

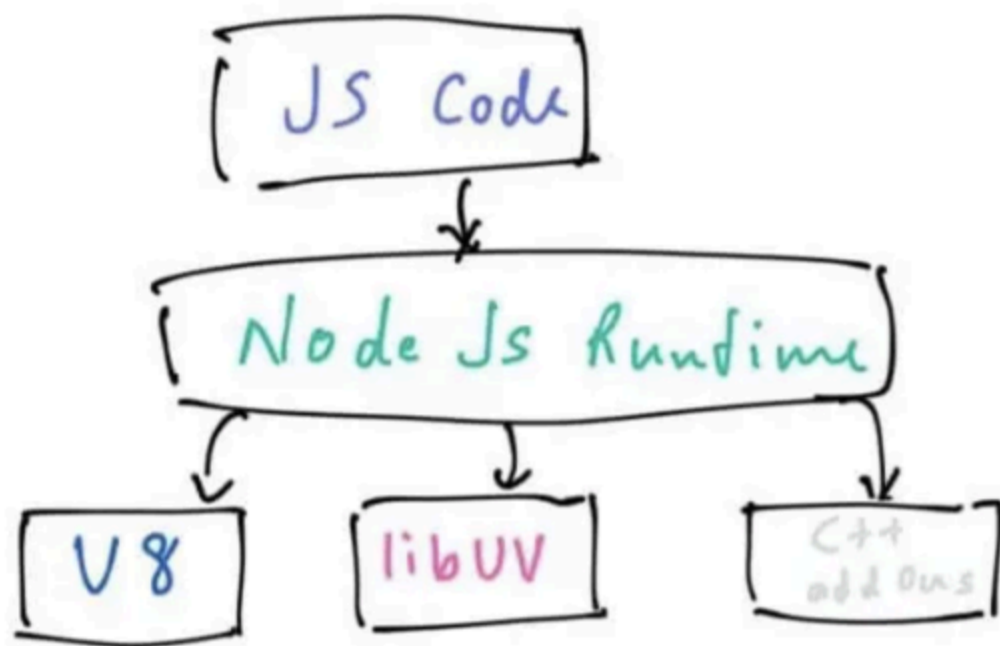
# Memory leak

Heap out of memory или утечки памяти в Node.js

## План доклада

- Краткий экскурс в архитектуру Node.js
- Утечки памяти - определение, кейсы и дебаггинг

# Архитектура Node.js - 1й вариант



**V8**





# V8

- Разработан датчанами из Google для повышения производительности и масштабируемости
- Написан на C++
- Благодаря нему можно использовать Node.js для написания обычных приложений (VScode, Postman)



## А что было до V8

- JS-код был только клиентским и мог воспроизводиться только в браузере

## Поэтому, он полезен ещё тем, что -

- Предоставляет функции, связанные с запуском JS-файлов (компилирует и выполняет)
- Обработывает стек вызовов\*
- Управляет выделением памяти для объектов JS
- Сбор мусора

**libuv**

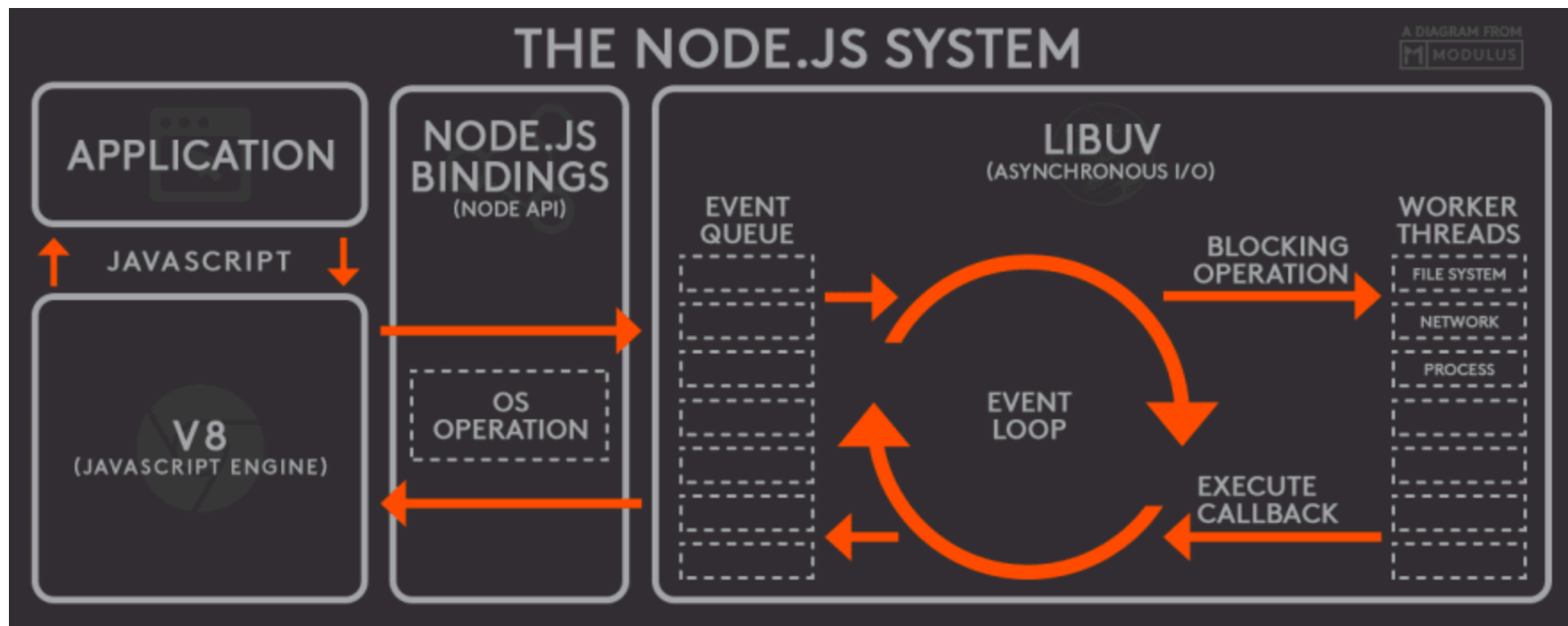


**libuv**

# libuv

- Сторонняя библиотека
- Написана на C
- Предоставляет доступ к операционной системе
- Предлагает функции, самые важные из которых - Event loop, Streaming, FileAsync, Pipes, ChildProcesses

**Как это работает всё вместе?**



## **Доп.вопрос - чем эта модель отличается от асинхронности в других ЯП?**

- Неблокирующий ввод-вывод, обратный вызов
- Go - параллелизм с доступом ко всем ядрам ЦП и общением через каналы



## V8 и память

- V8 - движок
- В его ЖЦ есть операции выделения памяти под собственные объекты и структуры данных
- Но нас интересуют `stack` и `heap` (стек и куча)

## V8 и память. Стек

- Это структура данных
- Список элементов, которые обрабатываются и хранятся по принципу LIFO
- В стеке JS хранит примитивные типы данных  
(string/number/boolean/null/undefined) и ссылки на всё остальное
- Стек - статическое\* выделение памяти

## V8 и память. Куча

- Это дерево
- Хранятся элементы, размер которых неизвестен заранее
- V8 выделяет на них память по мере необходимости
- Куча - динамическое\* выделение памяти

**Доп.вопрос - чем этот подход в работе с памятью отличается от других ЯП?**

## V8 и мусор

- Объекты-сироты из кучи, на которые никто не ссылается из стека (прямо или косвенно) являются мусором
- Сборщик мусора освобождает неиспользуемую память для того, чтобы повторно её использовать
- В V8 СМ'ы поколенческие, используются два этапа, но оба `stop the world`

## **Доп.вопрос - как справляются языки без СМ?**

- Ручное управление памятью
- Rust - области видимости

## Утечки памяти

## Memory Leak

2



## Sorcery



Target opponent reveals their hand. You choose a nonland card from that player's graveyard or hand and exile it.

Cycling 1 (1, Discard this card: Draw a card.)

*“By halves does youth depart. I remember halves of names, halves of years, halves of myself.”*

—Ogaro, wandering sage



## Утечки памяти

- Цикл потребления памяти - выделение, потребление, освобождение
- Утечка - это ситуация, при которой третий шаг не наступает
- Чаще всего это происходит потому, что где-то до сих пор хранится ссылка на данные


## **Опыт и статьи из интернета**

## Статьи из интернета

## Статьи из интернета

- Всегда предлагают следующие потенциальные случаи утечек:
- Глобальные переменные
- Множественные ссылки
- Замыкания
- Таймеры и события
- Вебсокеты/соединения

# Глобальные переменные

```
1 function calculateArea(width, height) {  
2   area = width * height;  
3    return area;  
4 }  
5  
6 calculateArea( width: 10, height: 5 );
```

# Таймеры и события

```
1  function handleInterval() : void {  
2      const array1 : any[] = [...]  
3      const array2 : unknown[] = array1.map(() => {...})  
4  }  
5  setInterval(handleInterval, timeout: 1000)
```



# Замыкания

```
1 function createCountdown(start) {  
2   let count = start;  
3  
4   return function() {  
5     return count--;  
6   };  
7 }  
8  
9 let countdownFrom10 = createCountdown( start: 10);
```

# Опыт

## Мои кейсы

- Подозрения на утечки, hear out of memory
- Утечки коннектов к redis/db
- Исследование почтабанка

## Подозрения на утечки, heap out of memory

- Неоптимальная работа с памятью

**Задача - выгрузить в архивы много xml файлов.**

**Консольная команда.**

<--- Last few GCs --->

```
[22876:000001DF911E2DB0] 250167 ms: Mark-sweep 1598.8 (1674.9) -> 1598.7 (1643.9) MB, 1267.1 / 0.0 ms last resort GC in old space requested
[22876:000001DF911E2DB0] 251436 ms: Mark-sweep 1598.7 (1643.9) -> 1598.7 (1643.9) MB, 1269.5 / 0.0 ms last resort GC in old space requested
```

<--- JS stacktrace --->

==== JS stack trace =====

Security context: 000000B3130A5879 <JSObject>

```
1: completeMany [D:\Pallari's Code\dhaweeye\dhaweeye\node_modules\mongoose\lib\query.js:~1456] [pc=000001366F8263A0](this=0000003DF8F8C209 <JSGlobal Object>,model=000000F09890C5C1 <JSFunction model (sfi = 000000364FCFE0FA1)>,docs=0000003BD57A91A61 <JSArray[57740]>,fields=0000003BD57A885F1 <Object map = 0000003D357E023B9>,userProvidedFields=0000003BD57A886C9 <Object map = 0000003D357E023B9>,pop=0...
```

FATAL ERROR: CALL\_AND\_RETRY\_LAST Allocation failed - JavaScript heap out of memory

```
1: node_module_register
2: v8::internal::FatalProcessOutOfMemory
3: v8::internal::FatalProcessOutOfMemory
4: v8::internal::Factory::NewUninitializedFixedArray
5: v8::internal::WasmDebugInfo::SetupForTesting
6: v8::internal::interpreter::BytecodeArrayRandomIterator::UpdateOffsetFromIndex
7: 000001366F5043C1
```

[nodemon] app crashed - waiting for file changes before starting...

1 usage 1 petr+1 \*

```
public async archiveByDatasetCode(code: string) {  
    /** Удаляем ранее созданный архив */  
    await fs.promises.rm( path: `./files/archive/${code}.zip`, options: { force: true });  
    /** Получаем данные из БД */  
    const data = await this.dsQueries.getEpiccrisisFromDataset(  
        ArchiveDatasets[code],  
    );  
    /** Инициализируем архив */  
    const zip = new AdmZip();  
    for (const value of data) {  
        const name = value['p'].replace(' ', '_') + '_v' + value['v'];  
        zip.addFile( entryName: `${name}.xml`, Buffer.from(value['native'], 'utf8'));  
    }  
    /** Сохраняем архив */  
    zip.writeZip( targetFileName: `./files/archive/${code}/${code}.zip`);  
}
```



# Проблемы

- Плавающая ошибка heap out of memory, т.к cron
- Локально не воспроизводится
- Увеличение выделенной памяти под old результатов не дало

## Диагностика

- Визуально - память утекать не должна
- Потребление зашкаливает, нужен рефакторинг
- После рефакторинга - замеры потребления

# Рефакторинг

- Limit/Offset в выборке
- Много маленьких архивов вместо 1 большого в памяти
- 1 большой архив по пути, а не в памяти

```

    /** Удаляем ранее созданный архив */
    await fs.promises.rm( path: `./files/archive/${code}.zip`, options: { force: true });
    /** Анализируем датасет на кол-во элементов (уникальных, по source_file_id) */
    const count = await this.dsQueries.preflightLengthAnalyze(
        ArchiveDatasets[code],
    );
    if (count.length > 0) {
        const iterationsCount = Math.ceil( count.length / this.limit);
        for (let i = 0; i < iterationsCount; i++) {
            /** Объект архива */
            const zip = new AdmZip();
            const leftMargin = i * this.limit;
            const rightMargin = i === 0 ? this.limit : this.limit * i * 2;
            const iterationSlice = count.slice(leftMargin, rightMargin);
            const data = await this.dsQueries.getEpicrisisFromDataset(
                ArchiveDatasets[code],
                iterationSlice,
            );
            for (const value of data) {
                /** Имя в формате Эпикриз_123_v1 */
                const name = value['p'].replace(' ', '_') + '_v' + value['v'];
                zip.addFile( entryName: `${name}.xml`, Buffer.from(value['native'], 'utf8'));
            }

            zip.writeZip( targetFileName: `./files/archive/${code}/${code}-${i}.zip`);
        }
        /** Рекурсивная распаковка ранее созданных архивов */
        for (let i = 0; i < iterationsCount; i++) {
            const zip = new AdmZip( fileNameOrRawData: `./files/archive/${code}/${code}-${i}.zip`);
            zip.extractAllTo( targetPath: `./files/archive/${code}`, overwrite: true);
            fs.rmSync( path: `./files/archive/${code}/${code}-${i}.zip`);
        }
        /** Архивируем итоговую папку */
        const totalDirZip = new AdmZip();
        totalDirZip.addLocalFolder( localPath: `./files/archive/${code}`);
        totalDirZip.writeZip( targetFileName: `./files/archive/${code}.zip`);
        /** Удаляем папку с эпикризами */
        await fs.promises.rm( path: `./files/archive/${code}`, options: {

```

# Замеры

```
leak - y.js  
ow Help  
README.md × package.json × index.html × y.js ×  
1 process.memoryUsage()  
2  
3  
4 process.memoryUsage().heapUsed //объём используемой кучи  
5 process.memoryUsage().heapTotal //объём кучи  
6
```

```
datasource initialize
2023-12-29T20:31:52.994Z
{ code: 'doctor-qualifying-introductory' }
{ length: 0 }
{ memoryBeforeTotal: 1762582528, memoryBeforeUsed: 1701820496 }
{ code: 'doctor-semifinal-introductory' }
{ length: 1324 }
{ memoryBeforeTotal: 1762844672, memoryBeforeUsed: 1705798544 }
{ memoryInMainLoopTotal: 1788366848, memoryInMainLoopUsed: 1726497912 }
{ memoryInMainLoopTotal: 1812299776, memoryInMainLoopUsed: 1751375608 }
{ memoryInMainLoopTotal: 1859334144, memoryInMainLoopUsed: 1798994880 }
{ memoryInMainLoopTotal: 1613361152, memoryInMainLoopUsed: 1120950752 }
{ memoryInMainLoopTotal: 1371713536, memoryInMainLoopUsed: 1164782160 }
{ memoryInMainLoopTotal: 1267167232, memoryInMainLoopUsed: 1108848296 }
{ memoryInMainLoopTotal: 1379479552, memoryInMainLoopUsed: 1249736352 }
{ memoryInMainLoopTotal: 1267879936, memoryInMainLoopUsed: 1154785376 }
{ memoryInMainLoopTotal: 1282818048, memoryInMainLoopUsed: 1181333096 }
{ memoryInMainLoopTotal: 1300963328, memoryInMainLoopUsed: 1204817776 }
{ memoryInMainLoopTotal: 1536225280, memoryInMainLoopUsed: 1429695464 }
{ memoryInMainLoopTotal: 1558683648, memoryInMainLoopUsed: 1452397680 }
{ memoryInMainLoopTotal: 1582186496, memoryInMainLoopUsed: 1476025304 }
{ memoryInMainLoopTotal: 1608323072, memoryInMainLoopUsed: 1499567280 }
{ memoryInMainLoopTotal: 1590943744, memoryInMainLoopUsed: 1486450952 }
{ memoryInMainLoopTotal: 1564508160, memoryInMainLoopUsed: 1460170360 }
{ memoryInMainLoopTotal: 1546158080, memoryInMainLoopUsed: 1437008000 }
{ memoryInMainLoopTotal: 1523720192, memoryInMainLoopUsed: 1413780576 }
{ memoryInMainLoopTotal: 1495998464, memoryInMainLoopUsed: 1390985720 }
{ memoryInMainLoopTotal: 1472307200, memoryInMainLoopUsed: 1368138040 }
{ memoryInMainLoopTotal: 1455349760, memoryInMainLoopUsed: 1345433272 }
{ memoryInMainLoopTotal: 1580277760, memoryInMainLoopUsed: 1475044656 }
{ memoryInMainLoopTotal: 1531351040, memoryInMainLoopUsed: 1421781488 }
{ memoryInMainLoopTotal: 1384259584, memoryInMainLoopUsed: 1277953144 }
{ memoryInMainLoopTotal: 1443295232, memoryInMainLoopUsed: 1337249456 }
{ memoryInMainLoopTotal: 1392001024, memoryInMainLoopUsed: 1286033608 }
{ memoryInMainLoopTotal: 1402421248, memoryInMainLoopUsed: 1297143296 }
{
  memoryAfterMainLoopTotal: 1402421248,
  memoryAfterMainLoopUsed: 1296890688
}
{ code: 'doctor-final-introductory' }
{ length: 0 }
{ memoryBeforeTotal: 1115881472, memoryBeforeUsed: 1040325384 }
{ code: 'finder-qualifying-introductory' }
{ length: 141 }
{ memoryBeforeTotal: 1115881472, memoryBeforeUsed: 1040408592 }
```

## Результаты по кейсу

- Способ подойдёт, если Вы примерно понимаете, где потенциальная утечка
- Если скрипт выполняет достаточно долго, чтобы подключился GC
- Утечки не было, но потребовался рефакторинг
- Безудержное раздувание памяти - не выход
- После замеров девопсы ещё раз заглянули в лимиты



## Утечки коннектов к redis/db

- Утечки коннектов к redis/db

# Redis

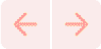
- Если самостоятельная работа, не через драйвер ФВ:
- Открыть коннект
- Выполнить работу
- Закрыть коннект

2 usages    Петр Кленкин

```
protected async setCacheBlockedPublicIds(  
  uploadSessionId: number,  
  sourceFileId: number,  
) {  
  try {  
    const client = new Redis( options: {  
      host: process.env.REDIS_DEFAULT_HOST,  
      port: +process.env.REDIS_DEFAULT_PORT,  
    });  
  
    await client.sadd(uploadSessionId.toString(), sourceFileId);  
    client.disconnect();  
  } catch (e) {  
    console.log(e.message);  
  }  
}
```

## DB

- Утекают чаще всего коннекты к БД, а не результаты/запросы
- Маркер - Ошибки БД в sentry/приложении



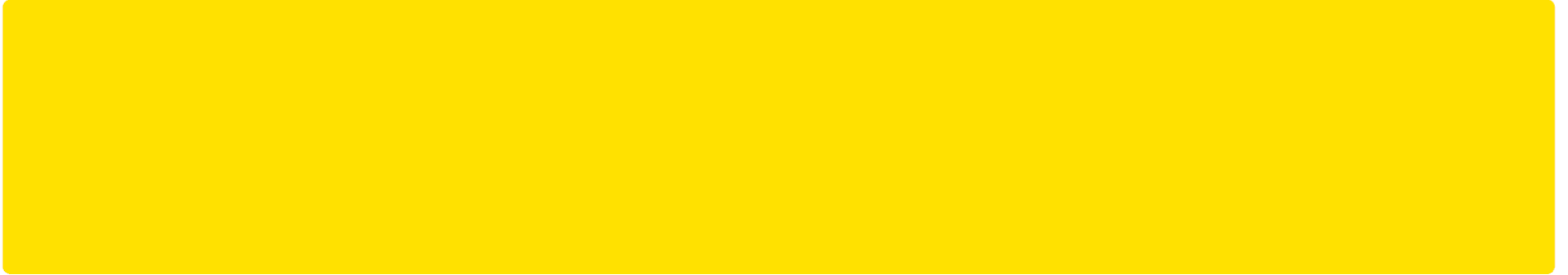
1 of 1 unhandled error

## Server Error

**error: too many connections for role "irmtoocg"**

This error happened while generating the page. Any console logs will be displayed in the terminal window.

## Call Stack



file:///Users/lavgoold/Desktop/lav/Projects/pokemon-battle/node\_modules/pg-protocol/dist/parser.js (39:38)

## Порядок действий

- Попытаться как можно уже определить use-case, которые вызывают эту ошибку
- Анализ траффика, опрос коллег
- При наличии 5< use-case подключиться к БД
- `select sum(numbackends) from pg_stat_database`

```
aimdoc=# SELECT sum(numbackends) FROM pg_stat_database;
sum
-----
    3
(1 row)

aimdoc=# SELECT sum(numbackends) FROM pg_stat_database;
sum
-----
    6
(1 row)

aimdoc=# SELECT sum(numbackends) FROM pg_stat_database;
sum
-----
    4
(1 row)

aimdoc=# SELECT sum(numbackends) FROM pg_stat_database;
sum
-----
    4
(1 row)

aimdoc=# SELECT sum(numbackends) FROM pg_stat_database;
sum
-----
    3
(1 row)

aimdoc=#
```

## Порядок действий

- Попытаться как можно уже определить use-case, которые вызывают эту ошибку
- Анализ траффика, опрос коллег
- При наличии 5< use-case подключиться к БД
- `select sum(numbackends) from pg_stat_database`
- Находим функционал, который не отпускает коннекты



## Решение (в моём случае)

- Проверка запросов
- Проверка транзакций (орм может работать с ними иначе)
- Проверка подключений и экспорта объекта подключения

```
controller.ts // options: {  
//           host: process.env.REDIS_HOST,  
module.ts //           port: process.env.REDIS_PORT,  
service.ts //       },  
// },  
};  
fig.ts const datasource = new DataSource(config);  
datasource.initialize();  
export default datasource;
```

```
41 };  
5+ usages  ⚙ Петр Кленкин  
42 const datasource = new DataSource(config);  
⚙ Петр Кленкин  
43 datasource  
44   .initialize() Promise<DataSource>  
45   .then(() => {  
46     console.log('datasource initialize');  
47   }) Promise<void>  
48   .catch((error) => {  
49     console.error('datasource initialize error');  
50     console.error(error.message);  
51     console.error(error.trace);  
52   });  
53 export { datasource };  
54
```

# Исследование почтабанка

## Исследование почтабанка. Инструменты

- `node --inspect`
- `require('v8').writeHeapSnapshot();`

## Исследование почтабанка. Условия

- Хорошая рабочая машина
- Prod сборка
- Проксирование
- Возможность в случае v8 делать дампы периодически
- Или делать дампы в graceful shutdown

**Задача - получить дампы памяти, чтобы было, что анализировать**

## writeHeapSnapshot

- Дополнительные пакеты не нужны
- Добавляем конструкцию прямо в код
- Ждём время (долго) и находим файл в ФС
- F12 + Загрузить снимок



```
342
343
344     server.listen(3000, err => {
345         if (err) throw err;
346         if (dev) {
347             console.log(`> Proxy-server has been started on http://localhost:${port}`);
348             console.log(`> All requests from ${LOCAL_URL} proxying on ${PROXY_URL}`);
349         }
350         setInterval(() => {
351             v8.writeHeapSnapshot('myHeap.' + Date.now() + '.heapsnapshot');
352         }, 1000 * 60 * 60)
353
354         // makeDump()
355     });
356
357
```

24.2 MB ↑3.5 kB/c Общий размер памяти JavaScript

Сделать снимок

Загрузить

Сводка | Фильтр классов | Все объекты

#### Конструктор

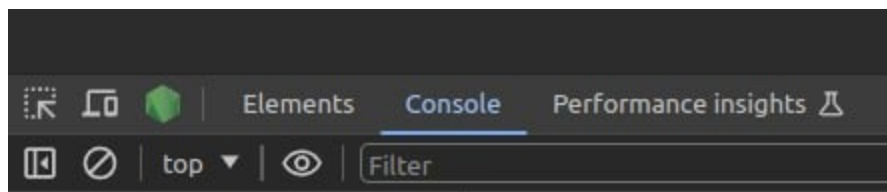
- ▶ global ×2
- ▶ Pack ×5
- ▶ system / Context ×16318
- ▶ PackFileCacheStrategy ×3
- ▶ Promise ×64
- ▶ ArrayBuffer ×960
- ▶ system / JSArrayBufferData ×947
- ▶ Buffer ×1695
- ▶ Array ×54791
- ▶ (string) ×228902
- ▶ (compiled code) ×185005
- ▶ CachedSource ×264
- ▶ Function ×55798
- ▶ (array) ×67502
- ▶ Object ×40604
- ▶ Map ×7177
- ▶ PackItemInfo ×79491
- ▶ Set ×7508
- ▶ (object shape) ×40479
- ▶ PackContent ×31
- ▶ (concatenated string) ×59481
- ▶ FileSystemInfo ×6
- ▶ AsyncQueue ×57
- ▶ AsyncQueueEntry ×1707
- ▶ Hook ×2573
- ▶ NormalModule ×252
- ▶ (number) ×90555
- ▶ CacheBackend ×24

Расстоя...	Объем памяти, ...	Сохраненный...
1	72 0 %	528 486 361 93 %
9	504 0 %	412 237 102 73 %
3	994 344 0 %	391 959 500 69 %
9	744 0 %	354 998 460 63 %
3	3 056 0 %	353 321 828 62 %
3	84 448 0 %	259 297 071 46 %
6	259 203 947 46 %	259 203 947 46 %
6	162 720 0 %	257 662 276 45 %
3	1 753 400 0 %	250 020 152 44 %
2	249 883 976 44 %	249 883 976 44 %
3	16 230 008 3 %	58 750 600 10 %
13	21 120 0 %	54 246 408 10 %
2	3 356 072 1 %	51 794 543 9 %
2	16 708 512 3 %	38 996 024 7 %
2	1 906 744 0 %	28 876 498 5 %
2	229 648 0 %	27 670 649 5 %
12	5 087 424 1 %	6 918 992 1 %
3	240 248 0 %	5 728 279 1 %
2	3 858 400 1 %	3 919 360 1 %
11	2 480 0 %	3 203 272 1 %
3	1 903 392 0 %	3 112 952 1 %
10	2 736 0 %	2 455 360 0 %
11	7 752 0 %	2 301 720 0 %
14	122 904 0 %	2 099 968 0 %
11	391 096 0 %	1 810 744 0 %
12	82 656 0 %	1 771 191 0 %
2	1 448 880 0 %	1 448 880 0 %
9	3 264 0 %	1 360 602 0 %

Сохраненные пути

## node --inspect

- Билдим проект `npm run build`
- Запускаем проект с опцией `--inspection`
- Заходим на сайт, жмём `f12`

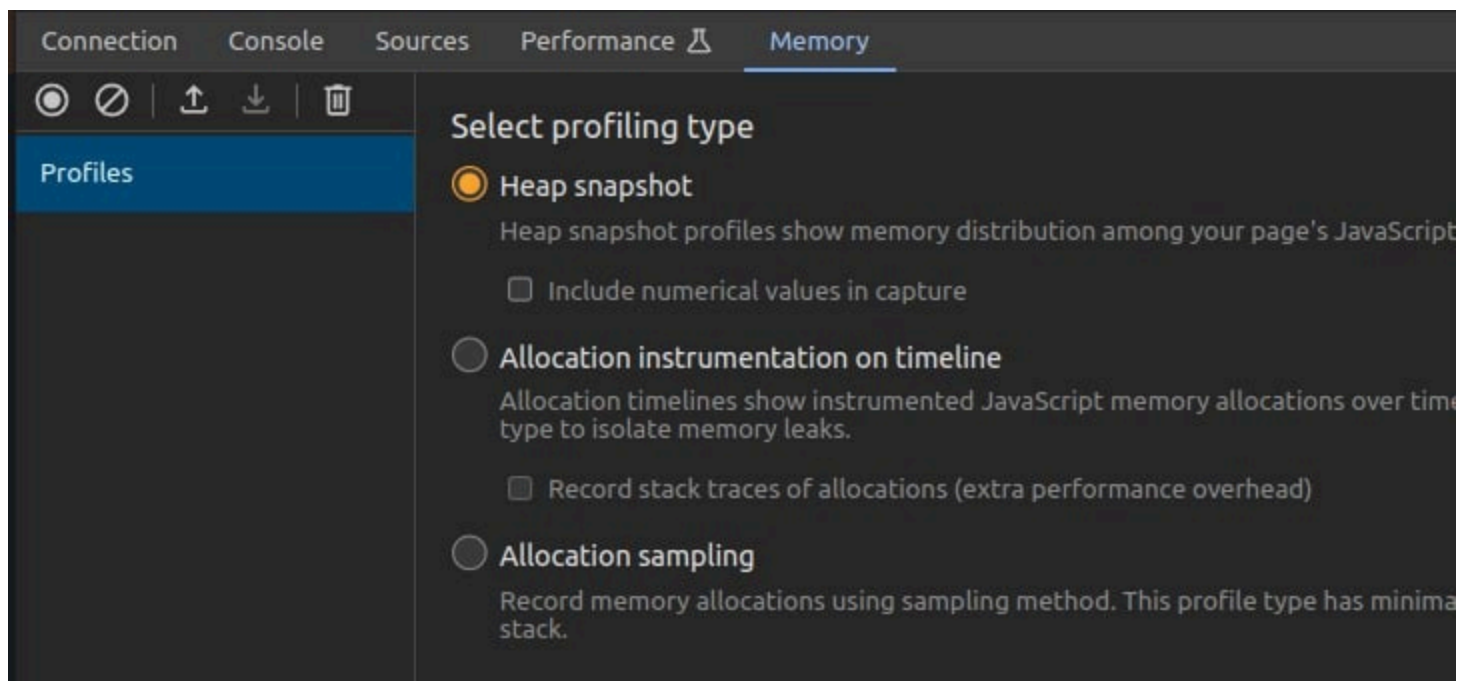




Specify network endpoint and DevTools will connect to it automatically. Read [Node.js debugging guide](#) to learn more.

localhost:9229

Add connection



Connection Console Performance **Memory** Sources

Profiles

HEAP SNAPSHOTS

Snapshot 1  
215 MB

## Select profiling type



## Heap snapshot

Heap snapshot profiles show memory distribution among your page's JavaScript objects and related DOM nodes.



Include numerical values in capture



## Allocation instrumentation on timeline

Allocation timelines show instrumented JavaScript memory allocations over time. Once profile is recorded you can select a time interval to see objects that were allocated within it and still alive by the end of recording. Use this profile type to isolate memory leaks.



Record stack traces of allocations (extra performance overhead)



## Allocation sampling

Record memory allocations using sampling method. This profile type has minimal performance overhead and can be used for long running operations. It provides good approximation of allocations broken down by JavaScript execution stack.

## Select JavaScript VM instance

27.4 MB ↓ 133 kB/s Node.js: file:///Users/pavel/projects/pochtabank.ru/apps/front/prod-server.js

27.4 MB ↓ 133 kB/s Total JS heap size

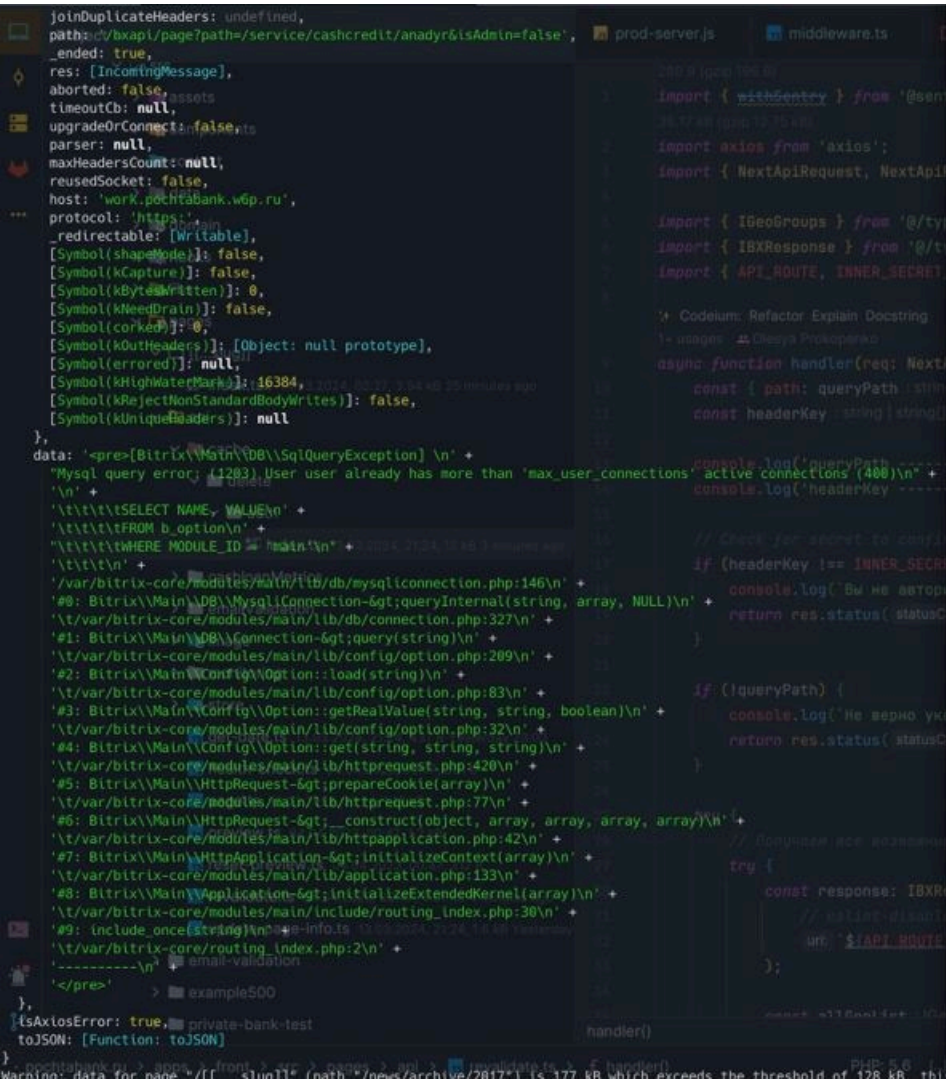
Take snapshot

Load



## **Задача - эмулировать пользователя через краулер**

- Берём любой краулер
- Настраиваем и травим его на сайтмап
- Следим за потребляемой памятью и смотрим логи



## Connection

## Console

Performance 

## Memory

## Sources

### Select profiling type

- ☒ Heap snapshot

Heap snapshot profiles show memory distribution among your page's JavaScript o

- ☐
- Include numerical values in capture

## Allocation instrumentation on timeline

Allocation timelines show instrumented JavaScript memory allocations over time. Click on an allocation to see objects that were allocated within it and still alive by the end of recording. Use

- ☐
- Record stack traces of allocations (extra performance overhead)

- Allocation sampling

Record memory allocations using sampling method. This profile type has minimal overhead and performs only a few operations. It provides good approximation of allocations broken down by JavaScript engine.

### Select JavaScript VM instance

291 MB **↑29.1 kB/s** Node.js: file:///Users/pavel/projects/pochtabank.ru/apps/fr

291 MB   ↑29.1 kB/s   Total JS heap size

### Take snapshot

Load

Конструктор	Расстоя...	Объем памяти, ...	Сохраненный...
▼ global ×2			
▶ global @6259			
▶ global @1681			
▶ Pack ×5			
▶ system / Context ×16318			
▶ PackFileCacheStrategy ×3			
▶ Promise ×64			
▶ ArrayBuffer ×960			
▼ system / JSArrayBufferData ×947			
system / JSArrayBufferData @982733			
system / JSArrayBufferData @982725			
system / JSArrayBufferData @1109675			
system / JSArrayBufferData @1109677			
system / JSArrayBufferData @1386285			
system / JSArrayBufferData @2001545			
system / JSArrayBufferData @1109919			
system / JSArrayBufferData @1386321			
system / JSArrayBufferData @1763201			
system / JSArrayBufferData @945277			
system / JSArrayBufferData @1762129			
system / JSArrayBufferData @1632453			
system / JSArrayBufferData @1632449			
system / JSArrayBufferData @1632373			
system / JSArrayBufferData @1237691			
system / JSArrayBufferData @1239649			
system / JSArrayBufferData @1231801			
system / JSArrayBufferData @943365			
system / JSArrayBufferData @1109679			

Сохраненные пути	Рассто...	Объем памяти, ...	Сохраненный р...
Объект			
▼ backing_store внутри ArrayBuffer @982715	20	88 0%	172 451 559 30%
▼ buffer внутри Buffer @982713	19	96 0%	172 451 655 30%
▼ N внутри system / Context @1432171	18	104 0%	172 452 167 30%
▼ previous внутри system / Context @1895525	17	40 0%	40 0%
▼ context внутри () @1109297	16	56 0%	96 0%
▼ v внутри system / Context @1109295	15	56 0%	152 0%
▼ context внутри () @1390361	14	56 0%	344 0%
▼ <symbol lazy serialization data> внутри () @1356413	13	56 0%	704 0%
▼ <symbol lazy serialization data> внутри () @1356409	12	56 0%	1 080 0%
▼ lazy внутри PackContent @777735	11	80 0%	1 704 0%
▼ [11] внутри Array @371831	10	32 0%	174 238 159 31%
▼ content внутри Pack @371821	9	152 0%	352 837 559 62%

**ВАЖНО - Делать снимки кучи нужно через  
временные интервалы и выполнение разных ис**

## Обозначившиеся проблемы

- Билдим проект `npm run build`
- Запускаем проект с опцией `--inspection`
- Заходим на сайт, жмём `f12`

## Решение этих проблем

- Висящие методы
- Статический контент вакансий, по какой-то причине плодящийся в памяти
- store - глобальная переменная
- Запрос в middleware всей страницы для id
- Бесконтрольные Console.log
- Неосторожная работа со гигантскими строками (replace всего контента)
- Методами с дублированием и копированием гигантских массивов данных

## Итоги

- Нарастающая память по 30 мб (в данном случае) ушла
- После прогрева кешей память фиксируется и GC под нагрузкой успевает увести её в адекватные значения
- Это ещё далеко не всё, что есть на ПБ, рефакторинг критически необходим

**Конец**