시다 매연과 미세먼지 보기만 해도 숨막혀여~"
## [8] "이게 다 드라마랑 연예인에 환장한 계집들 때문이다. 제발 정치와 경제에 관심을 가져라!"
## [9] "헐리웃 영화나 일본 애니메이션처럼 수십년을 사랑 받은 것도 아니고 반짝 드라마 세트장에 안일하게 세금 투입 ㅋㅋㅋ
## [10] "청주 제빵왕김탁구세트장이랑 영광의 제인 빵집은 지금 어린이집과 유치원 빵만드는 시설과 빵집 국수집 같이하고 있는

```

끝!

<https://mrchypark.github.io/getWebR>

[github] [pdf버전] [문의하기] [의견 및 오류 신고]