iPictionary

Guilherme Severo
Claudio Busatto

Como foi feito?

- node.js
  - Core!
- PHP/HTML
  - geração das páginas para o usuário
- socket.io
  - websockets!
- express
  - framework para node

node.js

- Criado por Ryan Dahl em 2009
- Server-side javascript
- Plataforma para desenvolvimento de aplicações de rede altamente escaláveis
- Baseada em *Event Loop* (*epoll*) com a *libev*.
  - biblioteca que abstrai as chamadas de sistemas e funciona como loop de eventos
- Construída sobre a Javascript Engine V8 (Google)

  node.js project:
  - *"To provide a purely evented, non-blocking infrastructure to script highly concurrent programs."*

node.js

Tradicionalmente

- Servidores web com Apache
- Uma thread para cada conexão
- Requisição de E/S bloqueante

Problema

- Uma thread usando 2MB de RAM
- 2MB x 3.000 conexões = 6GB!!
- C10K Problem!

Novo Paradigma

- Requisições de E/S não-bloqueantes
- Uma única thread
- Multiplexação.

node.js

Tradicionalmente, temos códigos como

var result = db.query("select..");
// use result

Resultado:

i. Bloqueia esperando o retorno da consulta.
ii. Ociosidade!
iii. MUITAS threads bloqueadas esperando E/S.

node.js

Usando o **node**, teremos códigos como

```
db.query("select..", function (result) {
    // use result
});
```

Resultado:

i. Permite que o programa retorne o EventLoop imediatamente.
ii. Nenhuma recurso de máquina é necessário.

node.js

- Se é melhor, porque não é o padrão?
  - Código é considerado "complicado".
  - Single threaded event loops precisa de E/S não bloqueante.
    - A maioria das libs não são.
      - *libmysql_cliente* não fornece suporte a consultas assíncronas, por exemplo.

node.js

- SOs possuem métodos de chamadas assincronas
  - *select(), poll(), epoll() e kqueue().*
  - Esses métodos funcionam com um loop de eventos que fica aguardando chamadas (eventos).
    - Em cada chamada é registrado um callback, dessa forma outras chamadas podem ser feitas sem bloquear o processo.
    - O loop de eventos reage a chegada de dados e chama os callbacks.
    - Programação orientada a eventos!

node.js

- *epoll*
  - substituto do select()
  - acessa um número muito grande de descritores de arquivo.
    - várias chamadas feitas em um único processo.
      - Economia de memória e ciclos de CPU.
      - Aumento de performance.
      - Uma única thread muitas => muitas conexões simultâneas.
        - NGINX

- Solução para C10K.
  - Agora nosso problema é C100K, C500K.

node.js

- EventLoop fica recepcionando os eventos (Reactor Pattern)
  - assim que recebido o evento, o mesmo é direcionado para um Thread-Pool que executará as tarefas em background liberando assim o event loop
  - Assim que as tarefas vão sendo concluídas, o Node.js acionará o callback que foi passado no início da execução do evento.

node.js

- O node entra no event loop após executar o script de entrada.
- Ele finaliza o event loop quando não há mais callbacks para executar.
  - Este é o mesmo comportamento do navegador web.
- O event loop é transparente ao usuário

node.js

Objeto **process**

"*Object emits an event when it receives a signal. Like in the DOM, you need only add a listener to catch them."*

process.pid,

process.ARGV

process.ENV

process.cwd()

process.memoryUsage()

Todos objetos que emitem eventos são instâncias de **process**.EventEmitter.

socket.io

"Busca criar aplicativos de tempo real funcionais em todos os browsers e dispositivos móveis, "eliminando" as diferenças entre os diferentes mecanismos de transporte. "

- Permite emitir e receber eventos customizados. Além dos tradicionais "connect", "message" e "disconnect"

socket.io

- Permite a execução de um callback quando um cliente confirma o recebimento de uma mensagem
  - Simplesmente passar uma função com o último parâmetro *.send* or *.emit*.
  - Quando usado o '.emit', a confirmação é feita pelo programador, logo podemos passar dados junto.

socket.io

- Para enviar uma mensagem em *broadcast* só precisamos ativar uma flag nas chamadas dos métodos "emit" and "send".

- Broadcast envia a mensagem a todos, exceto para o socket que inicia.

Referências

Uma visão rápida sobre NodeJS

http://www.slideshare.net/rafaels88/uma-viso-rpida-sobre-nodejs#btnNext

NodeJS

http://www.slideshare.net/thiago_alima/node-slide#btnNext

http://s3.amazonaws.com/four.livejournal/20091117/jsconf.pdf

http://nodejs.org/about/

Entendendo NodeJS

http://studiosecret.com.br/blog/arquitetura/entendendo-nodejs

The C10K Problem

http://www.kegel.com/c10k.html

Scalable Network Programming

http://bulk.fefe.de/scalable-networking.pdf