Web Scraping com RSelenium

Lucas Falcão Silva (lucas.falcao@ufabc.edu.br)

Utilidade

- Muitas vezes os dados não são exportáveis nos sites (i.e., não há opção de download).
- O *web scraping* nada mais é do que a extração dos dados diretamente das páginas da web.
- Por meio do *scraping*, toda a internet torna-se uma base de dados em potencial.
- Por exemplo, em alguns sites de instituições federais e estaduais nem por meio de requisição pela Lei de Acesso de Informação é disponibilizado dados públicos para download, mesmo estando disponíveis para consulta. Exemplos:
  - Procedimentos do MP-SP.
  - Medicamentos registrados na ANVISA.
- Logo, o web scraping nada mais é do que a coleta dessas informações na página.

Direferentes sites, diferentes necessidades

- Há diversos pacotes para realizar este procedimento no R: *rvest*, *scrapeR*, *Rcrawler*, entre outros.
- Entretanto, estes pacotes conseguem apenas extrair elementos não dinâmicos. Ou seja, elementos que, por exemplo, não utilizam *JavaScript*.
  - Exemplo: Na consulta de registros de medicamentos no site da Anvisa podemos perceber que o link não se altera quando mudamos para próxima página no sistema de consulta.

  - Ou seja, há um elemento dinâmico que altera os valores dos elementos a partir da interação do usuário com o site.

- Como proceder? **RSelenium** seria o pacote mais indicado para extrair as informações desejadas neste caso.
- Com a finalidade geral de ser um driver de navegador, i.e., um robô programável para navegação da web, torna-se a ferramenta ideal para simular um usuário.
- Devido à complexidade deste tipo de pacote, e também a necessidade de incorporar pacotes mais tradicionais (estáticos) de web scraper, abordaremos um exemplo prática com o RSelenium.

RSelenium noções básicas e pacotes complementares

- Instalando o RSelenium:

```
install.packages("RSelenium")
```

- RSelenium funciona relacionando os comandos dados no R com um navegador externo por meio do *Java*.
- Por este motivo precisamos tanto ter uma versão adequada de um navegador compatível com o *RSelenium* assim como o *Java* instalado no dispositivo.
- Navegadores compatíveis: Firefox, Google Chrome, PhantomJS, Internet Explorer.
- Recomendação: *Firefox*. Velocidade similar ao Chrome e funcionalidades que facilitam a identificação do caminho dos elementos.
- Para abrir o navegador a partir do R devemos fazer o seguinte procedimento:

```
# Carregando pacate:
library(RSelenium)

# Criando o "driver"
# Firefox:
driver<-rsDriver(browser = "firefox", port = 4448L) # Muitas vezes
basta estes dois parâmetros
```

```
# Chrome:
#driver<-rsDriver(browser = "chrome", port = 4440L, chromever = "7
8.0.3904.105")

remDr <- driver[["client"]] # Estabelecendo termo ("remDr") para a
cessar as funcionalidades do driver



# Abrir URL:
remDr$navigate("https://consultas.anvisa.gov.br/#/medicamentos/q/?
periodoPublicacaoInicial=1900-01-01&periodoPublicacaoFinal=1975-12
-31")
```

- OBS.: o navegador fecha ao não ser acionado por muito tempo. Caso isto ocorra, executar o seguinte comando:

```
remDr$open()
```

- Deste modo, percebemos que *RSelenium* nada mais é do que um automatizador de uma navegação manual.
- Isto gera um problema central: demora e possíveis carregamentos errados de elementos.
  - Repare como demora para carregar uma nova página no site da Anvisa.
- Para contornar estes problemas devemos pensar o algoritmo de forma análoga a extração de manual das informações.
  - *"Quais as etapas que tomaria para extrair as informações de forma mais rápida e sem erros?"*

Noções básicas de HTML: identificar elementos nos códigos das páginas

- Não é necessário ter grandes conhecimentos sobre HTML, a linguagem qual as páginas da web são feitas.
- Principal competência é reconhecer qual parte do código HTML está o elemento de interesse (como botão, texto, tabela etc.).
- No exemplo desta aula, precisaremos clicar no botão que exibe 50 itens por página.

- Na maioria dos navegadores há a opção de “inspecionar elemento” clicando com o botão direito do mouse sobre o elemento.

$$50$$

- O primeiro termo costuma ser a *tag* e os demais as *classes*.
- Os parâmetros das classes são os *atributos*.
- Podemos copiar o elemento como *CSS path*, que é como um caminho para o elemento.

“html.js.flexbox.flexboxlegacy.canvas.canvastext.webgl.no-touch.geolocation.postmessage.no-websqldatabase.indexeddb.hashchange.history.draganddrop.websockets.rgba.hsla.multiplebgs.backgroundsize.borderimage.borderradius.boxshadow.textshadow.opacity.cssanimations.csscolumns.cssgradients.no-cssreflections.csstransforms.csstransforms3d.csstransitions.fontface.generatedcontent.video.audio.localstorage.sessionstorage.webworkers.applicationcache.svg.inlinesvg.smil.svgclippaths body.ng-scope div.container.ng-scope div.ng-scope form.ng-pristine.ng-valid.ng-scope div.panel.panel-default.table-responsive div#containerTable.table-responsive div.ng-scope.ng-isolate-scope div.ng-scope div.ng-table-pager.ng-scope div.ng-table-counts.btn-group.pull-right.ng-scope button.btn.btn-default.ng-scope span.ng-binding”

- Se pegarmos a parte final do *CSS Path* (ex.: “button.btn.btn-default.ng-scope span.ng-binding”) e pesquisarmos pelo navegador, veremos o elemento como resultado.

Comandos usuais dos algoritmos:

1. Interface do console

- **print**: retorna no console o objeto inserido no parâmetro.
- **paste0**: unifica os parâmetros em um único *string*.

```
x<-1
print(paste0("x é igual a ", x))

## [1] "x é igual a 1"
```

- **beep**: comando que emite um som curto. Necessário instalar o pacote *beepr*.

```
install.packages("beepr")

library("beepr")
beep(2)
```

2. Repetição.

- **For**: loop alterando o valor de um elemento *i* de acordo com uma sequência pré-estabelecida.
  - Utilidade: paginação.

```
for(i in 1:5){
  print(paste0("i igual a: ",i))
}
```

3. Manipular o tempo

- **Sys.sleep**: R não executa o próximo comando por *n* segundos

```
for(i in 1:5){
  print(paste0("esperando ",i," segundos"))
  Sys.sleep(i)
}
```

- **Sys.time**: captura o tempo do sistema.

```
t00<-Sys.time() # Guardando tempo innicial
for(i in 1:5){
  print(paste0("esperando ",i," segundos"))

  Sys.sleep(i)

  t01<-Sys.time()

  s<-difftime(t01,t00,units = "secs") # Comando para calcular a di
ferença do tempo na unidade específica

  print(paste0(s))
}
```

4. Condicionais

- **If**: se uma dada relação é verdadeira, executa o(s) comando(s).
- **Else**: se a dada relação do *if* for falsa, executa o(s) comando(s).

```
for(x in 1:5){
  if((x%%2)==0){ # %% é operador para o resto da divisão
    print(paste0(x," é par"))
  }else{
    print(paste0(x," é ímpar"))
  }
}
```

```
## [1] "1 é ímpar"
## [1] "2 é par"
## [1] "3 é ímpar"
## [1] "4 é par"
## [1] "5 é ímpar"
```

- **while**: repete o(s) comando(s) até que a relação não seja mais verdadeira

```
x<-1

while(x<5){
  print(x)
  x<-x+1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

- **tryCatch**: se o comando apresentar algum problema, retornar um valor previamente definido.

```
x<-tryCatch({"laranja"+"maça"},error = function(e){"não deu certo"
})

print(x)
```

```
## [1] "não deu certo"
```

5. Selecionar elemento

- **remDr$findElements**: seleciona o elemento(s) com base em um determinado padrão (parâmetro = "*using*") e um caminho ()

```
#remDr$open() #Abrir novamento o navegador

#remDr$navigate("https://consultas.anvisa.gov.br/#/medicamentos/q/
?periodoPublicacaoInicial=1900-01-01&periodoPublicacaoFinal=1975-1
2-31")

## Para selecionar o botão de 50 itens por página:
webElems <- remDr$findElements(using = 'css selector', "button.btn
.btn-default.ng-scope span.ng-binding")

length(webElems) ## Selecionou três elementos. Podemos ver quais s
ão pesquisando o css path no navegador.
```

- Para evitar problemas com carregamento incorreto ou incompleto das páginas podemos usar o **tryCatch** em conjunto com o **while**:
  - Caso o elemento não seja encontrado o objeto *webElems* deverá ficar vazio (*NULL*).
  - Enquanto o objeto for vazio, tentar seleciona-lo.

```
## Fechando e abrindo o navegador
remDr$close()
remDr$open()

remDr$navigate("https://consultas.anvisa.gov.br/#/medicamentos/q/?
periodoPublicacaoInicial=1900-01-01&periodoPublicacaoFinal=1975-12
-31")

webElems <- NULL # Criando objeto ou apagando o que tem nele

## Condicional, enquanto não existir nada no webElems, tentar sele
cionar

    while(length(webElems)==0){
      print("Esperando página carregar")
      Sys.sleep(2)

      webElems <- tryCatch({remDr$findElements(using = 'css select
or', "button.btn.btn-default.ng-scope span.ng-binding")},
                           error = function(e){NULL})
    }

## [1] "Esperando página carregar"
## [1] "Esperando página carregar"

    length(webElems)

## [1] 3
```

6. Clicar em um elemento

- **webElems$clickElement()**: clica no elemento *webElems*

```
## Criando novo objeto apenas para o terceiro botão (50 itens):
webElems2<-webElems[[3]]

## Clicando no webElems2
webElems2$clickElement()
```

```
## Apagando elementos
rm(webElems2,webElems)
```

7. Criar data frames temporários

- Data frames temporários são úteis para armazenar a informação de uma página, e posteriormente ser anexado ao data frame principal.

```
df<-data.frame(numeros=numeric()) #criando um data frame com uma v
ariável numérica

for(x in 1:5){
  df0<-data.frame(numeros=numeric())

  df0<-data.frame(numeros=x)

  df<-rbind(df,df0) #Unindo df0 com df

  print(df)

  rm(df0) #apagando df0
}
```

```
##   numeros
## 1       1
##   numeros
## 1       1
## 2       2
##   numeros
## 1       1
## 2       2
## 3       3
##   numeros
## 1       1
## 2       2
## 3       3
## 4       4
##   numeros
## 1       1
## 2       2
## 3       3
## 4       4
## 5       5
```

8. Extrair informações

- Tabelas:

- **htmlParse**: do pacote *XML*, comando lê o código fonte da página
- **readHTMLTable**: se a página for essencialmente uma tabela, transforma a página em tabela

```
#install.packages("XML")

library("XML")

## Lendo a página
doc <- htmlParse(remDr$getPageSource()[[1]])

## Lendo a tabela
table<-readHTMLTable(doc)[[1]]

# Não é perfeito
table

## Limpando a tabela:
# Excluindo primeira coluna e linha
table<-table[-1,-1]

# Renomeando variáveis
colnames(table)<-c("nome","api","registro","processo","empresa","situacao","deferimento","vencimento")

table
```

- Textos:
  - **webElems$getElementText()**: retorno o texto aparente de um elemento

```
remDr$close()
remDr$open()

## remDr$navigate("https://consultas.anvisa.gov.br/#/medicamentos/2599200340370/?periodoPublicacaoInicial=1900-01-01&periodoPublicacaoFinal=1975-12-31")

Sys.sleep(4)
### Objetivo: pegar a classe trapêutica:
## Selecionar elemento
webElems <- NULL
while(length(webElems)==0){
```

```
      print("Esperando página carregar")
      Sys.sleep(0.5)

      webElems <- tryCatch({remDr$findElements(using = 'css select
or', "tbody tr td div.ng-binding.ng-scope")},
                           error = function(e){NULL})
    }
```

```
## [1] "Esperando página carregar"
```

```
    length(webElems)
```

```
## [1] 9
```

```
## Extraindo texto do primeiro elemento selecionado

webElems[[1]]$getElementText()
```

```
## [[1]]
## [1] "MEDICAMENTOS ATIVOS NA SECRECAO GORDUROSA"
```

```
(rm(webElems))
```

```
## NULL
```

```
## Fazendo tudo de uma vez:

remDr$findElements(using = 'css selector', "tbody tr td div.ng-bin
ding.ng-scope")[[1]]$getElementText()
```

```
## [[1]]
## [1] "MEDICAMENTOS ATIVOS NA SECRECAO GORDUROSA"
```

- **html_text**: do pacote *rvest*, mesma função que o do *RSelenium*, mas um pouco mais rápido.

```
#install.packages("rvest")

library("rvest")
```

```
## Loading required package: xml2
```

```
##
## Attaching package: 'rvest'
```

```
## The following object is masked from 'package:XML':
##
##     xml
```

```
## Lendo a página pelo comando do rvest
page<- read_html(remDr$getPageSource()[[1]])

## Retorna textos dos elementos
webElems <-page %>% html_nodes("tbody tr td div.ng-binding.ng-scop
e")
html_text(webElems)[[1]]
```

```
## [1] "MEDICAMENTOS ATIVOS NA SECRECAO GORDUROSA"
```

9. Download

- Extrair o link

  -**webElems$getElementAttribute**: retorna o valor do atributo de um determinada classe

```
## Exemplo: banco de dados do perfil do aluno da UFABC
remDr$navigate("http://propladi.ufabc.edu.br/informacoes/perfil")

## Selecionando elemento:
webElems <- NULL
while(length(webElems)==0){
      print("Esperando página carregar")
      Sys.sleep(0.5)

      webElems <- tryCatch({remDr$findElements(using = 'css select
or', "div.full div#comp div#comp_100 div#comp-i div.item-page p sp
an a")},
                           error = function(e){NULL})
    }
```

```
## [1] "Esperando página carregar"
```

```
    length(webElems)
```

```
## [1] 7
```

```
## Extraindo atributo:
webElems[[1]]$getElementAttribute("href") #Só funciona com element
o único
```

```
## [[1]]
## [1] "http://propladi.ufabc.edu.br/images/perfil_graduacao/perfi
l_discente_2018.pdf"
```

```
webElems[[2]]$getElementAttribute("href")
```

```
## [[1]]
## [1] "http://propladi.ufabc.edu.br/images/perfil_graduacao/microdados_perfil_discente_2018.csv"
```

-**html_attr**: do *rvest*, mesma coisa

```
page<- read_html(remDr$getPageSource()[[1]])

## Extraindo lista com links
links <- page %>%
  html_nodes("div.full div#comp div#comp_100 div#comp-i div.item-page p span a") %>%
  html_attr("href")

links
```

```
## [1] "/images/perfil_graduacao/perfil_discente_2018.pdf"
## [2] "/images/perfil_graduacao/microdados_perfil_discente_2018.csv"
## [3] "/images/perfil_graduacao/microdados_perfil_discente_2016.csv"
## [4] "/images/perfil_graduacao/microdados_perfil_discente_2015.xls"
## [5] "/images/perfil_graduacao/microdados_perfil_discente_2014.xls"
## [6] "/images/perfil_graduacao/microdados_perfil_discente_2013.xls"
## [7] "/images/perfil_graduacao/microdados_perfil_discente_2012.xls"
```

- Download
  - **download.file**: com base em um link, realiza o download. No caso, este comando não funcionária em sites dinâmicos.

```
## Completando o link:
links<-paste("http://propladi.ufabc.edu.br",trimws(links,"both"),sep = "")

## download
# Local:
setwd("C:\\Users\\oOluc\\Dropbox\\web_scraping")

#Download
for(i in 1:length(links)){
  ## Objeto com o nome do arquivo
  file<-substring(links[i],54,nchar(links[i])) # Pega o string ent
```

```
re os caracteres indicados

  # Download
  download.file(links[i],file, mode="wb")
}
```

Caso prático: registros de medicamentos entre 1900 a 1975

- Site (já é que estávamos usando como exemplo)
- Objetivo:
  - Extrair informações gerais de cada registro (página inicial).
  - Extrair data de registro e classe terapêutica, informações expostas após clicar no registro.
- *"Quais as etapas que tomaria para extrair as informações de forma mais rápida e sem erros?"*
  - Insight: quando clicamos em um registro, é incluído no link o número do processo.
  1. Loop de paginação (condição: se é a última página, parar):
     - Garantir que a página carregou corretamente
     - Extrair tabela da página
     - Adicionar no data frame
     - Seguir para próxima página
  2. Extrair informações da última página
  3. Nova variável com apenas caracteres numéricos do número do processo
  4. Loop usando a nova variável para acessar os links de cada processo
     - Garantir que a página carregou corretamente
     - Extrair registro e classe

- Adicionar ao data frame

Script para a etapa 1 e 2

```
### Etapa 1 e 2 ###

## Limpando o R
rm(list=ls())

#Diretório
setwd("C:\\Users\\oOluc\\Dropbox\\web_scraping")

#Pacotes:
packages<-c("XML","RSelenium","rvest","beepr"); lapply(packages, r
equire, character.only=T)

# Objetivo:
df<-data.frame(
  nome = character(),
  api = character(),
  registro = character(),
  processo = character(),
  empresa = character(),
  situacao = character(),
  deferimento = character(),
  vencimento = character()
  )



# Abrir navegador
driver<-rsDriver(browser = "firefox", port = 4441L) #Os argumentos
podem variar.
remDr <- driver[["client"]]

#remDr$open()

#### Extraindo dados iniciais ####

## Acessando site
remDr$navigate("https://consultas.anvisa.gov.br/#/medicamentos/q/?
periodoPublicacaoInicial=1900-01-01&periodoPublicacaoFinal=1975-12
-31")

## Clicando em 50 itens
webElems<-NULL
```

```r
while(is.null(webElems) | length(webElems)==0){
  webElems <- tryCatch({remDr$findElements(using = 'css selector',
"button.ng-scope")},
                       error = function(e){NULL})
  #LOOP ATÉ A PAGINA CARREGAR

}

webElem2 <- webElems[[3]] # Escolhemos o terceiro botão, para most
rar 50 empresas por página

webElem2$clickElement() # clica no link

rm(webElems,webElem2) # apagando elementos

## Certificando que a pagina carregou
# Lógica: se não há 50 elementos, esperar 3 segundos
page<- read_html(remDr$getPageSource()[[1]])
webElems <-page %>% html_nodes("table.table.table-hover.table-stri
ped.ng-scope.ng-table tbody tr.ng-scope")
while(length(webElems)<50){
  print("Esperando página carregar")
  Sys.sleep(3)

  page<- read_html(remDr$getPageSource()[[1]])
  webElems <-page %>% html_nodes("table.table.table-hover.table-st
riped.ng-scope.ng-table tbody tr.ng-scope")
}

rm(page,webElems)
### Loop de paginação
# Como a quantidade total de páginas não está disponível, é melhor
usar o comando while
# Necessário achar alguma característica particular na última pági
na
# No caso, o botão de "próximo" fica bloqueado, mudando o código f
onte
  # Este botão é o único com o seguinte código: "ul.pagination.ng-
table-pagination.ng-scope li.ng-scope.disabled a.ng-scope"

## Elemento existente apenas na ultima pagina:
elems_ultima_pg <- NULL

## Loop
```

```r
# Marcando a página
k<-1
while(is.null(elems_ultima_pg) | length(elems_ultima_pg)==0){
  # Marcando tempo
  t00<-Sys.time() #Tempo inicial

  # Reportando
  print(paste0("Página ",k))

  # Extraindo tabela
  doc <- htmlParse(remDr$getPageSource()[[1]]) # Lendo página

  table <- readHTMLTable(doc)[[1]] #Lendo tabela
  rm(doc)

  # Confirmando se está ok: table está vazia?
  if(nrow(table)<2 |is.null(table)){
    beep(11);Sys.sleep(0.5);beep(11);Sys.sleep(0.5);beep(11);Sys.s
leep(0.5);beep(11);Sys.sleep(0.5)
  }

  # Excluindo primeira coluna e linha
  table<-table[-1,-1]

  # Renomeando variáveis
  colnames(table)<-c("nome","api","registro","processo","empresa",
"situacao","deferimento","vencimento")

  # Adicionando ao df
  df<-rbind(df,table)

  ## Próxima página

  # Selecionando botão de próxima página
  # Sempre é o último "ul.pagination.ng-table-pagination.ng-scope
li.ng-scope a.ng-scope"
  webElems<-NULL
  while(is.null(webElems) | length(webElems)==0){
    webElems <- tryCatch({remDr$findElements(using = 'css selector
', "ul.pagination.ng-table-pagination.ng-scope li.ng-scope a.ng-sc
ope")},
                         error = function(e){NULL})
    #LOOP ATÉ A PAGINA CARREGAR
```

```
  }

  webElem2 <- webElems[[length(webElems)]] # Escolhemos o sexto bo
tão

  webElem2$clickElement()

  rm(webElems,webElem2) # apagando elementos

  ## Certificando que a página carregou
  # Lógica: extraindo nova tabela, se for a igual aanterior, não c
arregou
  table0<-table

  while(isTRUE(all.equal(table,table0))){
    print("Esperando nova página carregar")
    Sys.sleep(1.5)

    doc <- htmlParse(remDr$getPageSource()[[1]]) # Lendo página

    table0 <- readHTMLTable(doc)[[1]] #Lendo tabela
    rm(doc)

    # Excluindo primeira coluna e linha
    table0<-table0[-1,-1]
    colnames(table0)<-c("nome","api","registro","processo","empres
a","situacao","deferimento","vencimento")

  }

  ## Objeto existente apenas na última página
  elems_ultima_pg <- tryCatch({remDr$findElements(using = 'css sel
ector', "ul.pagination.ng-table-pagination.ng-scope li.ng-scope.di
sabled a.ng-scope")},
                              error = function(e){NULL})
  ## Marcador de página
  k<-k+1

  ## Tempo final
  t01<-Sys.time()
  s<-difftime(t01,t00,units = "secs")

  #Reportando
  print(paste0("Tempo gasto: ",s))
```

```
  beep(2)
}#paginacao

## Última página:
# Reportando
print(paste0("Página ",k," (última página)"))

# Extraindo tabela
doc <- htmlParse(remDr$getPageSource()[[1]]) # Lendo página

table <- readHTMLTable(doc)[[1]] #Lendo tabela
rm(doc)

# Confirmando se está ok: table está vazia?
if(nrow(table)<2 |is.null(table)){
  beep(11);Sys.sleep(0.5);beep(11);Sys.sleep(0.5);beep(11);Sys.sle
ep(0.5);beep(11);Sys.sleep(0.5)
}

# Excluindo primeira coluna e linha
table<-table[-1,-1]

# Renomeando variáveis
colnames(table)<-c("nome","api","registro","processo","empresa","s
ituacao","deferimento","vencimento")

# Adicionando ao df
df<-rbind(df,table)

# Salvando
write.csv(df,"df_etapa1.csv")
beep(4)
```

Script para as etapas 3 e 4

```
## Limpando o R
rm(list=ls())

#Diretório
setwd("C:\\Users\\oOluc\\Dropbox\\web_scraping")

# Pacotes:
packages<-c("XML","RSelenium","rvest","beepr"); lapply(packages, r
equire, character.only=T)
```

```r
# Objetivo:
df<-data.frame(
  nome = character(),
  api = character(),
  registro = character(),
  processo = character(),
  empresa = character(),
  situacao = character(),
  deferimento = character(),
  vencimento = character(),
  n_processo = character(),
  date_reg = character(),
  classe = character()
)

#### Etapa 3: Crianco variável apenas com os número do processo ##
##
# Abrindo df da etapa1
df0<-read.csv("df_etapa1.csv",stringsAsFactors = FALSE)

# Criandonova variável
df0$n_processo<-gsub("[^0-9]","",df0$processo)

# Excluindo variável inútil
df0$X<-NULL

#### Etapa 2: extraindo variáveis

# Abrir navegador
driver<-rsDriver(browser = "firefox", port = 4442L) #Os argumentos
podem variar.
remDr <- driver[["client"]]

## Para fazer o tempo médio:
t<- data.frame(time=as.numeric(s))

## Loop para acessar cada registro
for(x in 1:nrow(df0)){
  ## Tempo
  t00<-Sys.time()

  ## Reportando
  print(paste0("Extraindo dados do registro ",x))
  ## Criando URL do registro x
```

```r
  n_proc<-df0$n_processo[x] #pegando o número do processo do regis
tro X

  # Colocando no link:
  url<-paste0("https://consultas.anvisa.gov.br/#/medicamentos/",n_
proc,"/?periodoPublicacaoInicial=1900-01-01&periodoPublicacaoFinal
=1975-12-31")

  ## Acessando Url
  remDr$navigate(url)

  ## Certificando que carregou:
  webElems <- NULL
  while(length(webElems)==0){
    print("esperando página carregar")
    Sys.sleep(1.5)
    webElems <- tryCatch({remDr$findElements(using = 'css selector
', "a.btn.btn-default.no-print.ng-scope")},
                         error = function(e){NULL})
  }

  ## Extraindo informações
  df1<-data.frame(
    nome = df0$nome[x],
    api = df0$api[x],
    registro = df0$registro[x],
    processo = df0$processo[x],
    empresa = df0$empresa[x],
    situacao = df0$situacao[x],
    deferimento = df0$deferimento[x],
    vencimento = df0$deferimento[x],
    n_processo = df0$n_processo[x],
    date_reg = as.character(tryCatch({(remDr$findElements(using =
'css selector', "div.ng-scope form.ng-pristine.ng-valid.ng-scope d
iv.panel.panel-default div.table-responsive table.table.table-bord
ered.table-static tbody tr td.ng-binding")[[6]])$getElementText()}
,
                         error = function(e){NA})[[1]]),
    classe = as.character(tryCatch({(remDr$findElements(using = 'c
ss selector', "form.ng-pristine.ng-valid.ng-scope div.panel.panel-
default div.table-responsive table.table.table-bordered.table-stat
ic tbody tr td")[[12]])$getElementText()},
                        error = function(e){NA})[[1]])
  )
```

```r
  # Unindo com data frame:
  df<-rbind(df,df1)

  ## Salvando a cada 50
  if((x%%50)==0){
    Sys.sleep(1)
    save.image(paste0("df_etapa2_1a",x,".RData"))
    beep(2)
  }

  ## Tempo
  t01<-Sys.time() #tempo final
  s<-difftime(t01,t00,units = "secs")
  t<-rbind(t,
           data.frame(
             time=as.numeric(s)
           ))
  print(paste0("Processo ",x," em (s) ",s))
  tempo<-((length(df0$n_processo)-(x+1))*(mean(t$time))/60)
  print(paste0("Minutos até o fim: ",tempo))

  beep(1)
}

write.csv(df,"df_etapa2.csv")

beep(4)
```