

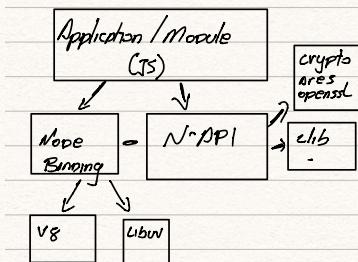
Node Under the hood

Javascript

Node é essencialmente JS com acesso ao filesystem então ele possui os mesmos conceitos e o mesmo modo de funcionamento

- Single Thread
- Paralelismo e concorrência
- Sync e Async
- I/O não bloqueante (libuv)
- Orientação a eventos (como o JS, mas focados em eventos de dados)

Node depende de poucos pacotes: v8, libuv, crypto



A N-API abstrai grande parte da API dos versões do node de forma própria

Libuv

Plataforma open source para I/O assíncrono desenvolvida pelo Node pelo próprio Ryan Dahl como uma copia do libev,

A grande vantagem do Node é que ele nunca trava a execução por conta de seu uso do libuv para criar um eventloop que desalota o processador e realoca novamente quando tem trabalho para fazer dessa forma o processador entra em estudo mode e nunca para (veja a talk do Ryan Dahl e mostrar o exemplo de requests em paralelo que leva 2s)

O libuv gerencia o eventloop, childprocess e sockets, além de permitir que o sistema não tenha sleeps.

Outline

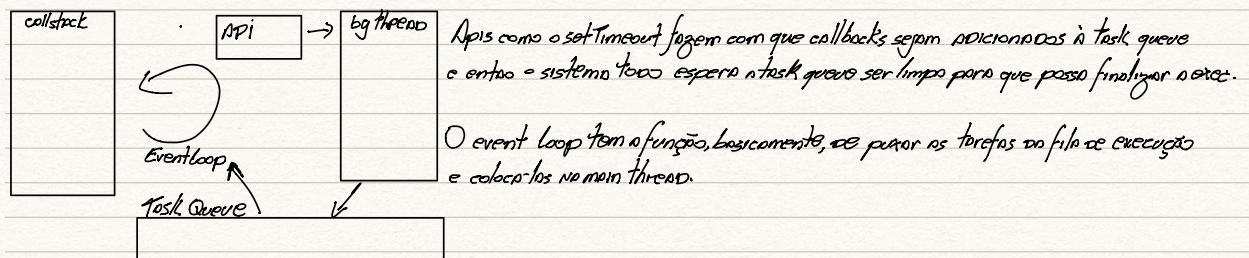
1. O que é Node
 - Breve história
 - Comentário sobre JS e seu história
 - Elementos do Node: v8, c, js
2. Estudando um chomodo de I/O
 - Apresentar o chomodo FS
 - Vamos acompanhar uma leitura assíncrona de arquivos
3. Javascript
 - Como funciona:
 - Callstack + Memória + Alocação
 - Heap ↗ Tail call optim.
 - EventLoop
4. Libuv
 - O que é
 - Como funciona
 - Como é implementado
5. V8
 - O que é
 - Como funciona (em linhas gerais)
 - AST e Esprima
 - Ignition
 - Turbo fan
 - Sea of Nodes ?
 - Optimizações do compilador
6. Native Modules
 - N-API
 - Acesso ao FS

Callstack

A callstack é o empilhamento de chamadas executors que trabalham no modelo Last in first out (LIFO).

O event loop é o que faz com que o JS consiga ter filhos chamados de Callback queues e Job Queues.

Junto com isso temos uma API externa que tem a capacidade de criar background tasks



A task queue pode ser dividida em duas partes:

- Macro tasks: Tasks como leituras de arquivos, I/O etc que são executadas pelo event loop e tem uma prioridade menor para serem executadas do que as micro tasks (podemos chamar essa fila de callback queue)

- Micro tasks: Tem prioridade maior e executam fora do thread, como promises... Podemos chamar de job queues

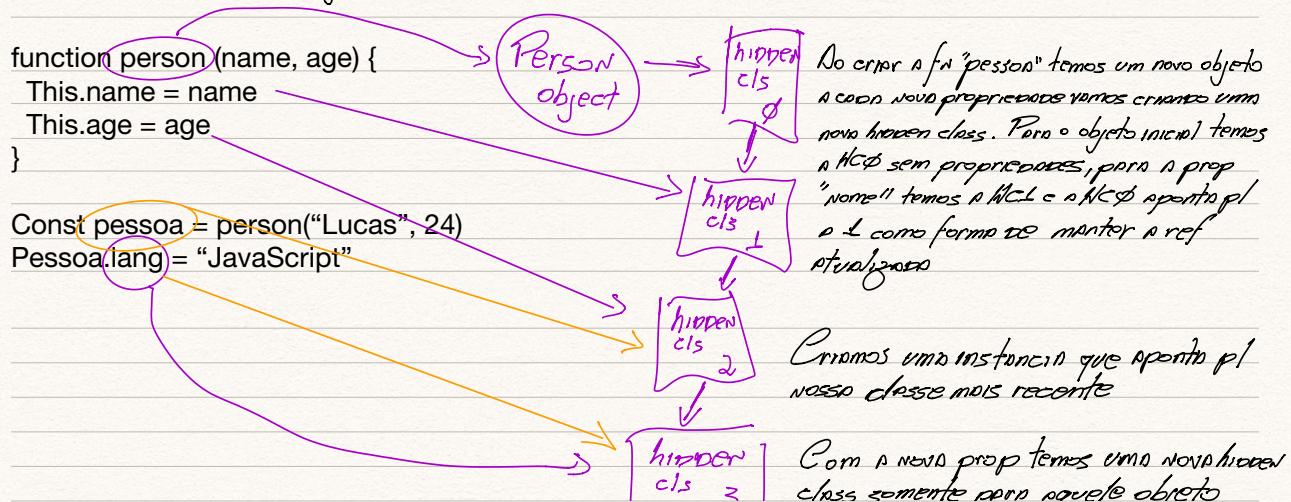
- V8

Motor Javascript por traz do Node. O mesmo utilizado pelo Chrome, e ele faz milagres com o que o JS pode fazer em termos de performance

→ Hidden Classes

O que torna o V8 tão performático é o uso de uma técnica chamada de hidden class. Em termos simples o V8 transforma estruturas de dados do JS em classes. Crie e aplica uma referência a elas e suas alterações

Para cada novo objeto o V8 cria um novo hidden class específico, e/ou caso modificação ele cria um novo apontando à antiga para o novo



```

function person (name, age) {
  This.name = name
  This.age = age
}

```

(se criarmos uma nova pessoa, ela não vai ter "lang")

```

Const p1 = new person("Lucas", 24)
p1.lang = "js" // { name: "Lucas", age: 24, lang: "js" } //

```

↪ Nc3

```

Const p2 = new person("Ana", 20) // { name: "Ana", age: 20 } //

```

↪ Nc2

Dynamic Machine code generation (DMCG)

No compilar pela primeira vez, o compilador vai somente aplicar algumas otimizações básicas como o inline expansion e constant folding. Após isso o código é executado, analisado e recompilado com melhorias como caching inline

