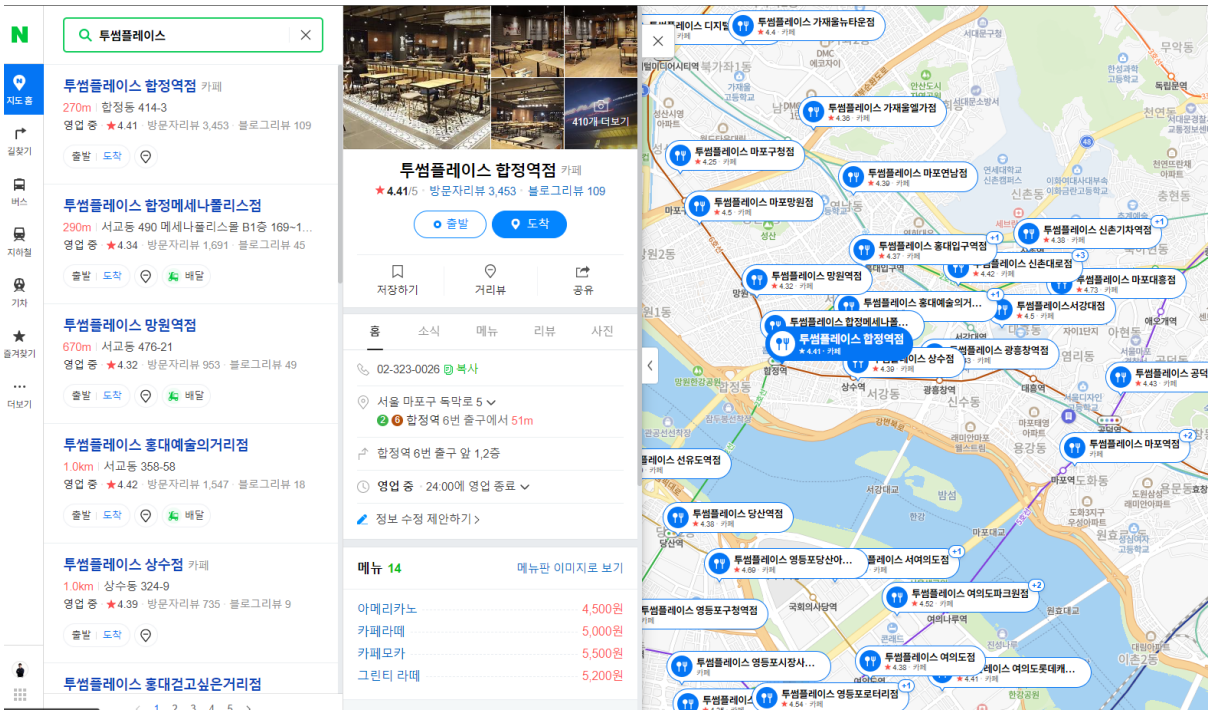
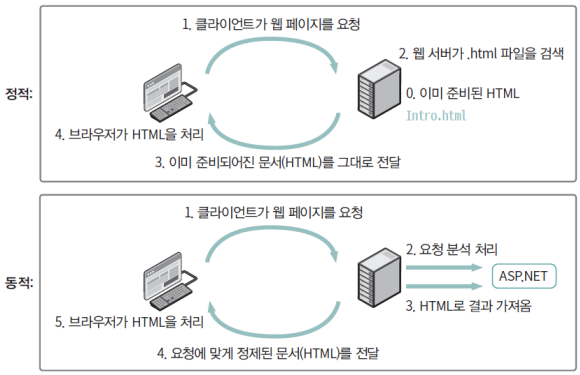
동적 크롤링 (R)

수업일	@2023년 5월 16일
완료	☐
주차	11주차

일반적인 크롤링으로는 정적 데이터, 즉 변하지 않는 데이터만을 수집할 수 있다. 한 페이지 안에서 원하는 정보가 모두 드러나는 것을 정적 데이터라 한다. 반면 입력, 클릭, 로그인 등을 통해 데이터가 바뀌는 것을 동적 데이터라 한다. 예를 들어 네이버 지도에서 매장을 검색을 한 후 좌측에서 원하는 선택할 때 마다 이에 해당하는 내용이 뜬다.



이는 웹페이지에서 사용자가 클릭 등과 같은 조작을 하면 AJAX 호출이 발생하여 그 결과가 페이지의 일부분에만 반영되어 변경되기 때문이다. 즉 매장을 클릭하면 웹브라우저가 연결된 자바스크립트 코드를 실행하여 해당 매장의 상세 정보가 동일한 페이지에 동적으로 표시된다. 아래 그림은 정적 페이지와 동적 페이지의 작동 방식의 차이를 나타낸다.



셀레니움을 이용할 경우 정적 페이지와 동적 페이지를 모두 크롤링 할 수 있다는 강력함이 있지만, 상대적으로 속도가 느리다. 따라서 정적 페이지는 기존의 방법을 이용한 크롤링을, 동적 페이지는 셀레니움을 이용한 크롤링을 하는 것이 일반적이다.

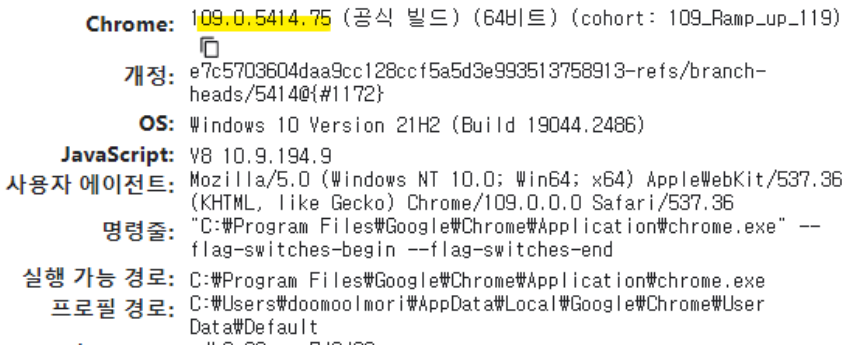
셀레니움이란 다양한 브라우저(인터넷 익스플로러, 크롬, 사파리 오페라 등) 및 플랫폼에서 웹 응용 프로그램을 테스트 할 수 있게 해주는 라이브러리다. 즉 웹 자동화 테스트 용도로 개발이 되었기에 실제 브라우저를 사용하며, 페이지가 변화하는 것도 관찰이 가능하기에 동적 크롤링에 사용할 수 있다.

세팅하기

먼저 셀레니움을 사용하기 위해서는 자바가 설치되어야 한다.

> Windows용 Java 다운로드
>
> Oracle Java 라이센스는 2019년 4월 16일 릴리스부터 변경되었습니다. Oracle Java SE에 대한 Oracle Technology Network 라이센스 합의서는 이전 Oracle Java 라이센스와는 상당히 다릅니다. 이 라이센스는 개인 용도 및 개발 용도와 같은 특정 목적의 무료 사용은 허용하지만, 이전 Oracle Java 라이센스에서 권한이 부여된 기타 사용은 더 이상 허용되지 않습니다.
>
> https://www.java.com/ko/download/ie_manual.jsp?locale=ko

그 후 크롬에서 `chrome://version` 을 통해 본인 브라우저의 버젼을 확인한다.



R에서 `binman::list_versions('chromedriver')` 를 입력하면 본인 PC에 설치된 크롬 드라이버의 버젼을 확인할 수 있다. 이 중 위에서 확인한 버젼 정보가 앞의 2개 혹은 3개 (예: 190.0.5414)가 일치하는 버젼을 찾는다.

```
library(wdman)
library(binman)
library(RSelenium)
library(webdriver)

binman::list_versions('chromedriver')
```

```
$win32
[1] "100.0.4896.20" "108.0.5359.22" "108.0.5359.71" "109.0.5414.25" "109.0.5414.74" "110.0.5481.30" "99.0.4844.35"
[8] "99.0.4844.51"
```

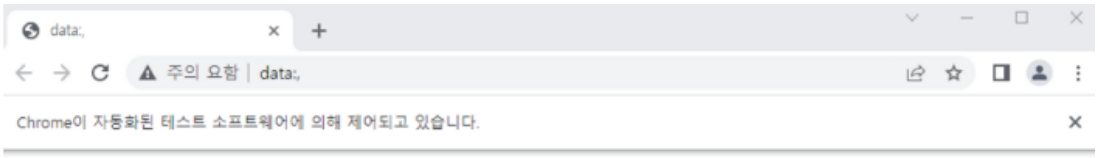
"109.0.5414.25" 혹은 "109.0.5414.74"이 매치된다. 해당 버젼의 크롬 브라우저를 연다.

```
chrome(port = 4567L, version = '109.0.5414.25')
```

4567번 포트에 크롬 드라이버가 열렸다. 이제 셀레니움 서버를 연다.

```
rs_driver_object = rsDriver(browser = 'chrome',
                            chromever = '109.0.5414.25',
                            port = 4567L)
```

- browser:사용할 브라우져,
- chromever: 크롬 버젼
- port: 포트번호



이러한 페이지가 열리면 성공적으로 셀레니움이 실행된 것이다. 이제 페이지를 닫은 후 클라이언트 객체에 접근한다.

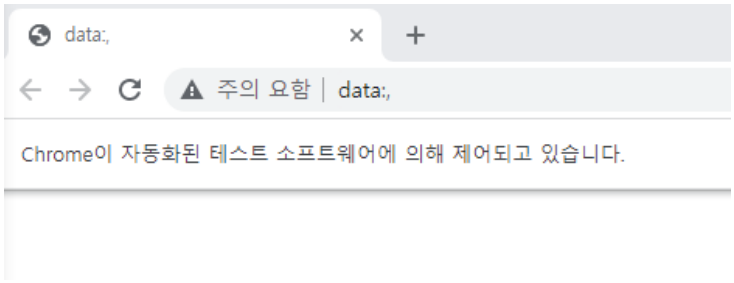
```
remDr = rs_driver_object$client
remDr
```

```
$remoteServerAddr
[1] "localhost"

$port
[1] 4567

$browserName
[1] "chrome"

$version
[1] ""
```

```
$platform
[1] "ANY"

$javascript
[1] TRUE

$nativeEvents
[1] TRUE

$extraCapabilities
list()
```

셀레니움에 크롬 브라우저가 연결되었다.

```
remDr$open()
```



위 코드를 치면 크롬 브라우저가 열리며 Chrome이 자동화된 테스트 소프트웨어에 의해 제어되고 있습니다.'라는 문구가 뜬다. 이제 R 코드를 통해 웹페이지를 조작할 수 있다.

```
remDr$navigate('https://www.naver.com')
remDr$getPageSource
```

`remDr$navigate()` 내부에 URL을 입력하면 이에 해당하는 페이지로 이동한다. 또한 `$getPageSource` 를 통해 HTML 정보를 확인할 수도 있다.

이제 네이버 메인에서 [뉴스]버튼을 누르는 동작을 실행해보자. 개발자도구 화면을 통해 확인해보면 [뉴스] 탭은 아래 HTML에 위치하고 있다.


```
remDr$findElement(using = 'xpath', '//*[@id="NM_FAVORITE"]/div[1]/ul[2]/li[2]/a')$clickElement()
```

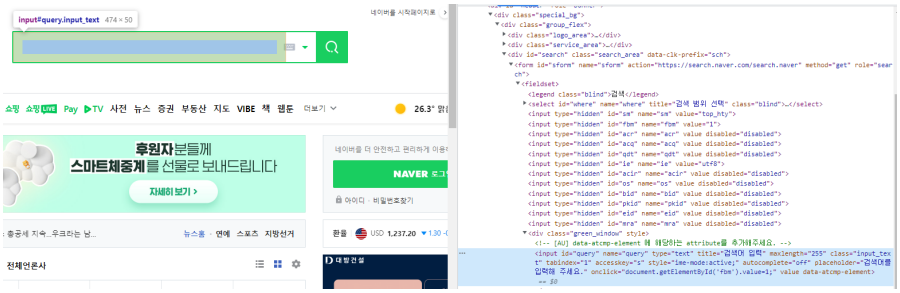
브라우저 상에서 보이는 버튼, 검색창, 사진, 테이블, 동영상 등을 엘레먼트(element, 요소)라고 한다. `findElement()`는 다양한 방법으로 엘레먼트에 접근하게 해주며, using을 통해 어떠한 방법으로 엘레먼트에 접근할지 선언한다. 여기서는 xpath를 통해 해당 부분에 접근하였다. `clickElement()`는 마우스 클릭을 실행하며 결과 적으로 뉴스 탭을 클릭한 후 페이지가 이동되는 것을 확인할 수 있다.

이제 뒤로가기를 실행해보도록 하자.

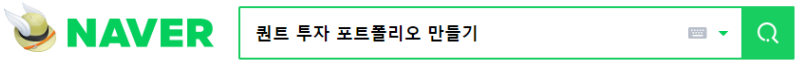
```
remDr$goBack()
```

goBack()은 뒤로가기를 의미하며, 기존 페이지던 네이버 메인으로 이동한다.

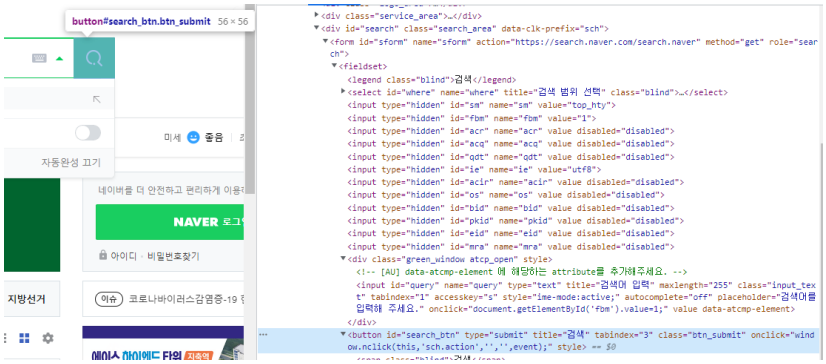
이제 특정 검색어를 검색하는 방법에 대해 알아보자. 먼저 검색창의 위치가 어디에 있는지 확인해보면 query라는 id와 input_text라는 class에 위치하고 있다.




```
remDr$findElement(using = 'class', 'input_text')$sendKeysToElement(list("퀀트 투자 포트폴리오 만들기"))
```



findElement 내에 **class**를 입력하면 클래스 명이 있는 엘레먼트에 접근하며, 여기서는 검색창에 접근한다. 그 후 `sendKeysToElements()` 내에 리스트 형태로 텍스트를 입력하면 해당 내용이 웹페이지에 입력된다. 이제 웹페이지에서 검색 버튼 해당하는 돋보기 모양을 클릭하거나 엔터키를 누르면 검색이 실행된다. 먼저 돋보기 모양의 위치를 확인해보면 **search_btn** id와 **btn_submit** 클래스에 위치하고 있다.

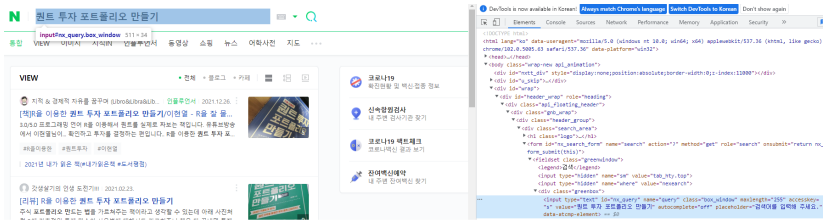


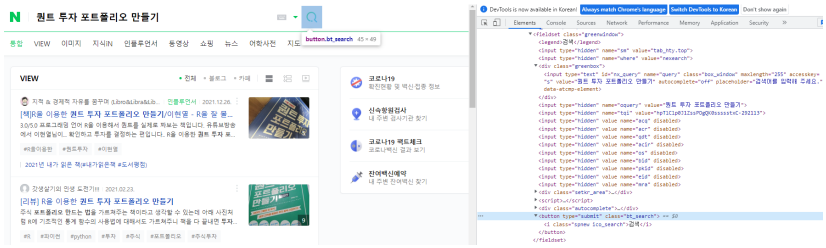

```
remDr$findElement(using = 'class', 'btn_submit')$clickElement()
```

`findElement(using = 'class', 'btn_submit')`를 통해 검색 버튼에 접속한다. 그 후 `clickElement()` 를 통해 클릭 동작을 수행한다. 페이지를 확인해보면 검색이 실행된 후 결과를 확인할 수 있다.

이번에는 다른 단어를 검색해보도록 하자. 웹에서 기존 검색어 내용을 지운 후, 검색어를 입력하고, 버튼을 클릭해야 한다. 이를 위해 검색어 박스의 위치를 찾아보면 다음과 같다.

- 검색어 박스: box_window 클래스
- 검색 버튼: bt_search 클래스





```
remD$findElement(using = 'class', 'box_window')$clearElement()
remD$findElement(using = 'class', 'box_window')$sendKeysToElement(list("이현열 퀀트", key = "enter"))
```

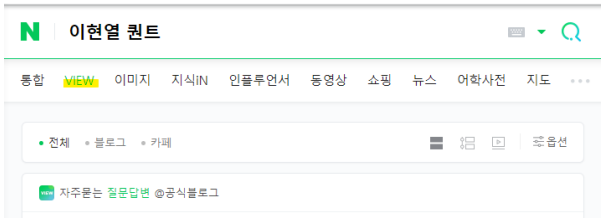
1. 검색어 박스에 접근한 후, `clearElement()`를 실행하면 모든 텍스트가 지워진다.
2. `sendKeysToElement('이현열 퀀트')`를 실행하여 새로운 검색어를 입력한 후, key = "enter"를 추가하면 엔터 동작이 실행된다.

이번에는 [VIEW] 버튼을 클릭하는 동작을 실행해보도록 한다. 해당 부분의 XPATH 위치는 다음과 같다.

```
//*[@id="lnb"]/div[1]/div/ul/li[2]/a
```

이를 이용해 해당 부분을 클릭하는 동작을 실행해보자.

```
remDr$findElement(using = 'xpath', '//*[@id="lnb"]/div[1]/div/ul/li[2]/a')$clickElement()
```



이제 page down 기능을 수행해보도록 하자.

```
remDr$executeScript("window.scrollTo(0,document.body.scrollHeight);")
```

`document.body.scrollHeight`는 웹페이지의 높이를 나타내는 것으로써, `window.scrollTo(0, document.body.scrollHeight);`는 웹페이지의 가장 하단까지 스크롤을 하라는 자바스크립트 명령어다.

그러나 결과를 살펴보면 스크롤이 끝까지 내려가며 얼마간의 로딩이 있은 후 새로운 데이터가 생성된다. 이처럼 유튜브나 인스타그램, 페이스북 등 많은 검색결과를 보여줘야 하는 경우 웹페이지 상에서 한 번에 모든 데이터를 보여주기 보다는 스크롤을 가장 아래롤 위치하면 로딩을 거쳐 추가적인 결과를 보여준다. 따라서 스크롤을 한 번만 내리는 것이 아닌 모든 결과가 나올 때까지 내리는 동작을 실행할 필요가 있다.

```
prev_height = remDr$executeScript('return document.body.scrollHeight')

while (TRUE) {
  remDr$executeScript('window.scrollTo(0, document.body.scrollHeight);')
  Sys.sleep(2)

  curr_height = remDr$executeScript('return document.body.scrollHeight')
  if (unlist(curr_height) == unlist(prev_height)) {
    break
  }
  prev_height = curr_height

}
```

1. return document.body.scrollHeight은 현재의 창 높이는 반환하는 자바스크립트 명령어이며, 이를 prev_height에 저장한다.
2. while 구문을 통해 반복문을 실행한다.
3. 셀레니움을 통해 페이지의 최하단으로 스크롤을 내린다.
4. 페이지가 로딩되는 시간을 기다리기 위해 2초간 슬립을 준다.
5. curr_height에 현재 창 높이를 저장한다.
6. curr_height와 prev_height가 동일하다는 의미는 페이지가 끝까지 내려왔다는 의미이다. 따라서 이 경우 break를 통해 while 구문을 멈추며, 그렇지 않을 경우 다시 스크롤을 내리는 동작을 반복한다.

7. prev_height에 새로운 창 높이를 입력한다.

이제 모든 검색 결과가 나타났으면 일반적인 크롤링을 통해 데이터 수집이 가능하다. 제목 부분을 확인해보면 api_txt_lines total_tit _cross_trigger 클래스에 위치하고 있으며, 이를 통해 모든 제목을 크롤링해보자.

```
library(httr)
library(rvest)

html = read_html(unlist(remDr$getPageSource()))
txt_list = html %>%
  html_nodes('.api_txt_lines.total_tit._cross_trigger') %>%
  html_text()

txt_list %>% head()
```

```
[1] "퀀트투자 시작한다면? 천영록 이현열님의 신간 - 경제도서 22년04"
[2] "[책]R을 이용한 퀀트 투자 포트폴리오 만들기/이현열 - R을 잘 몰라서 휘리릭 읽은 책"
[3] "이현열의 퀀트투자(1)"
[4] "[19-14] R을 이용한 퀀트투자 포트폴리오 만들기 (이현열 저)"
[5] "R을 이용한 퀀트 투자 포트폴리오 만들기 -이현열"
[6] "R을 이용한 퀀트 투자 포트폴리오 만들기 (개정판) - 이현열"
```

1. `remDr$getPageSource()` 를 통해 HTML 정보를 확인하며 `unlist()` 를 통해 리스트 형태를 풀어준다. 그 후 `read_html()` 함수를 통해 HTML를 불러온다.
2. `html_nodes()` 함수를 통해 태그 정보를 추출하며, `html_text()` 함수를 통해 텍스트 부분만 추출한다.

이처럼 동적 페이지의 경우도 셀레니움을 통해 웹페이지를 제어한 후 원하는 부분을 추출하면 얼마든지 크롤링이 가능하다.

```
remDr$close()
```

작업이 끝났으면 열려있는 페이지를 종료한다.

```
system("taskkill /im java.exe /f")
```

향후 포트가 꼬이지 않게 위 명령어를 통해 포트를 비워주어도 된다.

네이버 로그인 및 업무 자동화

셀레니움을 활용하면 각종 자동화 업무를 할 수도 있다.

- 좋아요 누르기
- 댓글 달기
- 메일 보내기
- 블로그 글쓰기
- 카페 글쓰기
- 웹 크롤링
- 자동 예약

네이버에서 자동으로 로그인을 한 후, 이메일 전송까지 해보도록 하자.

```
library(wdman)
library(binman)
library(RSelenium)
library(webdriver)
library(clipr)
library(KeyboardSimulator)

chrome(port = 4567L, version = '109.0.5414.25')
rs_driver_object = rsDriver(browser = 'chrome',
                            chromever = '109.0.5414.25',
                            port = 4567L)
remDr = rs_driver_object$client
remDr$open()
```

먼저 관련 패키지 및 셀레니움을 구동한다.

```
url = 'https://nid.naver.com/nidlogin.login?mode=form&url=https%3A%2F%2Fwww.naver.com'
remDr$navigate(url)
# remDr$maxWindowSize()
```

네이버 로그인에 해당하는 사이트를 연다.

```
id = remDr$findElement(using = 'xpath', value = '//*[@id="id"]')
id$clickElement()
id$setElementAttribute('value', 'YOUR ID')
Sys.sleep(2)
```

ID 부분에 해당하는 곳에 접근한 후, `setElementAttribute`를 통해 ID를 입력한다. 만일 `sendKeysToElement` 로 값을 입력할 경우 네이버에서 로봇으로 인식해 접근이 차단된다. 입력 후 2초간 슬립을 준다.

```
pw = remDr$findElement(using = 'xpath', '//*[@id="pw"]')
pw$clickElement()
pw$setElementAttribute('value', 'YOUR PASSWORD')
Sys.sleep(2)
```

비밀번호 부분도 동일하게 입력한다.

```
log_btn = remDr$findElement(using = 'xpath', value = '//*[@id="log.login"]')
log_btn$clickElement()
Sys.sleep(2)
```

로그인 버튼을 클릭한다.

```
remDr$findElement(using = 'xpath', '//*[@id="NM_FAVORITE"]/div[1]/ul[1]/li[1]/a')$clickElement()
Sys.sleep(2)

remDr$findElement(using = 'xpath', '//*[@id="root"]/div/nav/div/div[1]/div[2]/a[1]')$clickElement()
Sys.sleep(2)
```

홈페이지 메인에서 [메일] 부분을 클릭한다. 그 후 [메일 쓰기] 부분을 클릭한다.

```
remDr$findElement(using = 'xpath', '//*[@id="user_input_1"]')$sendKeysToElement(list("leebisu@gmail.com"))
Sys.sleep(2)
remDr$findElement(using = 'xpath', '//*[@id="subject_title"]')$sendKeysToElement(list("테스트 메일입니다."))
Sys.sleep(2)
```

[받는 사람]과 [제목] 부분을 각각 찾아 원하는 텍스트를 입력한다.

```
iframe = remDr$findElement(using = 'xpath', '//*[@id="content"]/div[3]/div/div[2]/div/div[3]/iframe')
remDr$switchToFrame(iframe)

remDr$findElement(using = 'xpath', '/html/body/div/div[1]')$sendKeysToElement(list("셀레니움 테스트 중입니다."))
remDr$switchToFrame(NA)
Sys.sleep(2)
```

내용을 쓰는 부분은 바로 xpath나 css selector를 이용해 접근할 수 없다. 이곳은 iframe을 통해 생성된 내부 프레임이기 때문이다. 따라서 해당 부분에 먼저 접근해줘야 한다.

개발자도구 화면에서 iframe이 위치하는 곳은 찾아 `findElement`에 입력한 후, `switchToFrame`을 통해 해당 부분에 접근한다. 그 후 내용에 해당하는 부분을 찾아 텍스트를 입력하고, `switchToFrame(NA)`를 통해 iframe을 빠져나온다.

```
remDr$findElement(using = 'xpath', '//*[@id="content"]/div[2]/div[1]/div/button[1]')$clickElement()
Sys.sleep(2)
remDr$close()
```

마지막으로 [보내기] 버튼을 누르고 창을 종료한다.