Производительный NodeJS

Вукмирович Александр

Сегодня

Объединение запросов

Кэширование

Масштабирование приложения

Redis

Пример

Просмотрщик домашних заданий

Пример

Просмотрщик домашних заданий

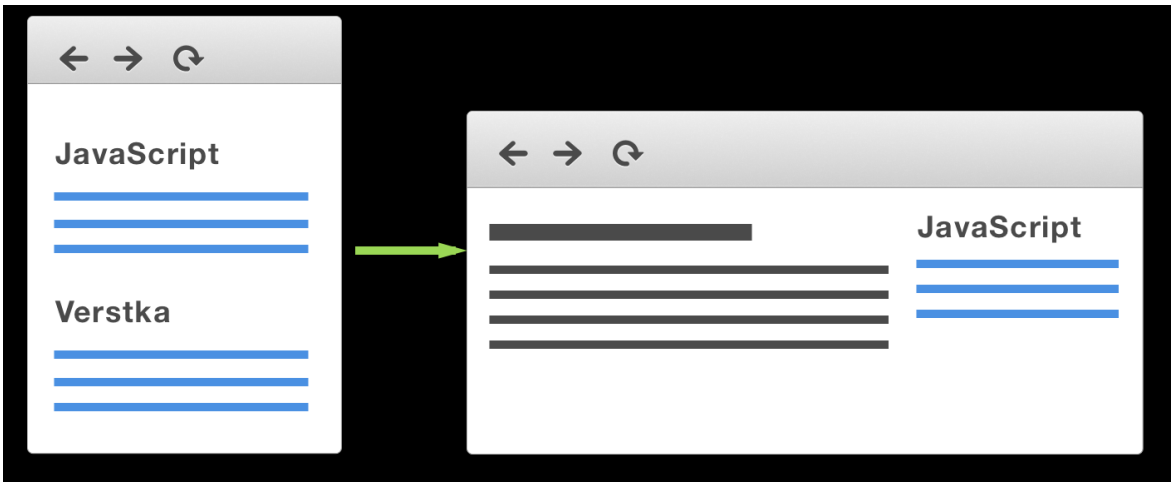
Навигация по домашним задачам

Должно быть быстро

Держать высокую нагрузку

Пример

Просмотрщик домашних заданий



Пример

getTasks(category)

```
function getTasks(category) {
    return github.getRepos('urfu-2015')
        .then(tasks => filterTasks(tasks, category))
        .then(getTasksInfo);
};
```

Пример

Главная страница

```
Promise
    .all([
        getTasks('javascript'),
        getTasks('verstka'),
        getTasks('webdev')
    ])
    .then(results => {
        res.render(...);
    });
```

Пример

Главная страница

```
Promise
    .all([
        getTasks('javascript'),
        getTasks('verstka'),
        getTasks('webdev')
    ]);

function getTasks(category) {
    return github.getRepos('urfu-2015')
        .then(...)
        .then(...);
};
```

Пример

Демо. Три одновременных пользователя

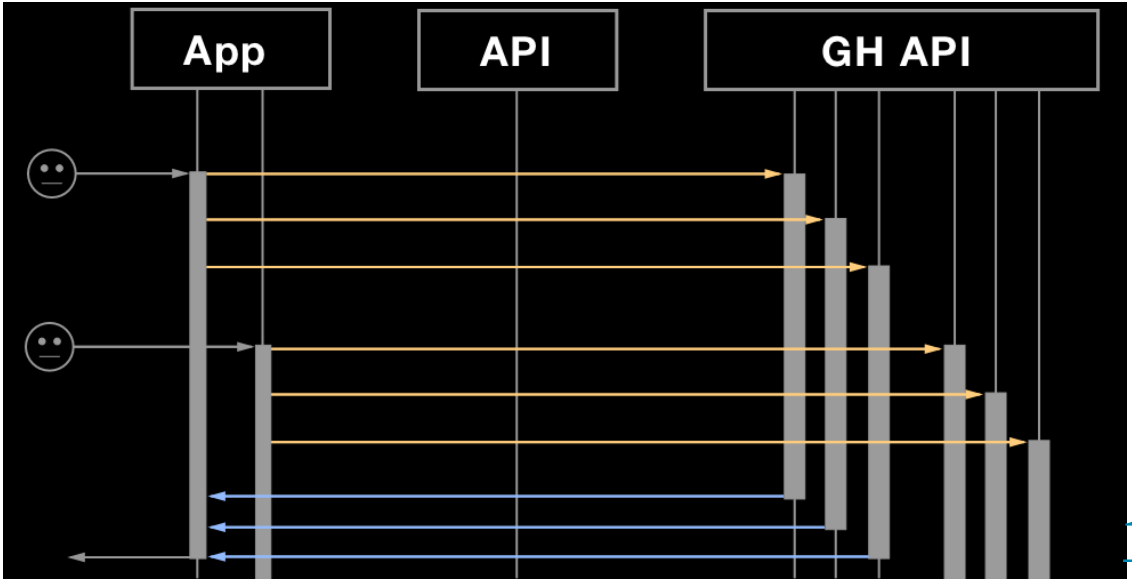
```
require('debug-http')();


GET https://api.github.com/orgs/urfu-2015/repos 891ms
GET https://api.github.com/orgs/urfu-2015/repos 889ms
GET https://api.github.com/orgs/urfu-2015/repos 902ms
GET https://api.github.com/orgs/urfu-2015/repos 933ms
GET https://api.github.com/orgs/urfu-2015/repos 953ms
GET https://api.github.com/orgs/urfu-2015/repos 1,003ms
GET https://api.github.com/orgs/urfu-2015/repos 1,018ms
GET https://api.github.com/orgs/urfu-2015/repos 1,067ms
GET https://api.github.com/orgs/urfu-2015/repos 1,072ms
```

Много одинаковых запросов

Объединение запросов

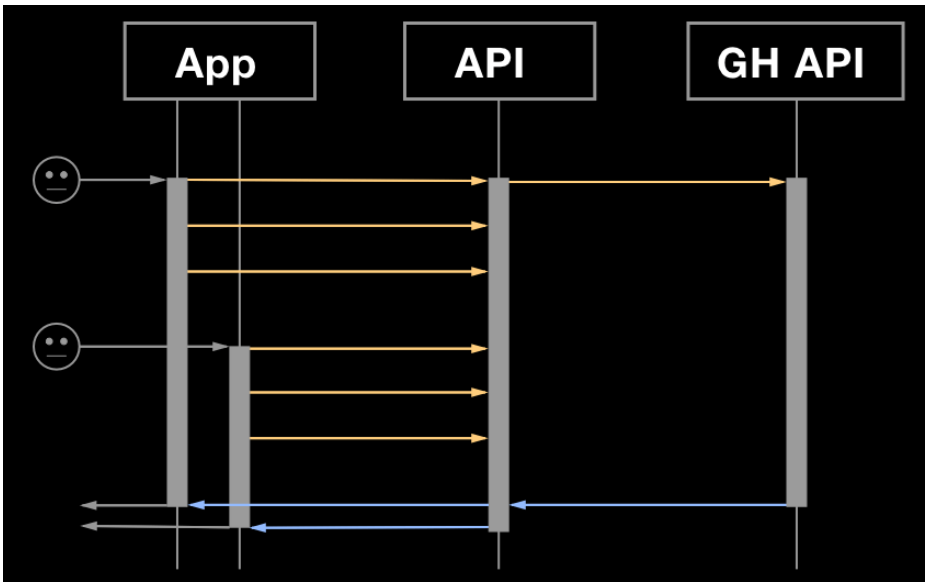
Объединение запросов

До объединения



Объединение запросов

После объединения



Объединение запросов

Объединяем запросы за репозиториями

```
let currentQuery;
function getReposBatch() {
    if (!currentQuery) {
        currentQuery = github.getRepos('urfu-2015')
            .finally(() => {
                currentQuery = null;
            });
    }
    return currentQuery;
}
```

Объединение запросов

Объединяем запросы за репозиториями

```
function getTasks(category) {
    return getReposBatch()
        .then(tasks => filterTasks(tasks, category))
        .then(getTasksInfo);
};
```

Объединение запросов

Демо. Пять одновременных пользователя

✓ Уменьшили количество запросов

✓ Быстрее отвечаем *некоторым* пользователям

? Если запрос завершился ошибкой, то для всех

Кэширование

Кэширование

Что это такое?

Данные ближе к месту использования

Хранение результатов вычислений

Оптимизация скорости получения данных

Кэширование

Решаемые проблемы

Низкая производительность

Избыточность запросов

Высокая нагрузка на внешний источник

Узкий сетевой канал

Высокие сетевые задержки

Кэш

Что это такое?

Промежуточное хранилище с быстрым доступом

Ограничен по размеру

Наиболее часто запрашиваемые данные

Данные не всегда актуальные

Кэш

Эффективность

Количество попаданий

```
Hit Rate = Попадание в кэш / Количество запросов
```

Скорость получения данных

“Разогретый” кэш

Кэш

Инвалидация

Данные устарели (“протухли”) и их нужно убрать

Данные удаляются вручную

Данные заменяются новыми

Данные вытесняются автоматически по алгоритму

Одна из самых сложных задач в программировании

Кэш

Алгортим вытеснения

Time period - Вытеснение по времени

LFU - Least frequently used
Вытеснение редко используемых

Кэш

Алгортим вытеснения

LRU - Least Recently Used
Вытеснение давно неиспользуемых

Segmented LRU - Многоуровневый LRU

От алгортима зависит быстродействие кэша

Кэш

Когда не кэшировать

Большая вариация данных

Персонализированные данные

Есть другие оптимизации

Мемоизация

Сохранение результата выполнения функции

```
const cache = new Map();
function memoize(key, fn) {
    if (!cache.has(key)) {
        const value = fn();
        cache.set(key, value);
        return value;
    }

    return cache.get(key);
}
```

Пример

Реализуем кэш

```
const LRU = require('lru-cache');
const cache = new LRU();

cache.set(key, value, maxAge);

cache.get(key);

cache.del(key);

cache.has(key);
```

Пример

Реализуем кэш

```
class Cache {
    constructor() {
        this._cache = new LRU();
    }

    memoize(key, maxAge, fn) {
        // ...
    }
}
```

Пример

Реализуем кэш

```
memoize(key, maxAge, fn) {
    const cache = this._cache;
    const value = cache.get(key);
    if (value) {
        return Promise.resolve(value);
    }
    return Promise.resolve()
        .then(fn)
        .then(result => {
            cache.set(key, result, maxAge * 1000);
            return result;
```

Пример

Без использования кэша

```
function getTasks(category) {
    return getReposBatch()
        .then(tasks => filterTasks(tasks, category))
        .then(getTasksInfo);
};
```

Пример

Кэшируем задачи

```
const cache = new Cache();

function getTasksCached(category) {
    const cacheKey = `tasks.${category}`;

    return cache.memoize(
        cacheKey,
        5 * 60,
        () => getTasks(category)
    );
}
```

Пример

Демо. Результаты кэширования

```
GET https://api.github.com/orgs/urfu-2015/repos 990ms
GET / 200 1249.771 ms
GET / 200 1254.626 ms
GET / 200 1236.532 ms
GET / 200 11.003 ms
GET / 200 18.849 ms
GET / 200 25.639 ms
```

Кэширование

Подведение итогов

Не решает всех проблем

Кэшировать только после всех оптимизаций

Кэш должен быть вспомогательной компонентой

Кэш должен легко включаться и отключаться

Все должно работать и без кэша

Кэш не постоянное хранилище!

Масштабирование

Масштабирование

Что это такое?

Процесс увеличения производительности и отказоустойчивости системы

Распределение нагрузки между несколькими процессами и машинами

Масштабирование

Что дает?

✓ Увеличивает доступность

✓ Увеличивает производительность

✓ Увеличивает отказоустойчивость

✗ Увеличивает сложность

Масштабируемость

Как свойство системы

Сильная. Система “легко” масштабируется

Слабая. Очень тяжело масштабируется

Масштабируемость ~ Монолитность

Масштабирование

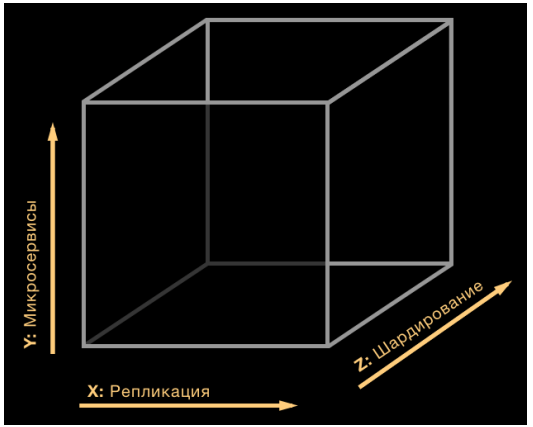
Виды масштабирования

Вертикальное. Добавление мощностей

Горизонтальное. Разбиение приложения на несколько частей

Масштабирование

Scale cube



ось X: Клонирование приложения

ось Y: Декомпозиция приложения

ось Z: Разделение в зависимости от данных

The Art of Scalability. Martin L. Abbott and Michael T. Fisher

Клонирование NodeJS-приложения

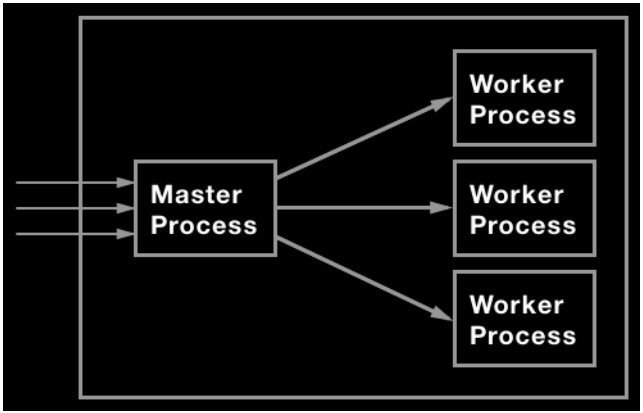
Запуск в разных процессах

Запуск на разных машинах

Решить вопрос балансировки запросов

Клонирование NodeJS-приложения

Cluster



Round-robin алгоритм балансировки

Клонирование NodeJS-приложения

Cluster

```
const cluster = require('cluster');
const os = require('os');

if (cluster.isMaster) {
    const cpus = os.cpus().length;
    for (let i = 0; i < cpus; i++) {
        cluster.fork();
    }
} else {
    require('./app.js');
}
```

Клонирование NodeJS-приложения

Демо. Cluster

✓ Распараллелена нагрузка

✗ При ошибке приложение недоступно

```
Availability:                   10.89 %
Successful transactions:        61
Failed transactions:            499
```

Клонирование NodeJS-приложения

Cluster. Обработка ошибок

```
if (cluster.isMaster) {
    // ...
    cluster.on('exit', (worker, code) => {
        if (code !== 0
                && !worker.exitedAfterDisconnect) {
            console.log('Worker crashed');
            cluster.fork();
        }
    });
}
```

Клонирование NodeJS-приложения

Демо. Cluster. Обработка ошибок

```
Availability:                87.10 %
Successful transactions:     243
Failed transactions:         36
```

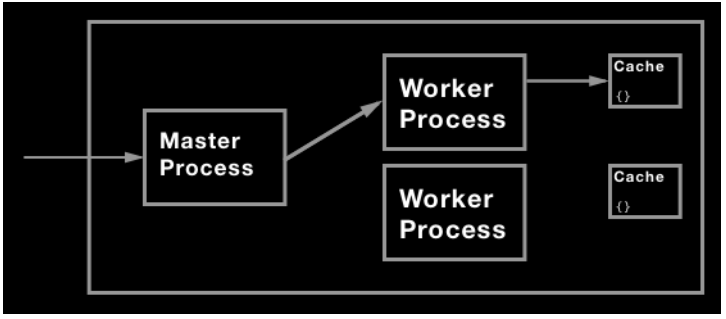
Масштабирование

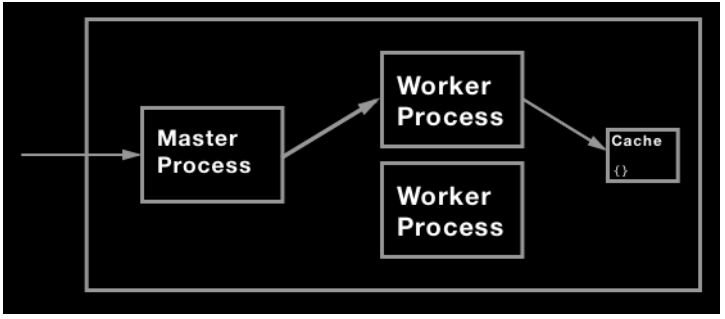
Подведение итогов

✓ Высокая доступность

✓ Отказоустойчивость

? Изолированная память


Master
Process
Worker
Process
Worker
Process
Cache
{}
Cache
{}


Master
Process
Worker
Process
Worker
Process
Cache
{}

Единый кэш

Shared cache

Единый кэш

Redis

✓ Хорошая документация

✓ Данные в памяти

✓ Транзакции

✓ Пакетная обработка команд

✓ Механизм pub/sub из коробки

✓ Поддержка LRU алгоритма

Redis

Try redis

The Little Redis Book

Using Redis as an LRU cache

Redis

Запуск в Docker

Скачиваем образ:

```
docker pull redis
```

Запускаем сервер в фоне:

```
docker run --name d-redis -p 6379:6379 -d --rm redis redis-server
```

Redis

Командная строка

Telnet:

```
telnet localhost 6379
```

Redis CLI (в Docker):

```
docker run -it --link d-redis:redis --rm redis redis-cli -h redis -p 6379
```

Список команд

Redis

Работа с ключами

Установка значения:

```
SET tasks.javascript "[{ name: ... }]"
```

Проверка существования ключа:

```
EXISTS tasks.javascript
```

Redis

Работа с ключами

Получение ключа:

```
GET tasks.javascript
```

Удаление ключа:

```
DEL tasks.javascript
```

Redis

Работа с ключами

Установка времени жизни ключа:

```
EXPIRE tasks.javascript 30
```

Получение времени жизни ключа:

```
TTL tasks.javascript
```

Установка значения вместе с временем жизни:

```
SETEX tasks.javascript 30 "[{ name: ... }]"
```

Пример

Использование Redis из NodeJS

```
const Redis = require("ioredis");
class Cache {
    constructor() {
        this._cache = new Redis(6379, '127.0.0.1');
    }
}
```

Пример

Текущая реализация кэша

```
memoize(key, maxAge, fn) {
    const cache = this._cache;
    const value = cache.get(key)
    if (value) {
        return Promise.resolve(value)
    }
    return Promise.resolve()
        .then(fn)
        .then(result => {
            cache.set(key, result, maxAge * 1000)
            return result;
```

Пример

Использование Redis как кэша

```
memoize(key, maxAge, fn) {
    const cache = this._cache;
    return cache.get(key)
        .then(value => {
            if (value) {
                return JSON.parse(value);
            }
            return Promise.resolve()
                .then(fn)
                .then(result => {
                    cache.setex(key, maxAge, JSON.stringify(result));
                    return result;
                });
        });
}
```

Итоги

Итоги

Объединение запросов

Кэширование

Масштабирование

Всегда нужно отталкиваться от задачи

Вопросы?