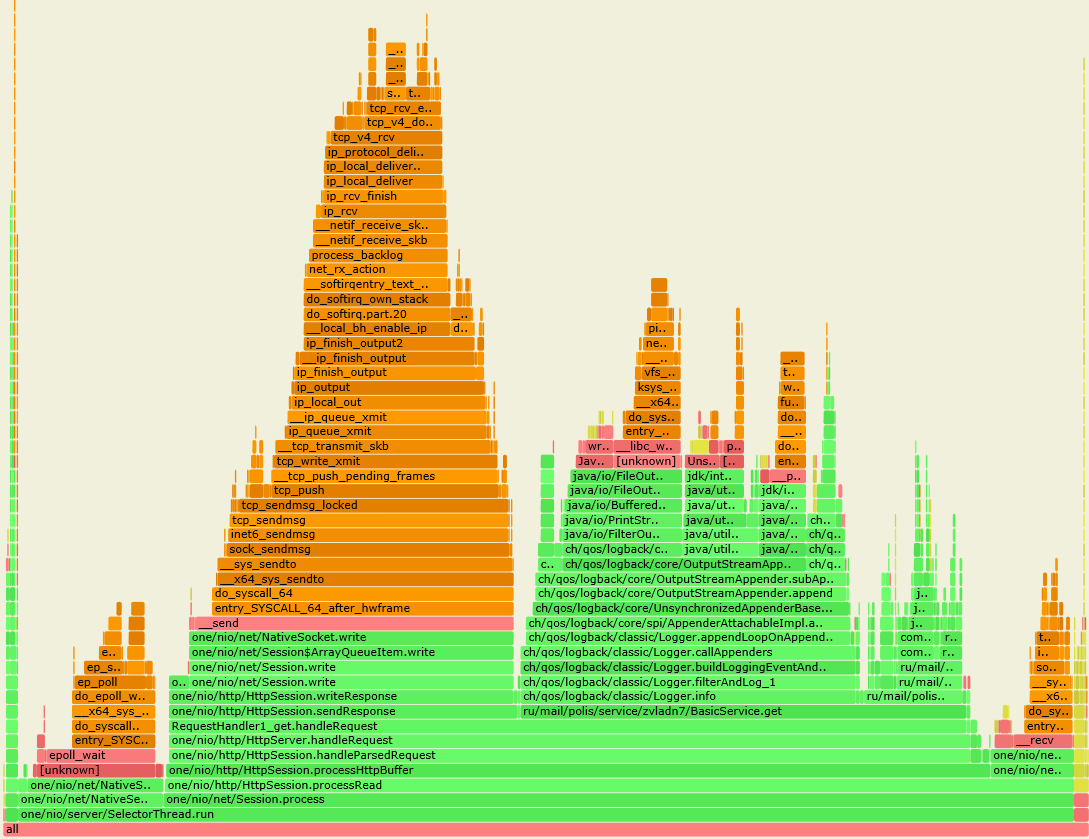
Проведем нагрузочное тестирования с помощью *wrk2.* Будем использовать 1 соединение и 1 поток. Проанализируем результаты профилирования нагрузочного тестирования, которые будут получены с помощью *async-profiler.*

*CPU:*

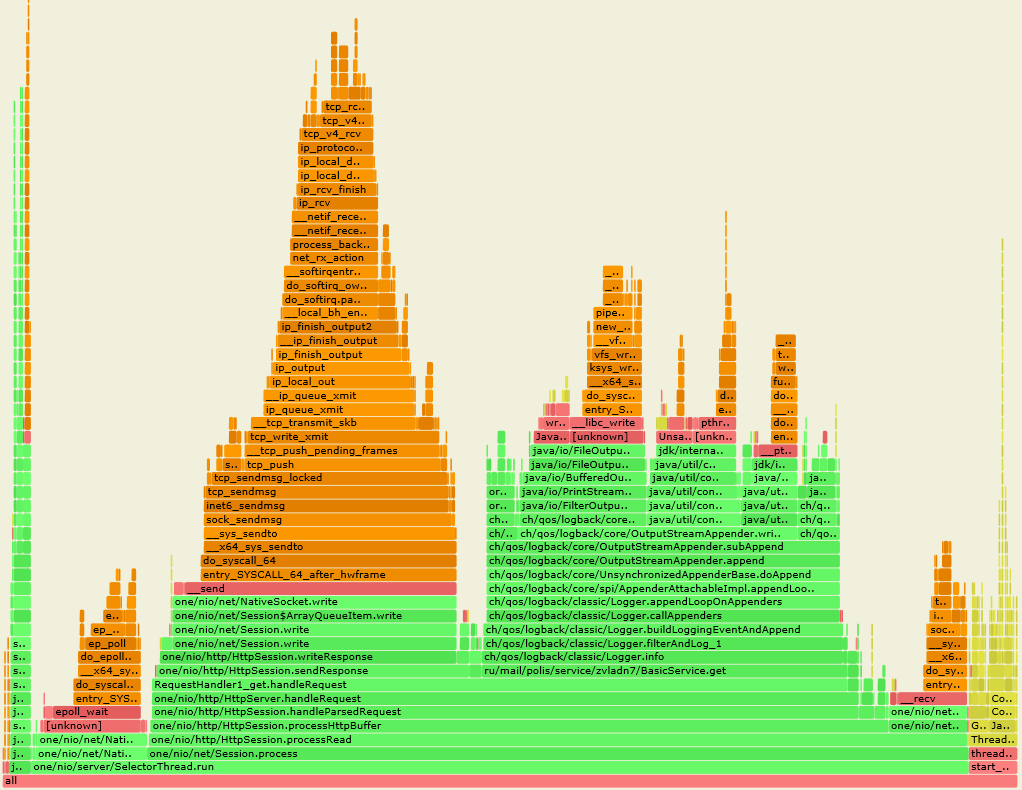
|  |
| --- |
| *wrk -t1 -c1 -d10m -R2000 –latency http://localhost:8080/v0/entity?id=key* |

**

Видим, что под данной нагрузкой наибольшее количество времени занимает передача данных по сети (~ 32%)., а также логирование (~ 30%). Взаимодействие с DAO занимает всего лишь (~ 9%).

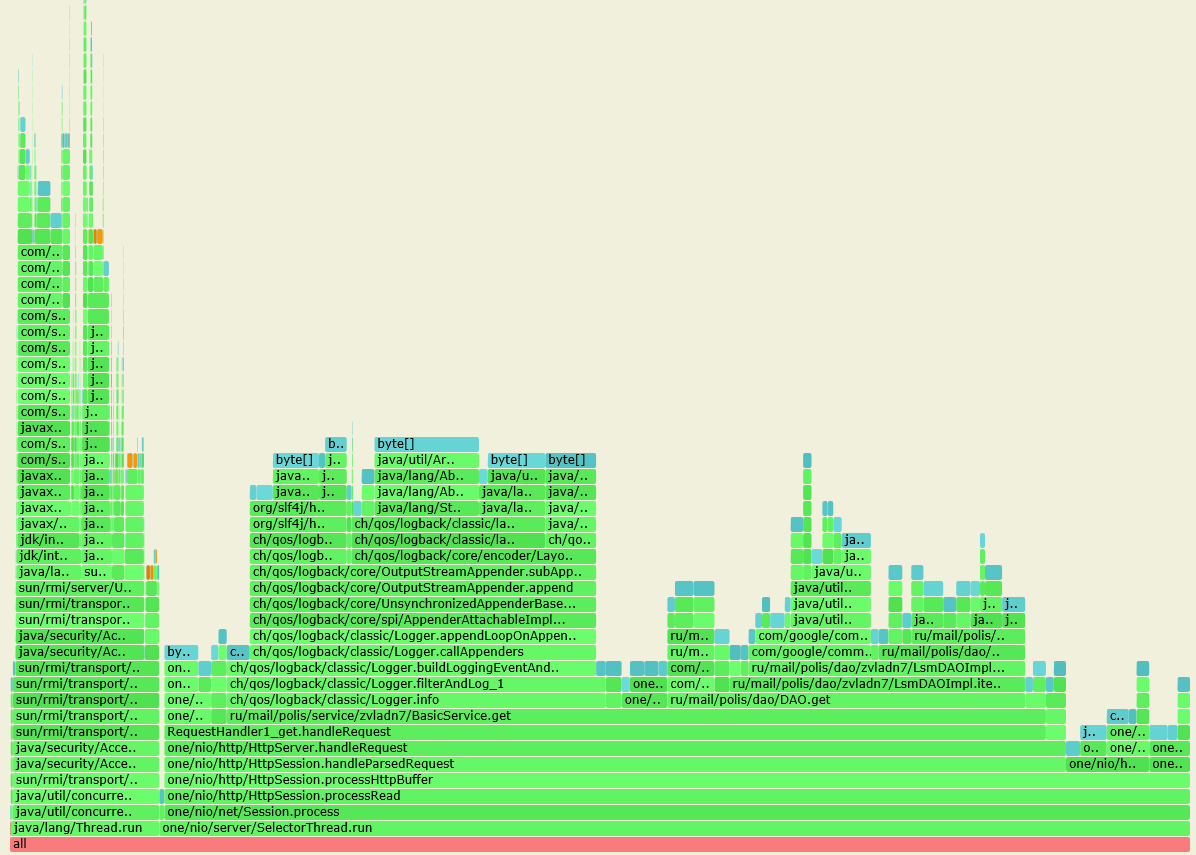
Если добавить кэширование ответов на запросы в виде LinkedHashMap, что поможет поддерживать последовательность поступления запросов, и при этом константный доступ по ключу, то получим следующий результат. (Картинка ниже)

Видим, что теперь время обработки в целом уменьшилось, поскольку время обработки запроса в целом снизилось с 73% до 68%. Получили 5% прирост, поскольку нам не нужно преобразовывать строку в массив байт, оборачивать этот массив в ByteBuffer и делегировать запрос DAO. Мы просто получаем ответ за константу.



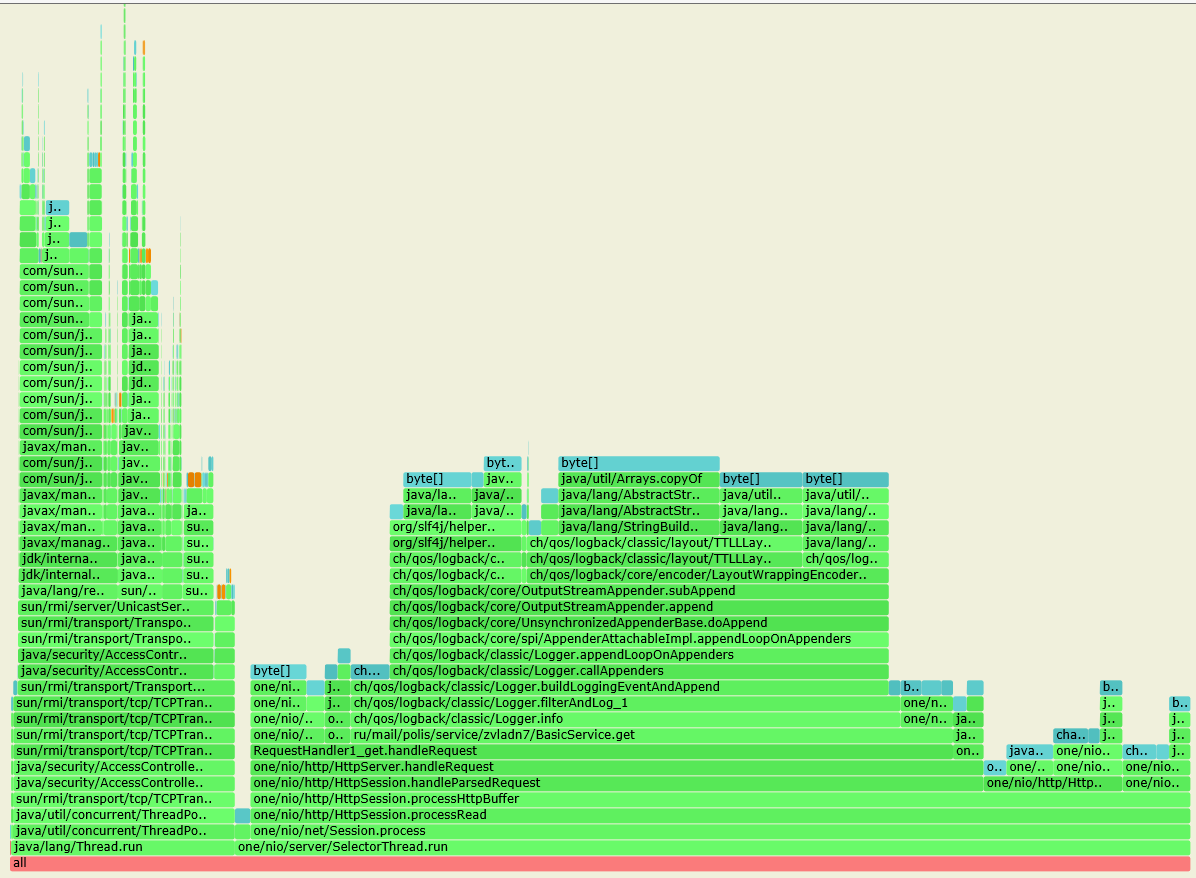
ALLOC:

Теперь проверим, каковы результаты профилирования под такой же нагрузкой, но в режим alloc.



На рисунке видно, что большую часть в куче занимают массивы байт выделенные под логирование запроса, а также на создание нескольких итераторов, массивов байт и еще множество легких объектов при обращении к DAO.

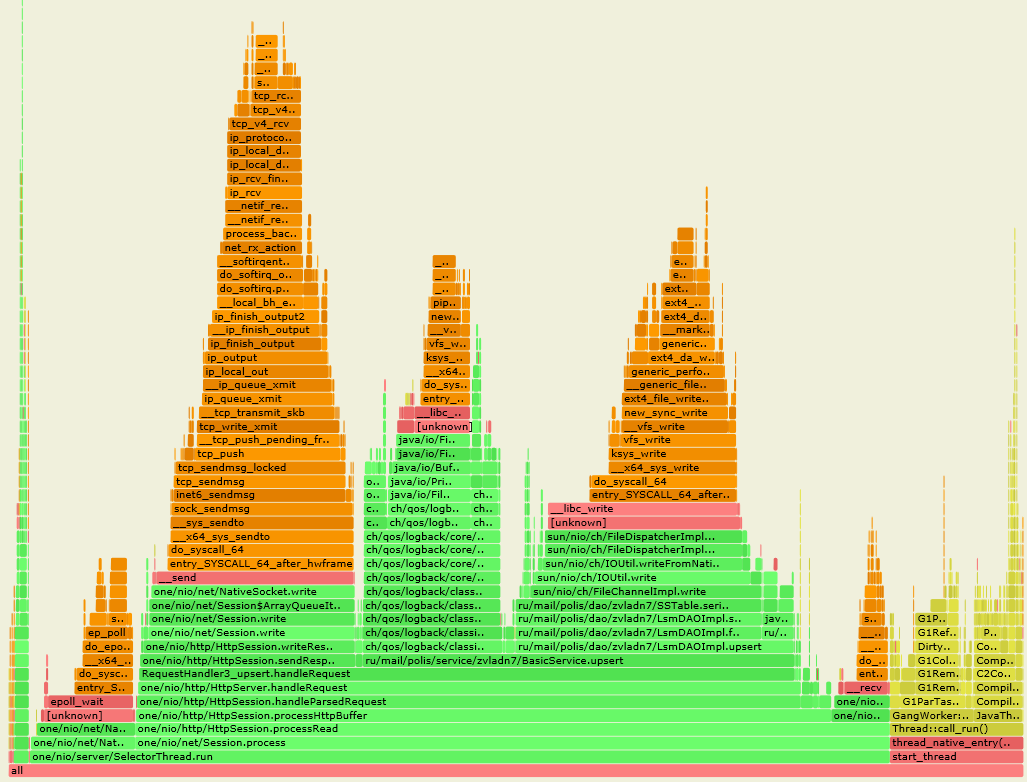
Также проверим, что будет если добавить уровень кэширования, как и в случае профилирования в режиме cpu.

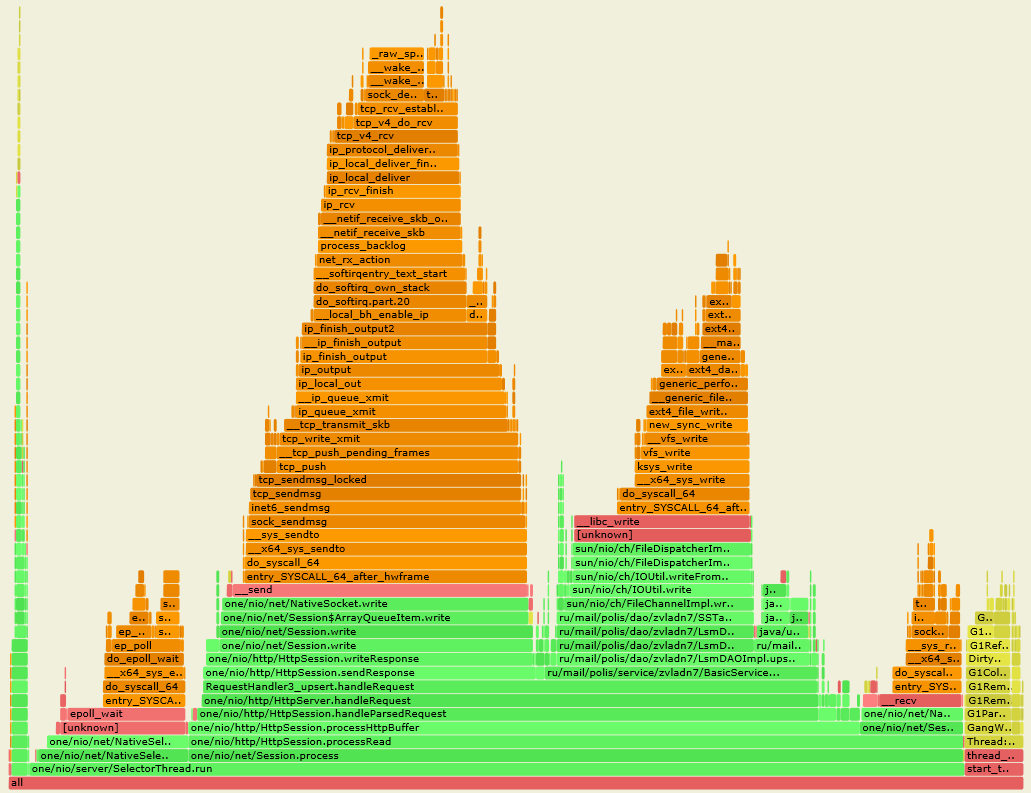


Видно, что размер выделяемой памяти существенно снизился, что доказывает процентное содержании выделения памяти при обработке запросы. А снижении операции употребления памяти вызовов *BasicService.get* c 69% до 51%.

Теперь обратимся нагрузочному тестированию PUT запросов посредством использования скрипта, написанного на lua.

|  |
| --- |
| *wrk -t1 -c1 -d10m -R000 –latency -s wrk/put.lua http://127.0.0.1:8080* |



Из результатов профилирования видно, что большую часть занимает flush данных на диск, а именно ~27,5%, что соизмеримо с затратами на передачу запроса по сети ~21%. Также видно, что достаточно весомую часть процессорного времени занимает процесс логирования - ~14%. Напишем, конфиг для установления более строгого уровня логирования и повторим эксперимент. Получили прирост. Теперь sendResponse занимает -32%, а общение с DAO - 25%.

Также при повторных тестах можно увидеть, что если flush происходит не очень часто, то получаем следующий результат, где делегирования запроса к DAO занимает всего 8%.

