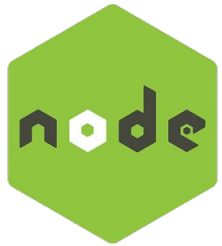
NodeJS



Prof. Celso Henrique Masotti

Você verá nesta aula:

1. A criação do NodeJS
2. O que é Node.JS?
3. V8
4. Thread
5. Multithread
6. Multithread (Outras tecnologias)
7. Single Thread
8. Stack
9. Event-loop (operação curta)
10. Event-loop (operação longa)
11. Conclusão
12. Outras Informações

A criação do NodeJS

Criado por **Ryan Dahl** em 2009, focado em migrar a programação do Javascript do cliente (frontend) para os servidores.

Dahl se inspirou depois de ver a barra de progresso de upload de arquivos no Flickr, ele percebeu que o navegador não sabia o quanto do arquivo foi carregado e tinha que consultar o servidor web. Pensando nisso, procurou maneiras do JavaScript também trabalhar “back end”.

Ryan Dahl palestrando na YUIConf 2010 em Sunnyvale (California, EUA).

O que é NodeJS?

Node é um interpretador de linguagem javascript que não depende do browser (navegador). Este interpretador possui algumas características:

- **Código aberto**: promove o licenciamento livre;
- **Orientado a eventos**: não segue o fluxo de controle padronizado como acontece nas linguagens tradicionais, o controle de fluxo de programas orientados a evento são guiados por indicações externas, chamadas eventos.
- **Alta escalabilidade**: possui a capacidade de manipular uma porção crescente de trabalho de forma uniforme, estando preparado para crescer.

O nodeJS pode manipular milhares de conexões/eventos simultâneas em tempo real numa única máquina física.

V8

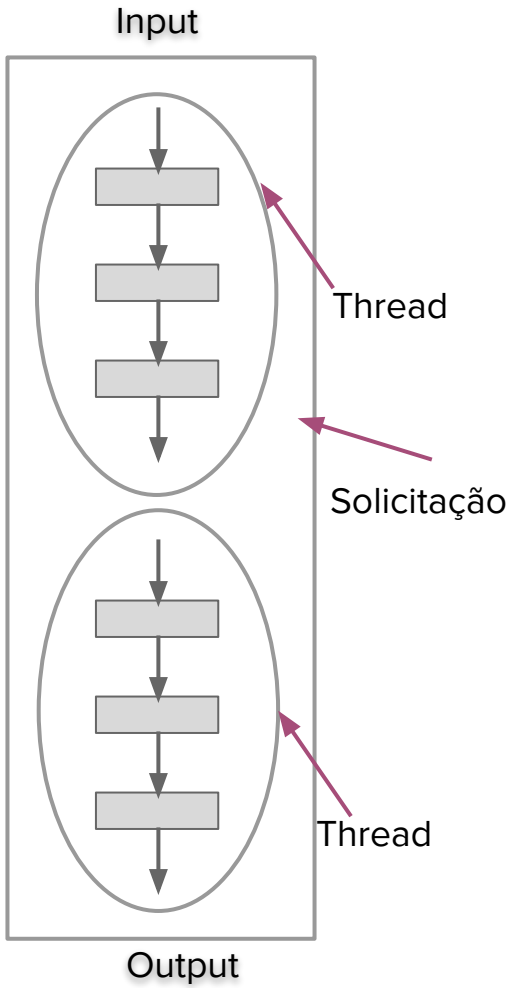
O V8 é uma engine criada pela Google para ser usada no browser chrome. Em 2008 a Google tornou o V8 open source e passou a chamá-lo de ***Chromium*** project. Essa mudança possibilitou que a comunidade entendesse a engine em sí, além de compreender como o javascript é interpretado e compilado por esta.

O javascript é uma linguagem interpretada, o que o coloca em desvantagem quando comparado com linguagens compiladas, pois cada linha de código precisa ser interpretada enquanto o código é executado. O V8 compila o código para linguagem de máquina, além de otimizar drasticamente a execução usando heurísticas, permitindo que a execução seja feita em cima do código compilado e não interpretado.

Thread

Quando fazemos uma solicitação ou solicitações a um determinado sistema, uma aplicativo subdivide esta(s) solicitação(ões) em pequenas tarefas e providencia suas execuções. É como uma pessoa que chega em uma lanchonete e pede (input) um sanduíche, batata frita e um refrigerante (solicitação). Para realizar tal pedido este é dividido em tarefas menores: um cuida de fritar a batata; outro pega o refriferante; outro frita a carne, coloca os molhos, queijo, aquece os pães, monta o sanduíche e encaminha para a entrega (output).

No computador é a mesma coisa. Uma solicitação é composta por atividades. A capacidade de dividir as atividades em ações ainda menores e especializadas é chamada de “**thread**”.

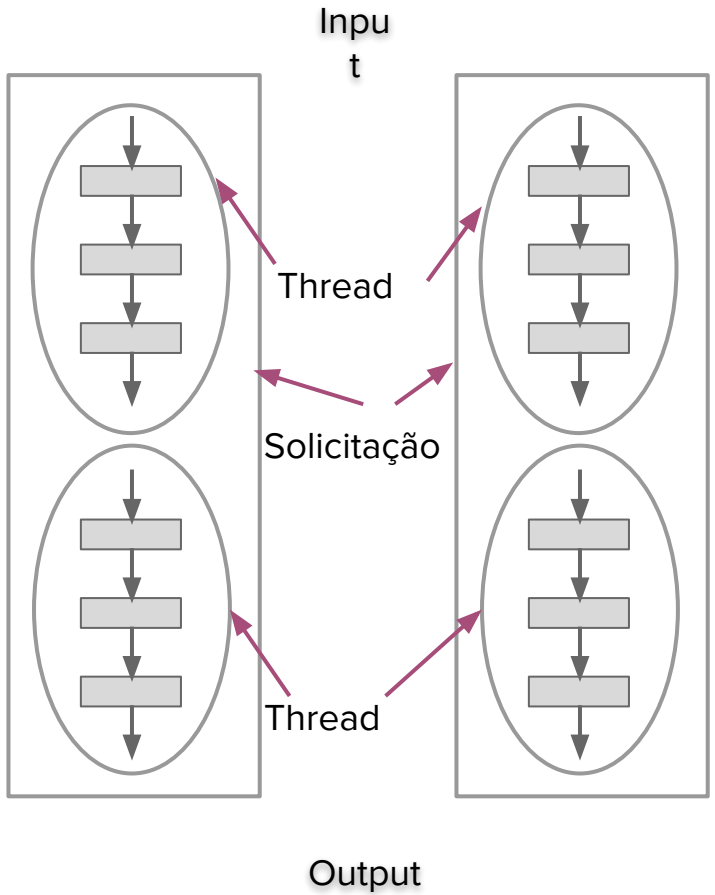


Multithread

Existem “thread” cujas ações é de fácil execução (pegar o refrigerante) e de complexa execução (fritar a batata ou cada uma das partes para se fazer o sanduíche). Assim as threads não são iguais.

Todas as threads são executadas pelo processador do computador. Então podemos afirmar que existem tarefas que são processadas rapidamente enquanto outras demandam um pouco mais de tempo.

Existem linguagens de programação que, trabalhando em servidores tradicionais, encaminham duas ou mais threads simultaneamente ao processador, estas linguagens são chamadas de “multithread”.



Multithread (Outras tecnologias)

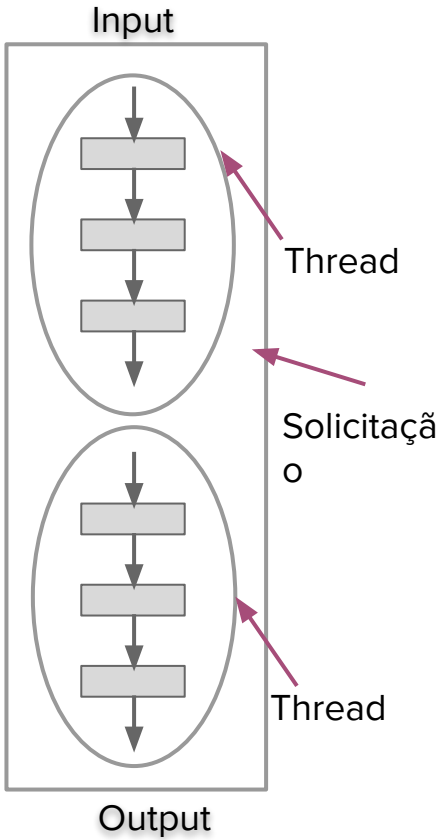
É comum alguns processadores terem grande capacidade e por este motivo ficar em parte na ociosos mesmo quando em processamento.

SuperThreading: trata-se de uma tecnologia desenvolvida com o objetivo principal de aproveitar períodos de ociosidade para a execução de instruções de outra thread, fazendo a máquina render seu máximo possibilitando um pronto retorno do evento solicitado pelo usuário.

Hyper-threading ou **Simultaneous multithreading (SMT)**: basicamente, seria uma espécie de evolução da tecnologia SuperThreading, porém sem a limitação de que todas as instruções executadas em um mesmo ciclo de processamento sejam da mesma thread.

Single Thread

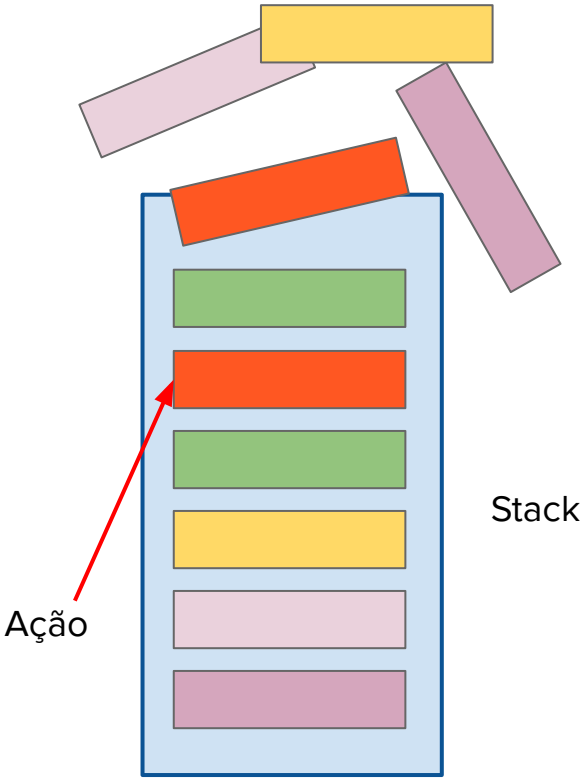
Todavia, o node.JS não encaminha mais que uma thread por vez para ser processada, ou seja, a característica deste modelo é denominado “single thread”.



Stack

Muitas vezes, em dado momento, podem existir mais que um usuário fazendo solicitações ao sistema. As solicitações precisam ser organizadas, divididas em pequenas partes (threads), para serem encaminhadas ao processador.

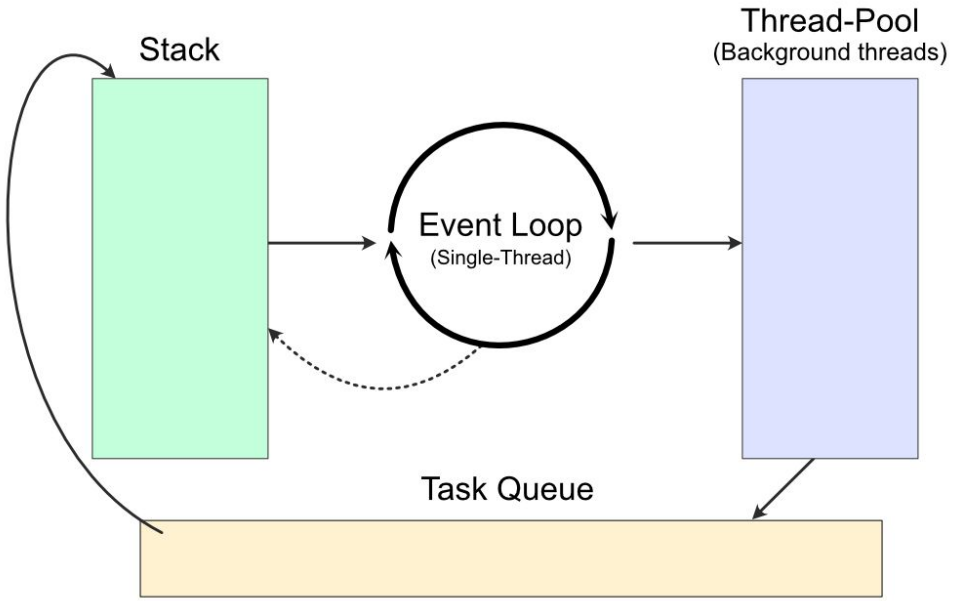
O “Stack” é o programa responsável pela organização das solicitações. Ele monta uma fila de ações que, no momento oportuno, serão encaminhadas organizadamente ao processador.



Event-loop (operação curta)

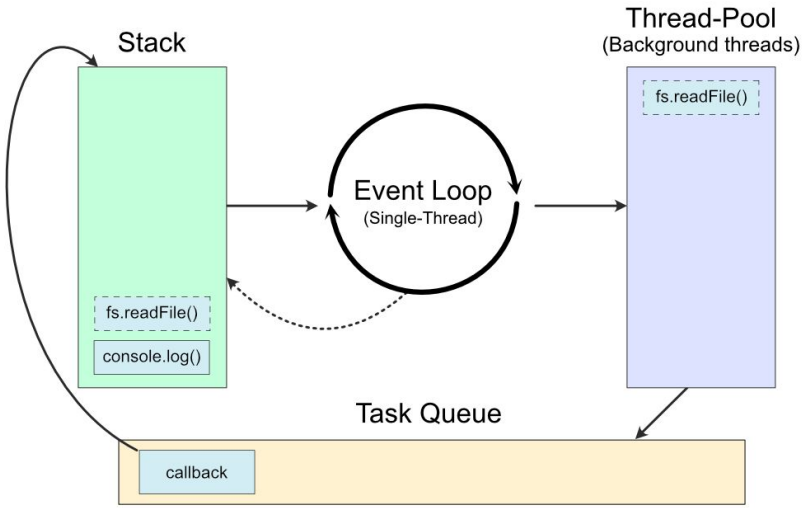
Bem, vimos que quando um usuário faz uma solicitação (evento) esta fica empilhada em um “local” chamado “Stack”.

O Event-Loop fica responsável por monitorar a Stack em busca de eventos a serem processados, quando um evento é encontrado e este não representa uma “operação longa” o mesmo é imediatamente executado e o Event-Loop liberado para prosseguir com a execução do próximo evento da Stack.



Event-loop (operação longa)

Se tivermos uma “operação longa”, como por exemplo ler um arquivo em disco, realizar transações de rede e assim por diante, o Event-Loop irá despachar essa operação juntamente com seu callback (chamada de retorno) para o Background Thread, ou seja, neste momento a libuv ganhará a responsabilidade de gerenciar essa execução em uma thread separada do Event-Loop, liberando o mesmo para prosseguir com a execução dos eventos ainda presentes na Stack. Quando essa tarefa que está rodando em uma thread separada for concluída, a sua função de callback será adicionado a Task Queue, que por sua vez aguardará o Event-Loop terminar sua tarefa (lembra, ele é Single-Thread, executa uma coisa por vez) para entrar na Stack e ter sua instrução executada.



Conclusão

Podemos então concluir, muito sucintamente, que o node.JS é um interpretador javascript com capacidade server side, com grande performance graças à sua arquitetura de tratamento de threads.

Por trabalhar com uma linguagem bastante conhecida e de fácil entendimento tem ganhado cada vez mais simpatizantes no meio dos desenvolvedores. E, além de tudo isso, muito provavelmente em breve o node.JS poderá escolher o melhor “engine” para o seu desenvolvimento, o V8 do Google ou o Chakra da Microsoft, que abrem mais portas para grandes inovações.

Outras Informações

Para saber mais acesse os links:

A coleção de guias e artigos oficiais: https://nodejs.org/en/docs/guides

A documentação de referência das APIs nativas: https://nodejs.org/api

Stack Overflow, este é um espaço (fórum/comunidade) onde você poderá achar soluções para os problemas que irá encontrar durante a longa jornada: http://pt.stackoverflow.com

https://nodejs.dev/the-nodejs-event-loop

https://dev.to/khaosdoctor/node-js-por-baixo-dos-panos-3-um-mergulho-no-event-loop-38l9 e
https://imasters.com.br/front-end/node-js-o-que-e-esse-event-loop-afinal

Obrigado

Profº Celso Henrique Masotti