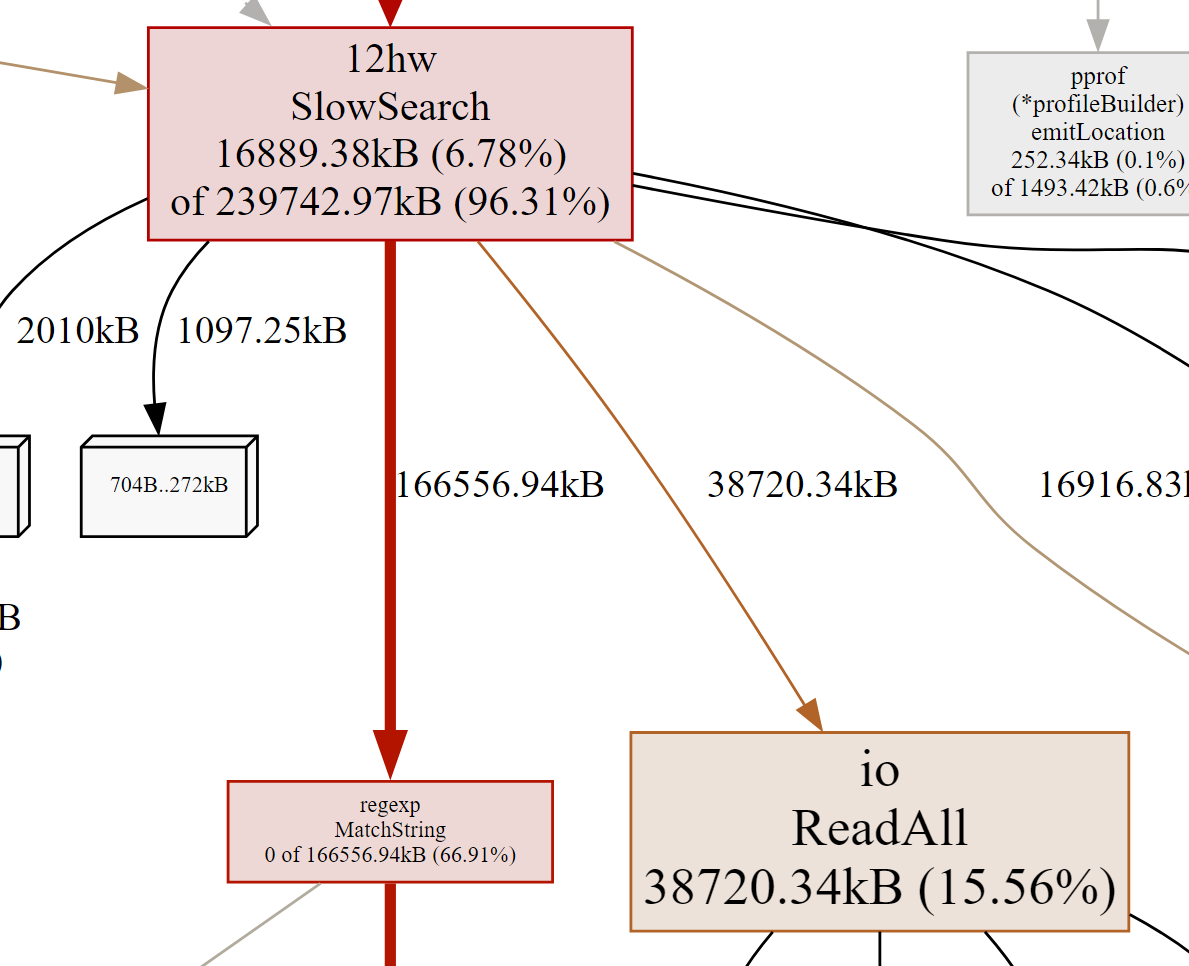
1. Оптимизация считывания файла.

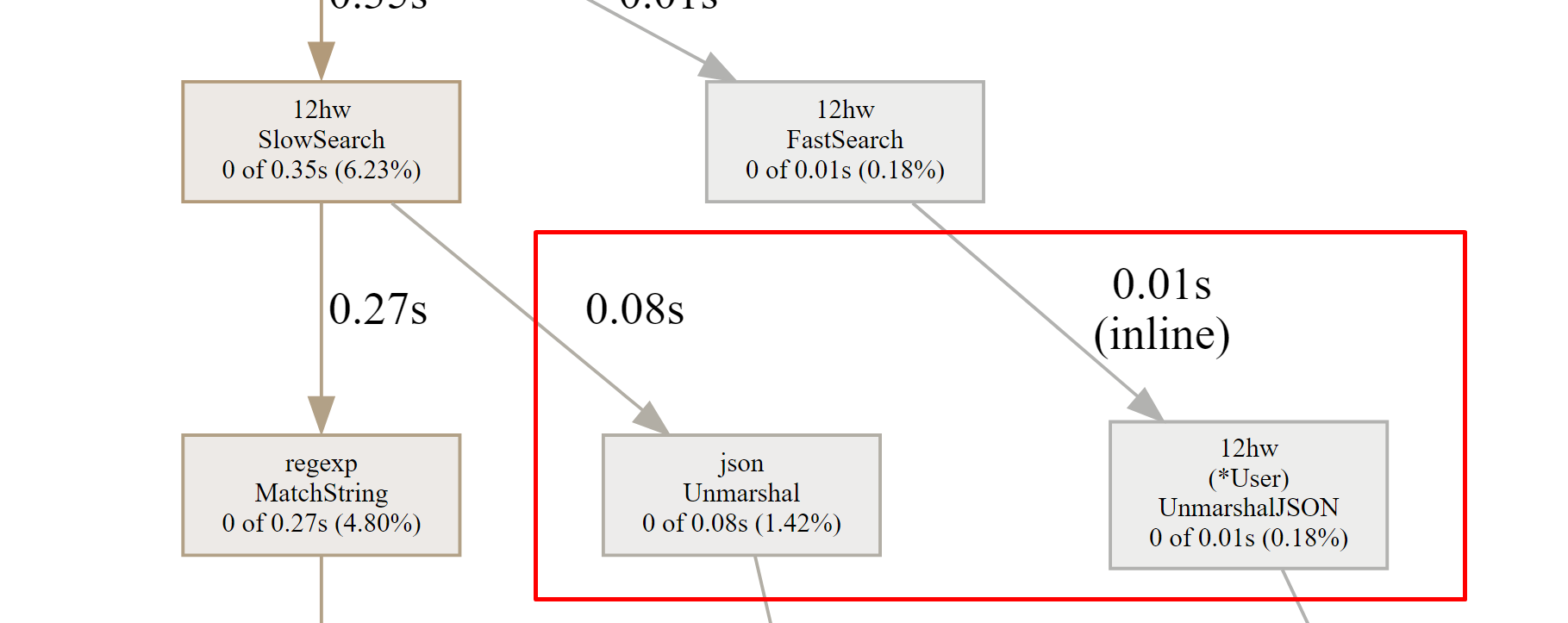
В медленной версии файл сначала считывался, а потом разделялся на строчки. Т.е. выделялась память на хранение всего файла и делалась операция для разбиения файла на строки.



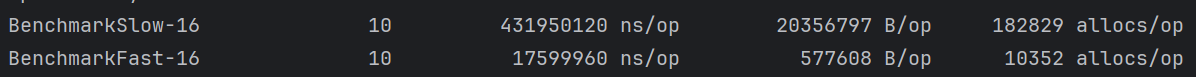
На скрине видно, сколько лишней памяти выделялось. Я же сразу считывал файл построчно, обрабатывая сразу каждую строку.

1. Использование easyjson вместо json.

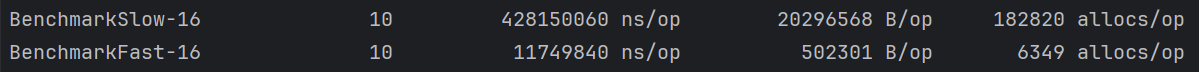
Это ускорило unmarshal. На скрине наглядно видно разницу.



Также я заметил, что не все поля из файла используются (но не сразу заметил). Я оставил только используемые поля, и это очень сильно снизило число аллокаций. Было:



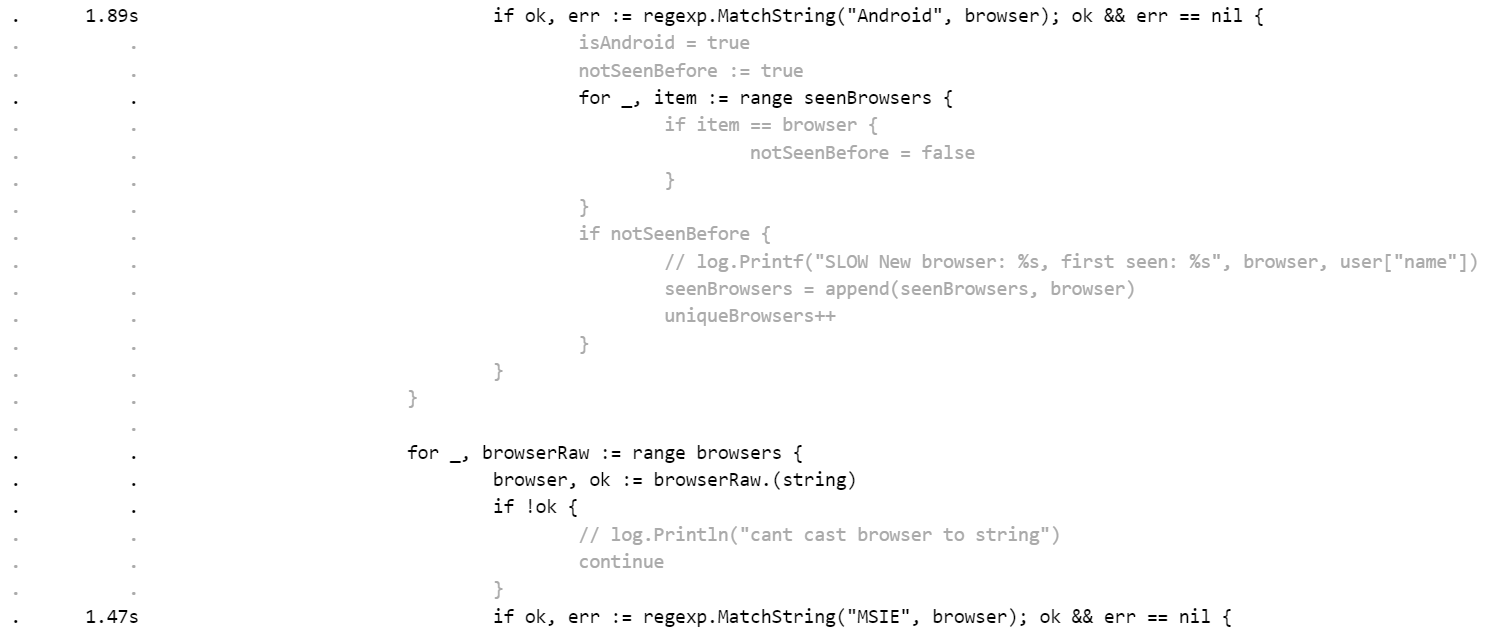
Стало:



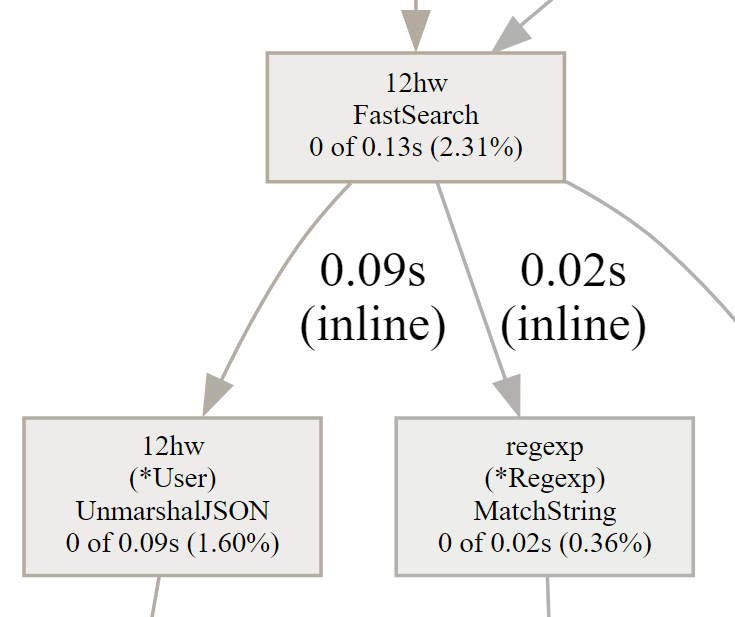
Больше всего уменьшилось число аллокаций, но также и количество памяти, а также время выполнения снизились.

1. Изменение структур переменных, вынос регулярок

Были вынесены из цикла регулярки, чтобы сразу их создать и потом только использовать. До этого на регулярки уходило более 3 секунд:



А стало тратится всего 0,02 секунды:



Была удалена переменная uniqueBrowsers. Она нигде не использовалась.

Был изменён тип seenBrowsers на map быстрого поиска сложностью O(N), а не O(N^2).

Также был удалён users. Из-за того, что я считывал файл построчно, у меня не было необходимости хранить всех юзеров. Он занимал лишние 205 Кбайт:

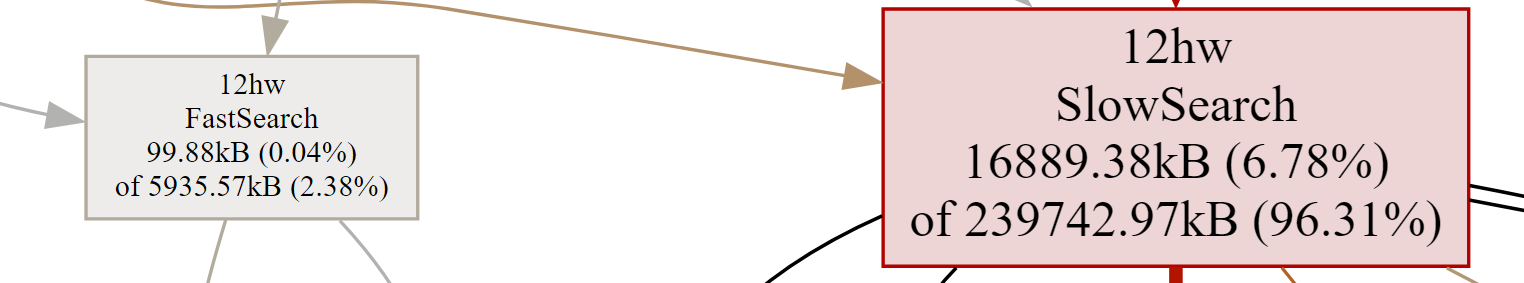


1. Вывод пользователя сразу

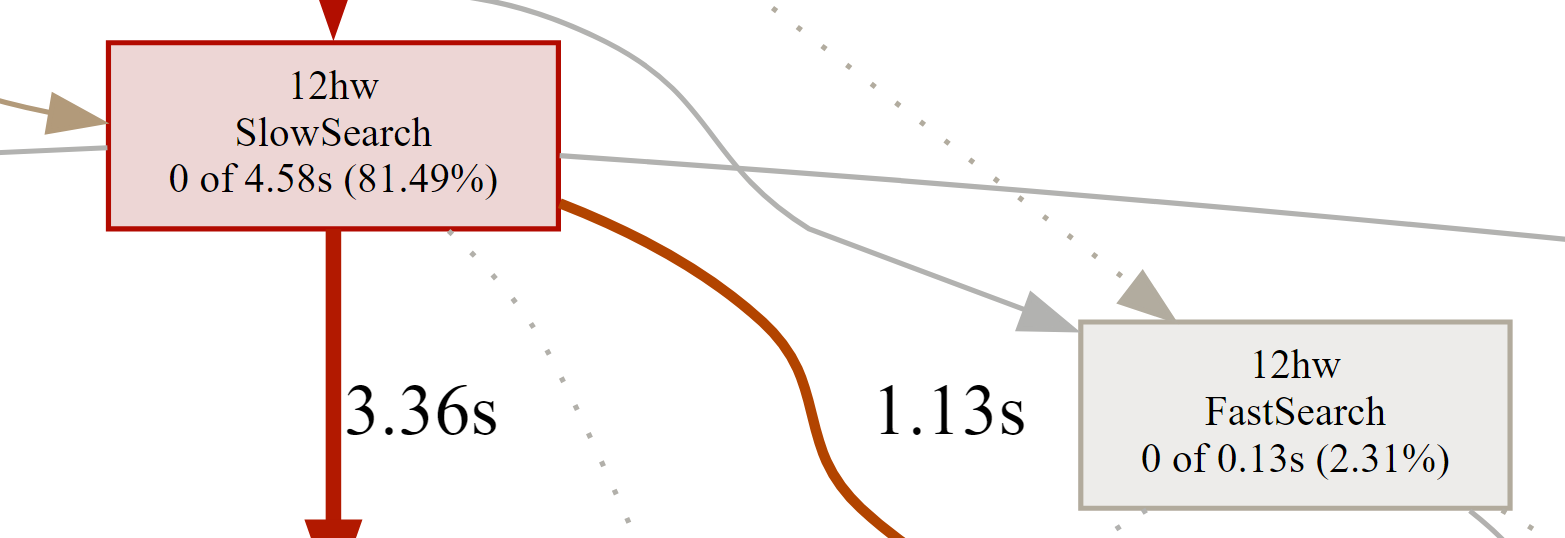
Я также выводил сразу пользователя в out, а не сохранял его в foundUsers, что сэкономило 2,4 Мбайта:  


Также я переписал код в соответствии с изменёнными типами данных. Упростил это: if !(isAndroid && isMSIE) до просто if isAndroid && isMSIE.

В целом получилось следующее:



Быстрая реализация использовала на 234 000 Кбайт меньше.



Также быстрая реализация выполнилась на 3,23 секунды быстрее.