FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Arquiteturas Disruptivas e Big Data

PROF. ANTONIO SELVATICI

Versão 3– Fevereiro/2017

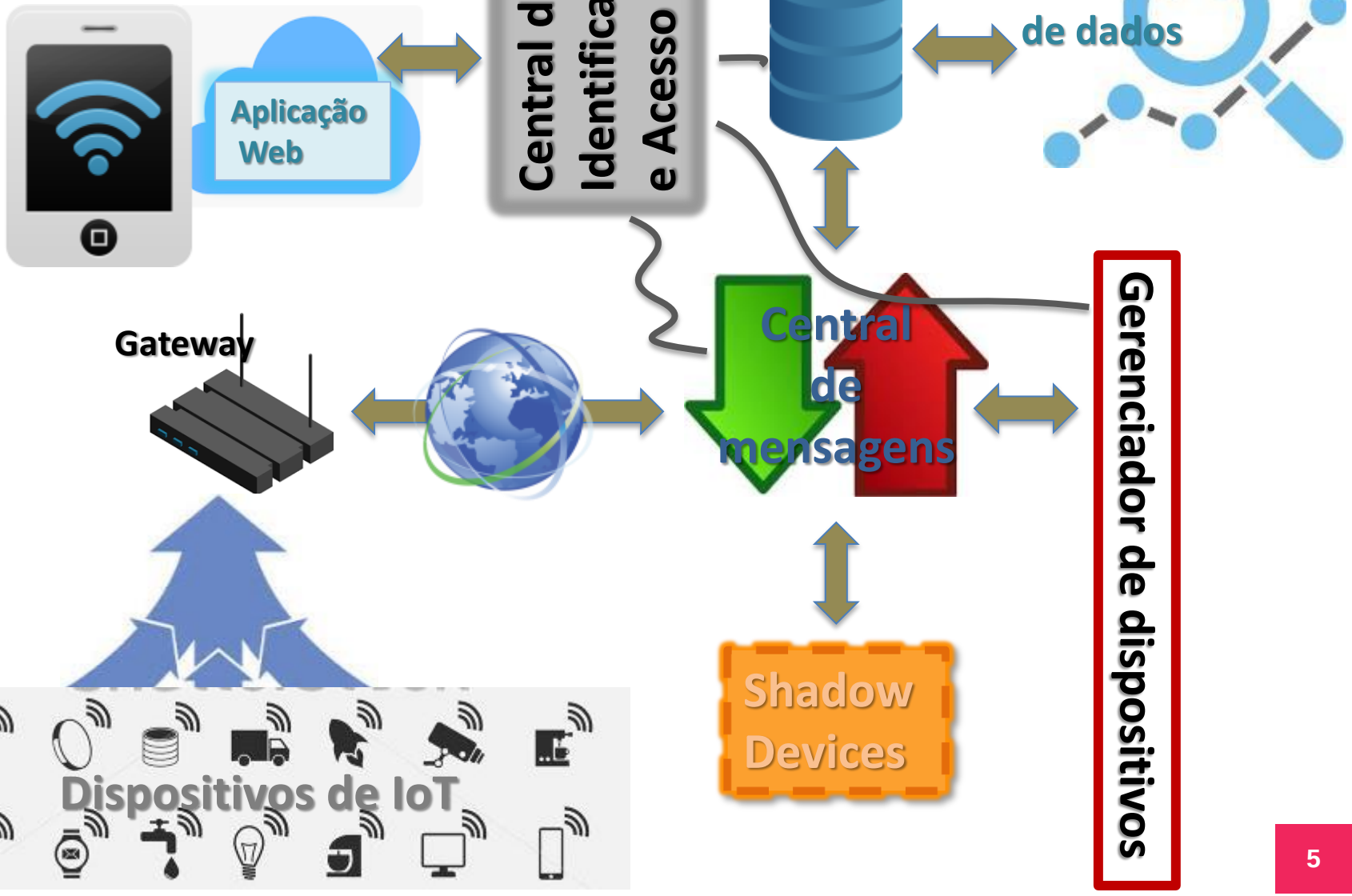
SHORT BIO

É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos. Desenvolveu projetos para Avibrás, IPT e Systax.

PROF. ANTONIO SELVATICI
profantonio.selvatici@fiap.com.br

INTERNET DAS COISAS

Arquitetura proposta



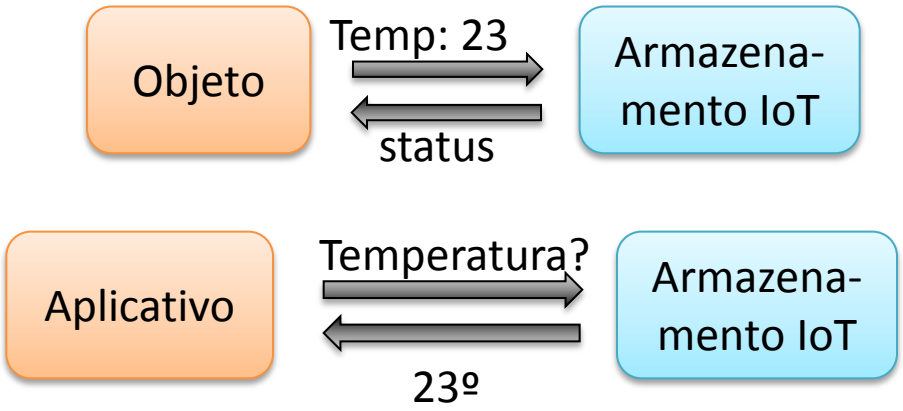
Gateway

- Realiza a conversão de protocolo entre os dispositivos de IoT e a central de distribuição de mensagens
- O uso do IPv6 pelos dispositivos facilita a resolução do endereçamento do dispositivo, mas não é suficiente para resolver as mensagens específicas da aplicação
- Gerenciamento de múltiplos protocolos, especialmente com LAN's, PAN's e HAN's : Zigbee, Bluetooth, Wi-Fi, Thread/6LoWPAN, etc.
- Serviço de dados em redes WAN: uso gateways compartilhados, podendo ou não ser pagos
  - Dados móveis (GPRS, 2G, 3G, HDSP, LTE, 5G ...)
  - LoRaWAN, SigFox

Gateway IoT

Integra objetos à internet, convertendo protocolos e servindo de ponte para serviços na rede

- Como se conectar a serviços disponíveis na internet, como Webservices?
- Arquitetura cliente/servidor: um programa cliente faz requisições remotas a um programa servidor que "escuta" em um endereço IP e uma porta, por exemplo:



- A mensagem enviada obedece a um protocolo de mensagens da camada de aplicação.

Transmissão de conteúdo na internet

- Os servidores da rede podem se comunicar com os dispositivos ou aplicações conectadas através de diversos protocolos de mensagens que funcionam bem para comandos simples
  - **HTTP:** Hyper-Text transfer Protocol (cabeçalho mais complexo)
  - **MQTT:** MQ Telemetry Transport
  - **CoAP:** Constrained Application Protocol
- As mensagens da IoT em geral são formatadas de acordo com regras que permitem o uso de interpretadores padrão:
  - **XML:** campos do documento definidos através de markups
  - **HTML:** subconjunto do XML usado para páginas web
  - **JSON:** documento definido como um objeto JavaScript

Exemplo de requisição HTTP com conteúdo JSON

- Requisição
  - `POST /request HTTP/1.1`
  - `Accept: application/jsonrequest`
  - `Content-Length: 72`
  - `Content-Type: application/jsonrequest`
  - `Host: json.penzance.org`
  - `{"user":"doctoravatar@penzance.com","forecast":7,"t":"vlIj","zip":94089}`
- Resposta
  - `HTTP/1.1 200 OK`
  - `Content-Type: application/jsonrequest`
  - `Content-Length: 15`
  - `{"status":true}`

Retirado de: `http://www.json.org/JSONRequest.html`

Como construir gateways para nossas aplicações de IoT?

- Os gateways devem ter duas pontas:
  - A ponta da rede de objetos, onde há a comunicação com o Arduino ou outros dispositivos através da porta serial ou módulos de comunicação sem fio
  - A ponta da rede de transmissão, onde mensagens são trocadas através da internet
  - Construiremos o gateway usando Node.js e Node-Red
- **Node.js**
  - Ambiente de execução em tempo real para a linguagem JavaScript, com foco na execução de programas no lado do servidor
  - Uso do motor Javascript V8 do Chrome, desenvolvido em C++, em conjunto com uma biblioteca de tratamento de eventos
  - Possui um conjunto de extensões elaboradas e melhoradas por contribuintes do mundo todo
- **Papel na IoT:** prover, de modo simples e eficiente, com alta disponibilidade, serviços demandando pouco esforço computacional
  - A simplicidade do servidor permite seu uso em dispositivos tipo Mini-PC mais simples, como Raspberry Pi e Beagle Bone

Nosso primeiro serviço WEB

- Criar pasta `C:/Users/rmXXXX/node`
- Criar arquivo `hello.js` com Notepad++

```
var http = require('http');
http.createServer(
  function (req, res) {
    res.writeHead(200, {'Content-Type':
'text/plain'});
    res.end('Hello World\n');
  }).listen(8080);
console.log('Server running at
http://localhost:8080/');
```

- Executar: `node hello`

Diferenças de um servidor tradicional

- Não é focado em um único tipo de serviço, ou tipos específicos de serviço (HTTP, banco de dados, FTP, etc.)
- Servidores tradicionais usam uma thread para cada requisição
  - Servidores são projetados para aplicações pesadas
  - Threads ocupam muita memória e outros recursos do sistema
  - 2 MB por thread → 1GB = 500 requisições simultâneas, com overhead do load balancer
- Uso de processamento de eventos assíncronos acionados pelas notificações do sistema operacional (signals e file descriptors)
  - Cada requisição ao servidor gera um evento
  - A mesma thread processa vários eventos
  - Capaz de processar milhares de requisições por segundo com pouco uso de memória

Tabela comparativa com outras linguagens de programação

Linguagem	Desempenho de execução	Requisito de Memória	Abstração de código	Tamanho do bytecode
Assembly	Muito Alto	M Baixo	Baixo	M Baixo
C	Muito Alto	Baixo	Médio	Baixo
C++	Muito Alto	Baixo	Médio/Alto	Depende*
Java	Alto	Alto	Alto	Médio/Alto
Python	Baixo	Alto	Alto	Baixo
Node.js	**Médio**	**Médio**	**Alto**	**Baixo**

Baseado em http://benchmarksgame.alioth.debian.org/

*Usar a STL aumenta muito o tamanho do código

Relembrando: programação JavaScript

- Declaração de variáveis:
  - `var <nome> = <valor>;`
  - Exemplo: `var i = 30;`
- Declaração de 'closures' (funções como variáveis)
  - `var <fun_name> = function(arg1, arg2,..., argN) {<code>}`
  - Exemplo: `var soma = function(a, b) { return a + b; }`
- Declaração de strings:
  - `var s = 'texto';`
  - `var s = "texto1" + 'texto2';`
- Declaração de objetos:
  - `var myobject = { field1:1, field2:['a','b','c']};`
  - Retornando um objeto JSON:
    - `console.log( JSON.stringfy(myobject) );`

- Node.js traz diversas APIs que proporcionam funcionalidades diferentes.

Entendendo o exemplo

```
//importando objeto com funcionalidades
//de servidor HTTP
var http = require('http');
//criação do servidor
http.createServer(
    //Passando função como argmento; ela trata
    //a requisição, preenchendo uma resposta
    function (req, res) {
        //Escreve o cabeçalho HTTP, com código de retorno 200
        res.writeHead(200, {'Content-Type': 'text/plain'});
        //Envia o corpo da mensagem e finaliza a conexão
        res.end('Hello World\n');
    }).listen(8080); //Método listen inicia o serviço
//Escreve mensagem no console
console.log('Server running at http://localhost:8080/');
```

Perguntas sobre o exemplo

- Qual é o protocolo de rede que o serviço usa?
- Qual é o protocolo de aplicação?
- Qual é o formato de dados?
- Qual é a linha de código que cria o serviço?
- Qual é a linha de código que inicia o serviço?
- Quando a função `function(req, res){...}` é executada?

Programação assíncrona em Node.JS

- Programar Node.JS *is all about* processar eventos de forma assíncrona
- Grande parte dos métodos de objetos que vamos usar recebem como argumento a função que irá tratar algum evento relacionado
- A dificuldade está mais em mudar o paradigma de programação, de síncrono (ou seja, código executado em sequência) para assíncrono, sem bloqueio do fluxo do programa
- Por exemplo, podemos agendar uma função para ser executado daqui a 5s, mas não é usual interrompermos o programa por 5s

Node-Red [nodered.org]

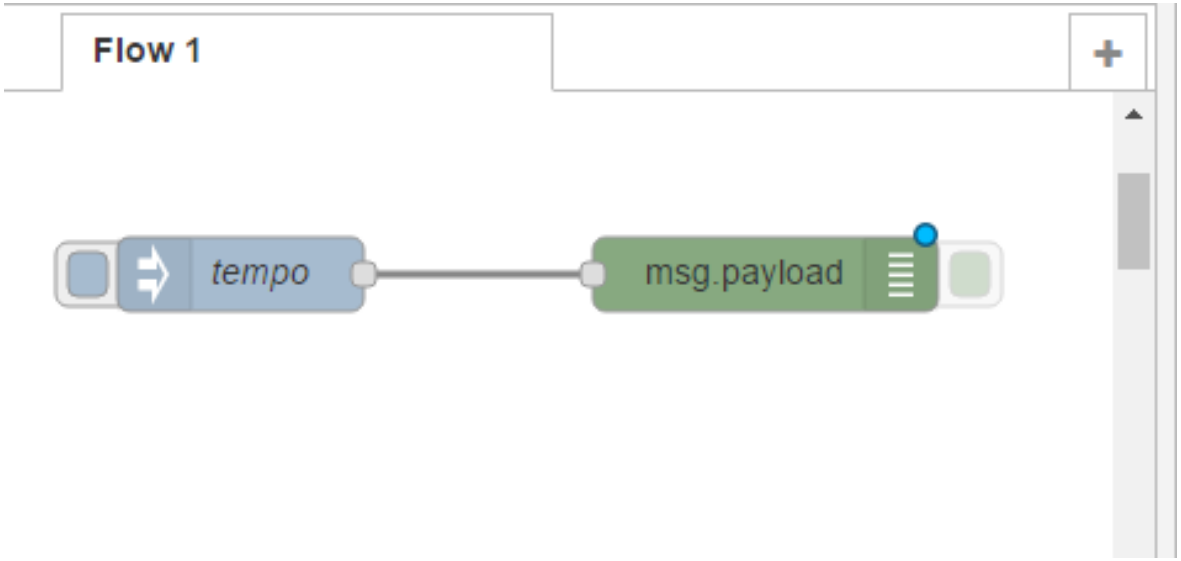
- Uma vez que a programação do Node.js é assíncrona, podemos pensar em programas em que todas as ações são acionadas por um evento gatilho.
  - Sendo assim, podemos pensar na programação do Node.js como fluxos de dados que iniciam a partir de algum evento, como o disparo de um temporizador, a requisição de um cliente de webservice ou um dado vindo do Arduino
- O Node-Red é um serviço escrito para Node.js que provê uma ferramenta visual para editar fluxos de mensagens, vindas de diferentes fontes, podendo ser processadas e mandadas para diferentes destinos, como uma conta de e-mail ou do Twitter
  - A ferramenta para edição dos fluxos roda no próprio browser
  - É possível exportar e importar fluxos no formato JSON usando o menu de opções
- O Node-Red está disponível no IBM Bluemix e em outros provedores de Cloud Computing

Instalação do Node-Red

- O nome do pacote a ser instalado é `node-red`
  - `npm install -g --unsafe-perm node-red`
- Para executar o serviço, executamos:
  - `node-red` para uma instalação global, ou
  - `node node_modules\node-red\red.js`
- Para acessar o serviço, acessamos no browser:
  - http://localhost:1880
- Instalando o Node-Red no Android
  - https://nodered.org/docs/platforms/android

Primeiro fluxo (Flow)

- Inicialmente, ligar um nó de entrada do tipo “inject” a um nó do tipo “debug”, fazer um “Deploy” e acionar o injetor de dados
- Observar o resultado na tela de Debug



Tipos de nodes

- Há basicamente três tipos de nodes:
  - **Entrada**, que emitem mensagens a partir de eventos de entrada
  - **Processamento**, que convertem mensagens de entrada em mensagens de saída
  - **Saída**, que enviam a mensagem de entrada para ser consumida em algum lugar
- Dentre os nós de processamento, podemos também criar funções JavaScript genéricas que podem manipular os campos da mensagem como se desejar
- Exemplo: convertendo timestamp em data formatada:

```
function(msg) {
  msg.payload = new Date(msg.payload).toString();
  return msg;
}
```

- O node function usa apenas o corpo da função acima

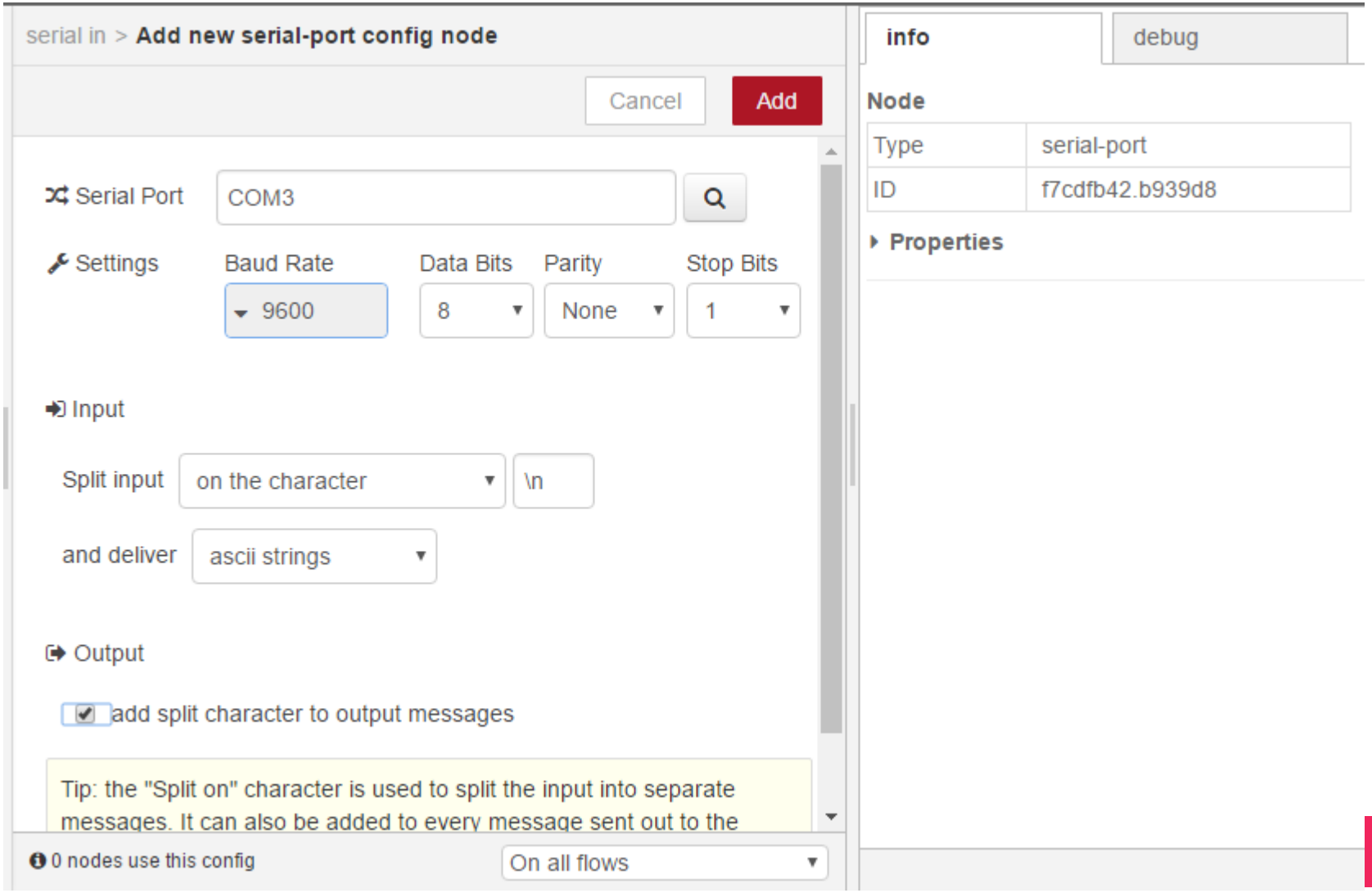
Capturando da e enviando para a porta serial

- Para usar as funcionalidades da porta serial, é necessário instalar o pacote do Node.js **node-red-node-serialport**
  - `npm install -g --unsafe-perm node-red-node-serialport`
- A captura da porta serial completa a leitura quando:
  - Recebe um caractere de terminação (default: '\n'), ou
  - Estoura o tempo limite (timeout) para novos caracteres, ou
  - Estoura o número máximo de caracteres que podem ser lidos
- O node de saída para a porta serial envia apenas o campo **msg.payload**, podendo ser configurado um caractere de terminação

Configuração do node serialport

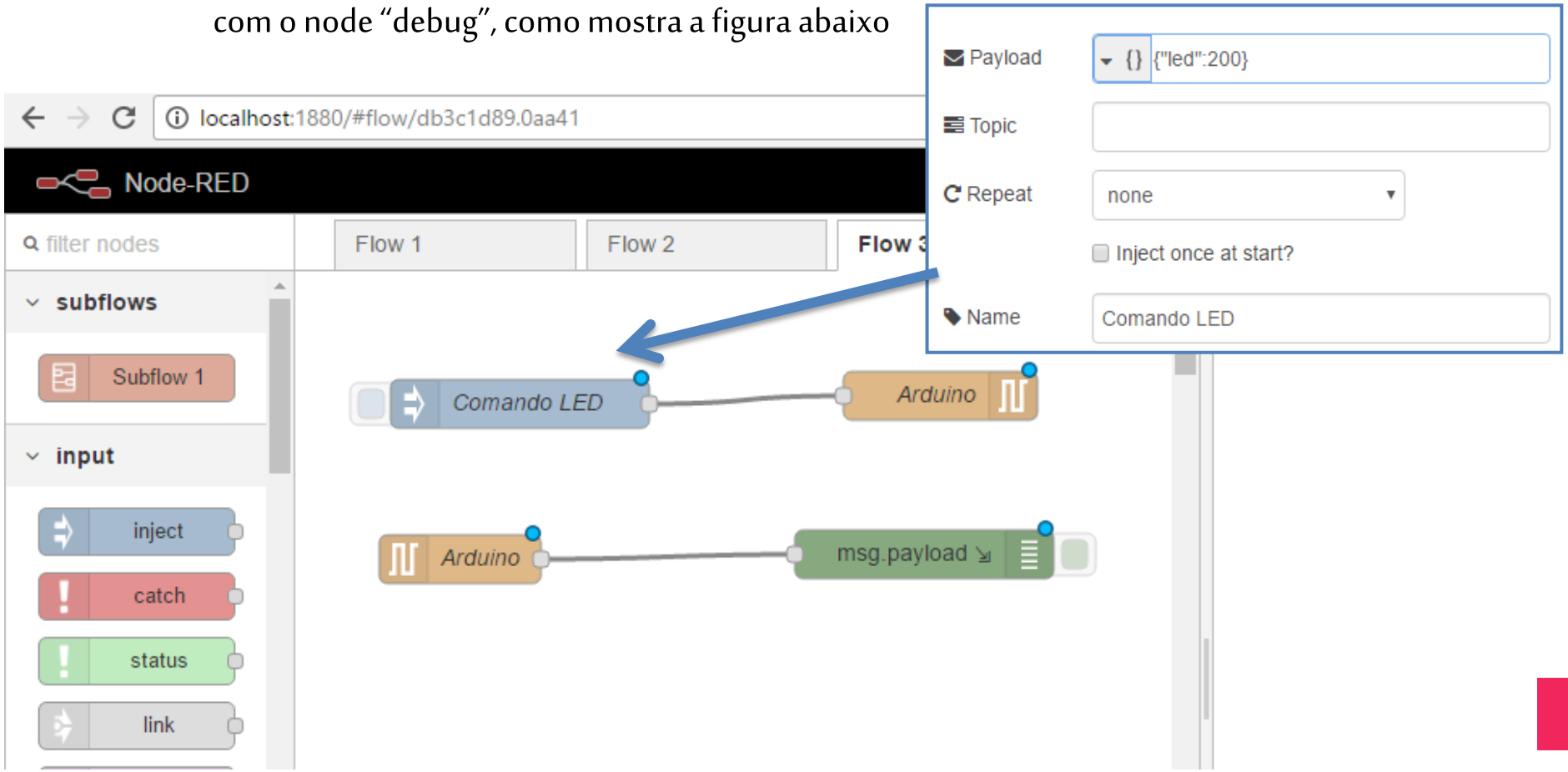
- Quando o Arduino é a fonte de dados, ele é representado no Node-Red como um node de **entrada** do tipo serial port.
  - Não confundir: a **saída** de dados do Arduino é usada como **entrada** de dados pelo Node-Red
- Na configuração da porta serial, fundamentalmente dois itens devem ser considerados:
  - O nome da porta serial deve ser o mesmo onde o Arduino está conectado
  - a velocidade de comunicação deve ser a mesma da do Arduino, geralmente 9600 baud
  - Além disso, caso o Arduino envie quebras de linha após cada dado enviado, podemos deixar que a porta serial dispare a cada vez que encontrar o caractere `\n`

Configuração da captura da porta serial



Trocando dados com o Arduino

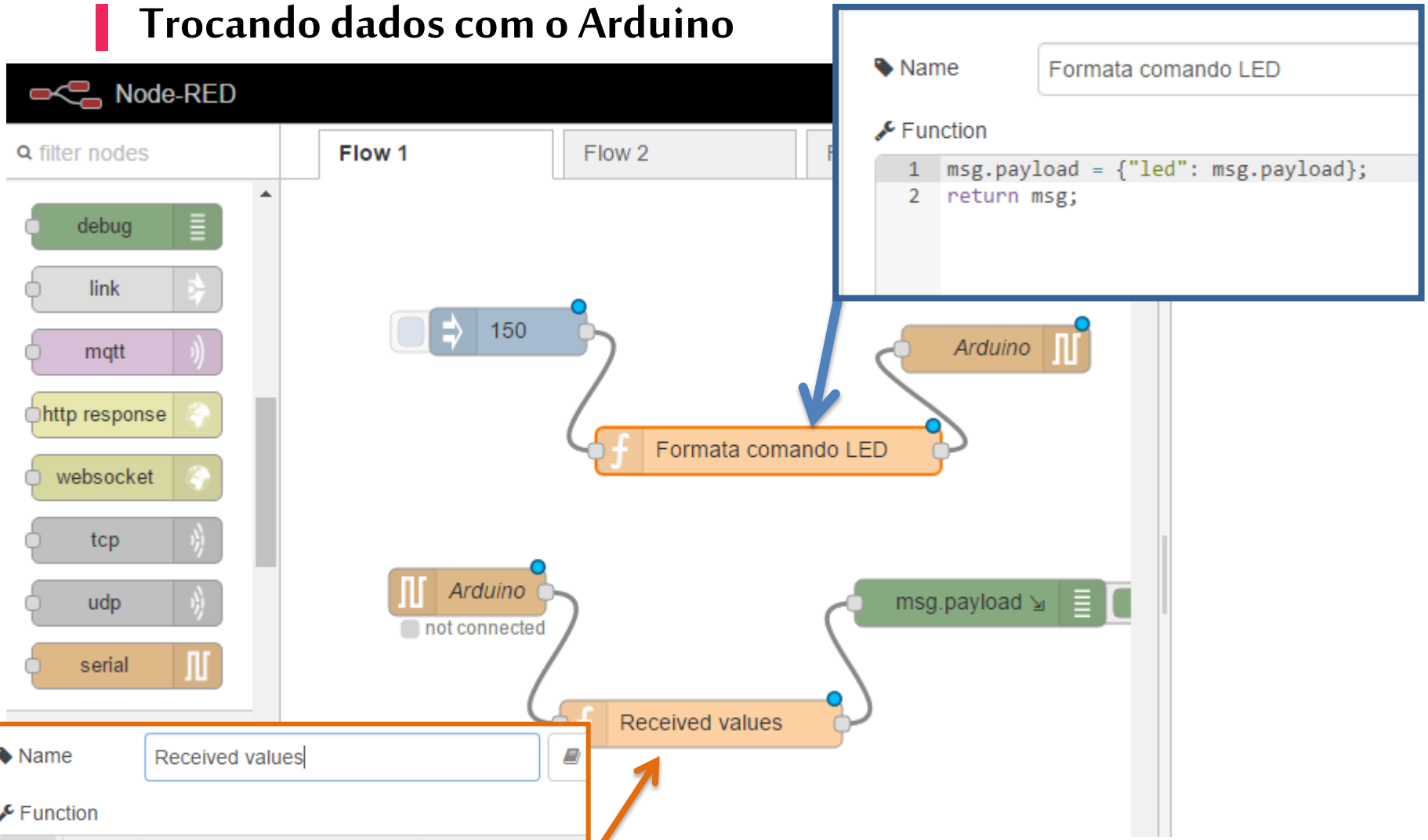
- A forma mais simples de comunicação do Node-Red com o Arduino é injetando diretamente os comandos JSON através do node "inject", e verificar diretamente a saída com o node "debug", como mostra a figura abaixo



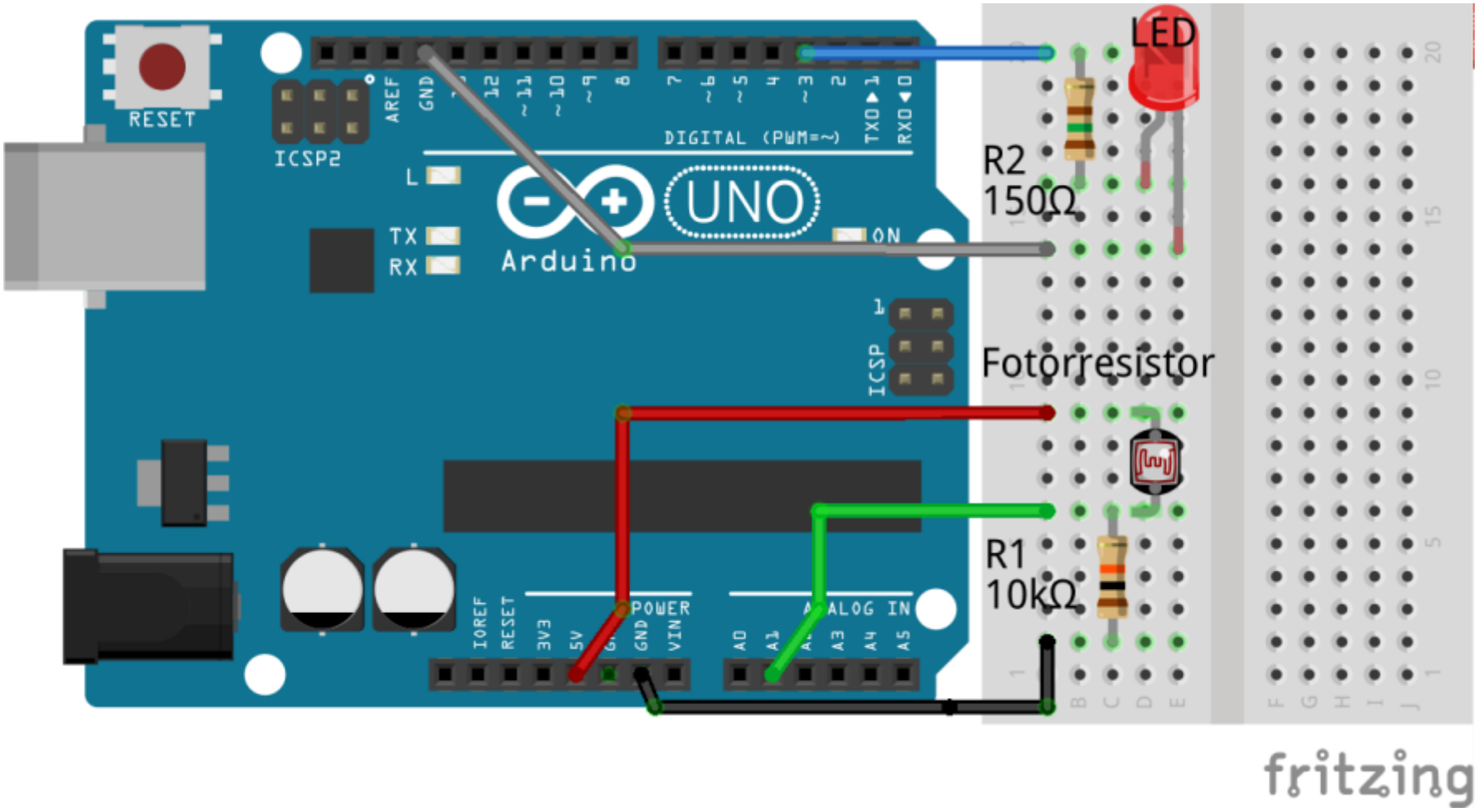
Codificando e decodificando JSON

- Por serem programados em Javascript, os nodes do tipo **function** podem facilmente criar objetos JSON ou decodificar JSON a partir de strings
- No exemplo a seguir, o node inject fornece apenas o valor a ser enviado ao led, enquanto a função "Formata comando LED" formata o comando a ser enviado ao LED do Arduino.
- Similarmente, a função "Parse luz" lê apenas o valor da luminosidade encapsulada no JSON vindo do Arduino

Trocando dados com o Arduino



Entrada e saída analógica



Lendo o JSON da porta Serial e mandando a luminosidade

```
#include <ArduinoJson.h>
const int LED = 3;
const int LUZ = A1;
const int TAMANHO = 200;
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(10); //1000ms é muito tempo
  pinMode(LED,OUTPUT);
}
void loop() {
  if (Serial.available() > 0) {
    //Lê o texto disponível na porta serial:
    char texto[TAMANHO];
    Serial.readBytesUntil('\n', texto, TAMANHO);
    //Grava o texto recebido como JSON
    StaticJsonBuffer<TAMANHO> jsonBuffer;
    JsonObject& json = jsonBuffer.parseObject(texto);
    if(json.success() && json.containsKey("led")) {
      analogWrite(LED, json["led"]);
    }
  }
  StaticJsonBuffer<TAMANHO> jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["luz"] = analogRead(LUZ);
  json.printTo(Serial); Serial.println();
  delay(1000);
}
```

REFERÊNCIAS

1. Joyent. **Node.js**. url: http://www.nodejs.org Acesso em 20/01/2016
2. Joyent. **Node.js API.** url: http://nodejs.org/api/
3. IBM Emerging Technologies. **Node-Red.** url: http://nodered.org

FIAP