Mac os 用 R+Docker 部署 Rselenium 搭建爬虫

最近在用 R 做爬虫的时候遇到了部分网页无法爬取。于是转投 Selenium 的怀抱。本文就简要介绍利用 mac 下的 docker 搭建 Selenium 环境，实现爬虫应用。
先简单介绍用的工具：

- `Selenium`也是一个用于 Web 应用程序测试的工具，Selenium 测试直接运行在浏览器中，就像真正的用户在操作一样。本文将采用 Selenium Grid，它用于分布式自动化测试，就是一套 Selenium 代码可在不同的环境上运行。Selenium Grid 有两个概念
  - `hub`：主节点，接受测试任务，并进行分配，只有一个主 hub 节点。
  - `node`：分支节点，不同的节点分支，接受 hub 任务，并进行测试，可以有多个。
- `RSelenium`通过调用 Selenium Sever 来模拟浏览器环境，实现爬虫任务。
- `Docker`一般叫 docker 容器，一个可爱的鲸鱼，上面坨着集装箱，可以比喻为电脑集群。先来搞清楚它里面的几个概念。
  - `Docker`镜像，运行 Docker 容器的一个环境，打个比方镜像就像已经配置好的程序包或者源代码的光盘，用于将程勋安装在电脑上。
  - `Docker`容器，简单的说，容器是独立运行的一个或一组应用，以及它们的运行态环境。 如果把镜像看成面向对象中的“类”的话，那么容器就是“类”的实

例化“ 对象”，打个比方：容器就像是已经安装好镜像的一台台电脑，但是容器一关闭后这台电脑就消失了。

    - `Docker`仓库，存放 Docker 镜像的仓库，作用和 GitHub 类似，Docker 仓库就相当于光盘盒子。
      通过 docker 可以快速搭建各种环境。刚好可用于本次爬虫任务的 hub+node 环境的搭建，另外说句题外话 docker 技术细节你可以不懂，但是你需要知道怎么使用。
- `R` 嗯～～～ 不用介绍了吧。
- `Mac OS`……
  接下来进入到实操环节：

下载 Docker 安装 Docker

- Mac 下载安装 Docker desktop 版本还是比较简单，到 Dorker mac 下载地址 注册一个账号，下载 docker dmg 直接安装即可，这也是最简单的方法。

- 安装成功后，打开 docker。

  看到 docker 就在运行了。接下来打开终端工具进行操作会比较方便。

搭建 Selenium 环境

搭建 Selenium 环境主要分为三个步骤：
1 . 启动 docker
打开 docker 程序后，进入终端输入`docker version`即可查看 docker 版本，说明 docker 已经运行。网上有许多

docker 相关的教程，这里就不再赘述。

- 拉取 Selenium 镜像, 我们这里拉取两个镜像 1 个 Selenium hub 镜像，1 个 Selenium chrome-node。Selenium 镜像有许多，更多的请参考 Selenium 镜像列表 。

```
docker pull selenium/hub
docker pull selenium/node-chrome
```

经过一段时间的下载，镜像就下载成功了，下载成功后就可以基于镜像启动容器。

- 基于 Selenium 镜像，启动 Selenium 容器, 启动一个 selenium hub，一个 chrome-node。

```
#单主机
docker run -d -p 4444:4444 --name selenium-hub selenium/hub
docker run -d --link selenium-hub:hub -v /dev/shm:/dev/shm selenium/node-c
hrome
```

启动一个 hub，docker 启动命令如下：docker run -d -p 4444:4444 --name selenium-hub selenium/hub.

做一些简单的说明：

- `run`：通过镜像启动一个容器
- `-p`: 端口映射，5555 是容器宿主机的端口就是我们 docker 这个轮船的端口，4444 是我们容器的端口就是我们集装箱的端口。这说明了我们把容器的 4444 端口开放给 docker 主机的 5555 端口，那么我们就可以通过 docker 主机的 5555 端口来访问容器了，有点啰嗦~~~
- `-d`:docker 后台运行这个容器，我们知道运行 server-standalone-2.52.0.jar 这个包实际上是启动一个 socket

程序的，是在一个 while 循环中的。如果不启用后台运行的话，在 xshell 当前窗口是不能进行其他的操作的，当然你要再开一个窗口连接 docker 也可以。

- `--name`: 指定容器运行的别名，如果不指定会随机生成一个。
- `selenium/hub`：就是我们要运行的镜像文件。

启动完 hub 后，我们启动一个 node, 启动 node 命令如下：docker run -P -d --link selenium_hub:hub selenium/node-chrome 做一些简单的说明：

- `run`：和上文相同
- `-P`：随机生成映射端口号，上文中的 - p 是指定特定的端口号，这里面是 node 我们并不需要知道容器内部的端口号，当然你要指定也可以，端口号不要冲突即可。
- `-d`：后台运行与上文相同。
- `--link`：说明我们这个容器是依赖上文中我们生成的容器 selenium_hub，后面我们会提到 link 的使用。
- `selenium_hub:hub`：前面的 selenium_hub 是我们上文中通过 selenium/hub 镜像启动容器的别名；后面的 hub 一定要写成 hub 或者 HUB，写成其他启动失败，为什么这样我们后面会和 - -link 一起说明。
- `selenium/node-chrome`：node 的镜像。

启动了`selenium/hub`与`selenium/node`后，我们运行`docker ps - a`有如下信息：

这是 selenium 就启动了。

浏览器输入

`localhost:4444`

. 可以查看 Selenium Grid Hub 的版本，也表示其运行正常。

启动 R 安装 Rselenium 包

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

爬取页面

```
library(RSelenium)
remDr<-remoteDriver(port=4444L, browserName = "chrome")
remDr$open(silent = TRUE)#打开浏览器
[1] "Connecting to remote server"
$acceptInsecureCerts
[1] FALSE
$acceptSslCerts
[1] FALSE
$applicationCacheEnabled
[1] FALSE
$browserConnectionEnabled
[1] FALSE
$browserName
[1] "chrome"
$chrome
$chrome$chromedriverVersion
[1] "2.44.609551 (5d576e9a44fe4c5b6a07e568f1ebc753f1214634)"
$chrome$userDataDir
[1] "/tmp/.org.chromium.Chromium.YhY2Yo"
$cssSelectorsEnabled
[1] TRUE
$databaseEnabled
[1] FALSE
$`goog:chromeOptions`
$`goog:chromeOptions`$debuggerAddress
[1] "localhost:35767"
$handlesAlerts
[1] TRUE
$hasTouchScreen
[1] FALSE
```

```
$javascriptEnabled
[1] TRUE
$locationContextEnabled
[1] TRUE
$mobileEmulationEnabled
[1] FALSE
$nativeEvents
[1] TRUE
$networkConnectionEnabled
[1] FALSE
$pageLoadStrategy
[1] "normal"
$platform
[1] "Linux"
$rotatable
[1] FALSE
$setWindowRect
[1] TRUE
$takesHeapSnapshot
[1] TRUE
$takesScreenshot
[1] TRUE
$unexpectedAlertBehaviour
[1] "ignore"
$version
[1] "70.0.3538.110"
$webStorageEnabled
[1] TRUE
$webdriver.remote.sessionid
[1] "d823c1c5d795381ab346daa2a32c7e87"
$id
[1] "d823c1c5d795381ab346daa2a32c7e87"
remDr$navigate("https://www.baidu.com")#爬取百度
remDr$getTitle()#获取title
remDr$screenshot(display = TRUE)#截图
```

结尾

用完后关闭、清除容器、关闭 docker。

```
#清除(关闭全部容器)：
docker kill $(docker ps -a -q)
docker stop $(docker ps -a -q)
#删除全部容器:
docker rm $(docker ps -a -q)
```

参考

RSelenium Docker 安装参考文档

Docker 官方网站

```
sessionInfo()
R version 3.5.1 (2018-07-02)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS  10.14.1

Matrix products: default
BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/A/Framewor
ks/vecLib.framework/Versions/A/libBLAS.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRla
pack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets
[6] methods   base

other attached packages:
[1] RSelenium_1.7.4

loaded via a namespace (and not attached):
 [1] httr_1.3.1        compiler_3.5.1    R6_2.3.0
 [4] assertthat_0.2.0 tools_3.5.1       wdman_0.2.4
 [7] binman_0.1.1      curl_3.2          yaml_2.2.0
[10] Rcpp_1.0.0        jsonlite_1.5      caTools_1.17.1.1
[13] openssl_1.0.2     bitops_1.0-6      semver_0.2.0
[16] XML_3.98-1.16
```