# Основы программирования в R

Парсинг HTML-страниц: часть 1

*Алла Тамбовцева, НИУ ВШЭ*

## Содержание



*Предполагается, что студенты уже знакомы с устройством HTML-страниц благодаря обязательному курсу по Python.*

### Поиск по HTML-странице

Для того, чтобы работать с исходным кодом HTML-страниц, нам понадобится библиотека `rvest`. Эту библиотеку можно считать аналогом `BeautifulSoup` в Python, она тоже преобразует строку с исходным кодом HTML в объект, который удобно использовать для поиска по тэгам, классам и атрибутам тэгов.

Библиотека `rvest` является частью уже знакомого нам набора библиотек `tidyverse`, поэтому устанавливать ее отдельно не нужно. Но если что-то пошло не так, можно установить ее отдельно через `install.packages("rvest")`. Обратимся к этой библиотеке, а заодно и к `tidyverse`, она нам тоже понадобится:

```
library(rvest)
library(tidyverse)
```

Выгрузим информацию о командах по квиддичу Великобритании с сайта. Передадим функции `read_html()` ссылку на страницу, с которой мы будем работать:

```
page <- read_html("https://www.quidditchuk.org/clubs/")
page
```

```
## {xml_document}
## <html lang="en-GB" class="no-js no-svg">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="page-template page-template-page-club_index page-template-pa ...
```

Объект `page` имеет тип `xml_document`. По нему можно выполнять поиск по тэгам, классам, атрибутам или по запросам *xpath* (о них поговорим позже). Например, найдем все части кода с тэгом `div`:

```
# head - первые несколько

page %>% html_nodes("div") %>% head
```

```
## {xml_nodeset (6)}
## [1] <div id="page" class="site">\n\t<a class="skip-link screen-reader-text" h ...
## [2] <div class="navigation-top">\n\t\t\t\t\t<div class="mobile-logo">\n\t<a h ...
## [3] <div class="mobile-logo">\n\t<a href="https://www.quidditchuk.org/">\n\t\ ...
## [4] <div class="logo">\n\t\t<a href="https://www.quidditchuk.org/">\n\t\t\t<i ...
## [5] <div class="menu-primary-navigation-container"><ul id="primary-menu" clas ...
## [6] <div class="menu-secondary-menu-top-container"><ul id="top-menu" class="m ...
```

Если мы посмотрим на исходный код страницы, мы заметим, что нужная нам информация по клубам находится в коде с классом `club-info-column`. В стилевом файле CSS класс записывается через точку, учтем это в поиске:

```
page %>%
  html_nodes(".club-info-column")
```

```
## {xml_nodeset (31)}
##  [1] <div class="club-info-column">\n<h2 class="h2-normal"><a href="bangor-br ...
##  [2] <div class="club-info-column">\n<h2 class="h2-normal"><a href="bath-quid ...
##  [3] <div class="club-info-column">\n<h2 class="h2-normal"><a href="bournemou ...
##  [4] <div class="club-info-column">\n<h2 class="h2-normal"><a href="bristol-q ...
##  [5] <div class="club-info-column">\n<h2 class="h2-normal"><a href="chester-c ...
##  [6] <div class="club-info-column">\n<h2 class="h2-normal"><a href="durham-un ...
##  [7] <div class="club-info-column">\n<h2 class="h2-normal"><a href="glasgow-g ...
##  [8] <div class="club-info-column">\n<h2 class="h2-normal"><a href="holyrood- ...
##  [9] <div class="club-info-column">\n<h2 class="h2-normal"><a href="keele-uni ...
## [10] <div class="club-info-column">\n<h2 class="h2-normal"><a href="kent-quid ...
## [11] <div class="club-info-column">\n<h2 class="h2-normal"><a href="leeds-gri ...
## [12] <div class="club-info-column">\n<h2 class="h2-normal"><a href="liverpudd ...
## [13] <div class="club-info-column">\n<h2 class="h2-normal"><a href="london-qu ...
## [14] <div class="club-info-column">\n<h2 class="h2-normal"><a href="the-londo ...
## [15] <div class="club-info-column">\n<h2 class="h2-normal"><a href="loughboro ...
## [16] <div class="club-info-column">\n<h2 class="h2-normal"><a href="mancheste ...
## [17] <div class="club-info-column">\n<h2 class="h2-normal"><a href="nottingha ...
## [18] <div class="club-info-column">\n<h2 class="h2-normal"><a href="olympiann ...
## [19] <div class="club-info-column">\n<h2 class="h2-normal"><a href="oxford-ma ...
## [20] <div class="club-info-column">\n<h2 class="h2-normal"><a href="oxford-un ...
## ...
```

Теперь осталось извлечь текст из каждого отрывка кода с тэгом `club-info-column`. Извлечем его с помощью `html_text()` и сохраним в переменную `info`:

```
info <- page %>%
  html_nodes(".club-info-column") %>%
  html_text()
info[1:5]
```

```
## [1] "Bangor Broken Broomsticks Bangor \nContact:  bangorquidditch@gmail.com \nManager:  Alice McNaugh
## [2] "Bath Quidditch Club Bath Recreation Ground, \n BA2 4DS \nContact:  bath.quidditch@gmail.com \nMa
## [3] "Bournemouth Banshees Quidditch Club Bournemouth \nContact:  subuquidditchsoc@bournemouth.ac.uk \
## [4] "Bristol Quidditch Club Bristol \nContact:  bristol.quidditch@gmail.com \nManager:  Edmund Kitche
## [5] "Chester Centurions Chester \nContact:  1712114@chester.ac.uk \nManager:  Lewis Jones \nWebsite"
```

Объект `info` — это обычный список. Давайте поэтапно извлечем из него разную информацию: почты, имена руководителей, названия клубов, ссылки на страницы клубов и ссылки на их страницы в Facebook.

Адреса почт находятся после слова *Contact:*. Воспользуемся регулярными выражениями и этим фактом для того, чтобы извлечь почты:

```
# str_extract из stringr
# stringr входит в tidyverse

mails <- str_extract(info, "Contact:\\s+.+")
mails[1:5]
```

```
## [1] "Contact:  bangorquidditch@gmail.com "
## [2] "Contact:  bath.quidditch@gmail.com "
```

```
## [3] "Contact:  subuquidditchsoc@bournemouth.ac.uk "
## [4] "Contact:  bristol.quidditch@gmail.com "
## [5] "Contact:  1712114@chester.ac.uk "
```

**Пояснения к регулярному выражению:**

- ищем подстроку `Contact:`;
- `\\s+`: сообщаем, что после нее может быть пробел (а может и не быть), а то и несколько;
- `.+`: сообщаем, что после подстрок с пробелами могут идти любые символы.

Теперь подобным образом найдем имена руководителей (либо *Manager*, либо *President*:

```
heads <- str_extract(info, "(Manager|President):\\s+.+")
heads[1:5]
```

```
## [1] "Manager:  Alice McNaught-Davis " "Manager:  Peter John Stace "
## [3] "Manager:  Jacob Renouf "         "Manager:  Edmund Kitchen "
## [5] "Manager:  Lewis Jones "
```

Как извлечь названия клубов? Заметим, что они находятся перед первым переходом на новую строку:

```
str_extract(info, ".+\\n")
```

Уберем лишний отступ в конце — воспользуемся функцией `str_trim()`, которая удаляет лишние проблелы (по умолчанию только в конце строки, как метод `.rstrip()` в Python):

```
clubs <- str_extract(info, ".+\\n") %>% str_trim()
clubs[1:5]
```

```
## [1] "Bangor Broken Broomsticks Bangor"
## [2] "Bath Quidditch Club Bath Recreation Ground,"
## [3] "Bournemouth Banshees Quidditch Club Bournemouth"
## [4] "Bristol Quidditch Club Bristol"
## [5] "Chester Centurions Chester"
```

Теперь извлечем все ссылки на странице в пределах отрывка кода с классом `club-info-column`:

```
links_all <- page %>%
  html_nodes(".club-info-column") %>%
  html_nodes("a")
links_all[1:5]
```

```
## {xml_nodeset (5)}
## [1] <a href="bangor-broken-broomsticks">Bangor Broken Broomsticks</a>
## [2] <a href="mailto:%20bangorquidditch@gmail.com%20%0A"> bangorquidditch@gmai ...
## [3] <a href="https://www.facebook.com/BangorBB" target="_blank" noreferrer no ...
## [4] <a href="bath-quidditch-club">Bath Quidditch Club</a>
## [5] <a href="mailto:%20bath.quidditch@gmail.com%20%0A"> bath.quidditch@gmail. ...
```

Вектор `links_all` включает все ссылки в рамках разделов о клубах, но их нужно рассортировать: отобрать ссылки на страницы клубов и ссылки на группы в социальных сетях. Если посмотрим внимательно на вектор `links_all`, заметим, что ссылки на страницы клубов находятся на местах с шагом 3: на первом, четвертом, седьмом и так далее. Как выбрать элементы вектора с заданными индексами? Ввести нужные индексы в квадратных скобках:

```
links_all[c(1, 4, 7)]
```

```
## {xml_nodeset (3)}
## [1] <a href="bangor-broken-broomsticks">Bangor Broken Broomsticks</a>
## [2] <a href="bath-quidditch-club">Bath Quidditch Club</a>
## [3] <a href="bournemouth-quidditch-club">Bournemouth Banshees Quidditch Club</a>
```

Теперь сформируем последовательность подобных индексов для всего вектора:

```
seq(1, length(links_all), 3)
```

```
##  [1]  1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73
## [26] 76 79 82 85 88 91
```

Посмотрим на то, что получится:

```
links_all[seq(1, length(links_all), 3)]
```

```
## {xml_nodeset (31)}
##  [1] <a href="bangor-broken-broomsticks">Bangor Broken Broomsticks</a>
##  [2] <a href="bath-quidditch-club">Bath Quidditch Club</a>
##  [3] <a href="bournemouth-quidditch-club">Bournemouth Banshees Quidditch Club ...
##  [4] <a href="bristol-quidditch-club">Bristol Quidditch Club</a>
##  [5] <a href="chester-centurions">Chester Centurions</a>
##  [6] <a href="durham-university-quidditch-club">Durham University Quidditch C ...
##  [7] <a href="glasgow-grim-reapers">Glasgow Grim Reapers</a>
##  [8] <a href="holyrood-hippogriffs">Holyrood Hippogriffs</a>
##  [9] <a href="keele-university-quidditch-club">Keele University Quidditch Soc ...
## [10] <a href="kent-quidditch-club">Kent Quidditch Club</a>
## [11] <a href="leeds-griffins-quidditch-team">Leeds Griffins Quidditch Team</a>
## [12] <a href="liverpuddly-cannons">Liverpool Quidditch Club</a>
## [13] <a href="london-quidditch-club">London Quidditch Club</a>
## [14] <a href="the-london-unspeakables">London Unspeakables</a>
## [15] <a href="loughborough-longshots">Loughborough Longshots</a>
## [16] <a href="manchester-universities-quidditch-club">Manchester Universities ...
## [17] <a href="nottingham-nightmares">Nottingham Nightmares Quidditch Club</a>
## [18] <a href="olympianns-quidditch-club">Olympians Quidditch Club</a>
## [19] <a href="oxford-mammoths">Oxford Mammoths</a>
## [20] <a href="oxford-university-quidditch-club">Oxford Universities Quidditch ...
## ...
```

Теперь из этого извлечем сами ссылки — значения атрибута `href`, которые мы получим с помощью функции `html_attr()`:

```
links_all[seq(1, length(links_all), 3)] %>%
  html_attr("href")
```

Остался последний шаг: все ссылки выше относительные, по ним мы не сможем перейти на страницы, если не доклеим основную часть. Значит, надо доклеить! Воспользуемся функцией `paste()`:

```
club_links <- paste("https://www.quidditchuk.org/clubs/",
      links_all[seq(1, length(links_all), 3)] %>%
  html_attr("href"), sep = "")
```

Аналогичным образом достанем ссылки на страницы в социальных сетях (только они стоят на других местах и доклеивать основную ссылку к ним не нужно):

```
group_links <- links_all[seq(3, length(links_all), 3)] %>%
  html_attr("href")
```

Осталось собрать всю собранную информацию в датафрейм. Склеим все по столбцам с помощью `cbind.data.frame()`:

```
qdat <- cbind.data.frame(clubs, club_links, group_links, mails, heads)
```

Теперь надо собрать описание клубов с их страниц. По-хорошему надо написать функцию, которая

будет принимать на вход ссылку на страницу, а возвращать — текстовое описание клуба. Но прежде чем ее писать, разберемся с одной ссылкой.

```
url1 <- qdat$club_links[1]
page1 <- read_html(as.character(url1))
page1
```

```
## {xml_document}
## <html lang="en-GB" class="no-js no-svg">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="page-template page-template-page-club page-template-page-clu ...
```

На странице клуба описание можно найти по классу `club-description` (для красоты еще уберем пробелы с обеих сторон):

```
text <- page1 %>%
  html_nodes(".club-description") %>%
  html_text() %>%
  str_trim(side = "both")
```

Работает! Теперь напишем функцию:

```
get_desc <- function(url1){
  page1 <- read_html(as.character(url1))
  text <- page1 %>% html_nodes(".club-description") %>%
    html_text() %>% str_trim(side = "both")
  return(text)
}
```

Проверим на другой ссылке:

```
tt <- get_desc(qdat$club_links[6])
tt
```

```
## [1] "DUQC is Durham University's official quidditch club - we are open to both university and communi
```

Тоже работает. Осталось применить функцию ко всем элементам вектора со ссылками. Как сделать это, избегая цикла? Отвлечемся немного и поговорим о функциях `lapply()` и `sapply()`.

## Функции `lapply()` и `sapply()`

Обе эти функции позволяют применить некоторую функцию ко всем элементам вектора или списка, минуя цикл. Разница в том, что функция `lapply()` возвращает результат в виде списка, а `sapply()` — в виде вектора. Создадим список `L` с числовыми векторами и сравним результаты применения функции `sum()` ко всем его элементам:

```
L <- list(c(1, 3, 5),
          c(1, 6),
          c(9, 0))
```

```
lapply(L, sum)
```

```
## [[1]]
## [1] 9
##
## [[2]]
## [1] 7
##
```

```
## [[3]]
## [1] 9
```

```
sapply(L, sum)
```

```
## [1] 9 7 9
```

Действительно, `lapply()` вернул суммы векторов в списке `L` в виде списка, а `sapply()` — те же суммы, но в виде вектора.

Теперь воспользуемся `sapply()` для нашей задачи — применим функцию `get_desc()` к каждому элементу в столбце `club_links` и добавим столбец с результатами в `qdat`:

```
res <- sapply(qdat$club_links, get_desc)
qdat <- qdat %>% mutate(desc = res)
```

Задание выполнено. Осталось обсудить один момент — как быть готовыми к тому, что на странице какая-то информация отсутствует или что ссылка оказалась нерабочей. Поможет ловля исключений.

## Исключения в R

Что будет, если мы попросим R выполнить какую-нибудь заведомо недопустимую операцию, например, сложить две строки?

```
test0 <- "a" + "b"
test0
```

Мы получим ошибку, и эта ошибка приведет к тому, что переменная `test0` создана не будет. Как этого избежать, то есть сделать так, чтобы объект `test0` создавался в любом случае, пусть и без результата исполнения кода? Воспользоваться функцией `try()`:

```
test0 <- try("a" + "b")
```

```
## Error in "a" + "b" : нечисловой аргумент для бинарного оператора
```

```
test0
```

```
## [1] "Error in \"a\" + \"b\" : нечисловой аргумент для бинарного оператора\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in "a" + "b": нечисловой аргумент для бинарного оператора>
```

Эта функция заставляет R исполнить код, указанный в скобках, и в случае, если код корректный, возвращает результат его исполнения, если нет — объект класса `try-error`.

Как воспользоваться функцией `try()` в более сложной ситуации, например, когда мы хотим написать более универсальную функцию, которая не будет ломаться и будет возвращать `NA`, если мы будем приниенять ее некорректно? Очень просто: можно поставить условие на тип результата: если тип `try-error`, сохраняем `NA`, если нет — результат, который задумывался.

Напишем функцию, которая считает сумму элементов вектора и возвращает `NA`, если эту операцию выполнить невозможно:

```
mysum <- function(v){
  res <- try(sum(v))
  if (class(res) == 'try-error'){res <- NA}
  return(res)
}
```

Теперь применим ее к списку `L2` и убедимся, что на последнем векторе этого списка функция `mysum()` не сломается:

```
# текстовый вектор в конце

L2 <- list(c(1, 3, 5), c(1, 6), c("a", "b"))
sapply(L2, mysum)
```

```
## Error in sum(v) : неправильный 'type' (character) аргумента
```

```
## [1]  9  7 NA
```

Всё. Теперь функцию `try()` можно использовать при написании кода для парсинга сайтов, чтобы в случае ошибок (отсутствие информации, неправильная разметка сайта, некорректная ссылка) исполнение кода не останавливалось.