Server REST con esp8266 e esp32: introduzione – Parte 1

DI RENZO MISCHIANTI · PUBBLICATO 16 MAGGIO 2020 · AGGIORNATO 27 SETTEMBRE 2023

Come creare un server REST con esp8266 o esp32: introduzione

Quando si parla di microservizi non si può che parlare di tecnologia REST.

Representational State Transfer (REST) è uno stile architetturale (di architettura software) per i sistemi distribuiti.
La separazione REST client-server degli interessi semplifica l'implementazione del componente, riduce la complessità della semantica del connettore, migliora l'efficacia dell'ottimizzazione delle prestazioni ed aumenta la scalabilità di componenti server puri. I vincoli di sistema a strati permettono di introdurre intermediari — proxy, gateway, e firewall — in vari punti della comunicazione senza cambiare le interfacce tra i componenti, consentendo loro di assistere nella traduzione della comunicazione o migliorare le prestazioni tramite cache condivisa di larga scala. REST consente la elaborazione intermedia vincolando i messaggi ad essere auto-descrittivi: l'interazione è priva di stato tra le richieste, i metodi di base ed i tipi di media sono utilizzati per indicare la semantica e scambiare informazioni e le risposte indicano esplicitamente la cachabilità. (cit. wiki)

Questa semplice architettura software è molto popolare per vari motivi ed è abbastanza semplice da implementare con il nostro microcontrollore esp32 o esp8266.

Inizializzazione

Possiamo usare la libreria ESP standard

```
#include <ESP8266WebServer.h>
```

Quindi dobbiamo configurare la porta e istanziare il server:

```
#define HTTP_REST_PORT 8080
ESP8266WebServer httpRestServer(HTTP_REST_PORT);
```

Al momento del setup aggiungeremo tutto il routing (nella prossima sezione spiegheremo meglio) e inizializzeremo il server:

```
restServerRouting();
```

```
httpRestServer.begin();
```

E nella funzione loop aggiungiamo l'handle:

```
httpRestServer.handleClient();
```

Routing

L'architettura RESTful fornisce un set standard di verbi:

Arduino esp8266 esp32 REST server schema

Verbi REST

	`GET` è il tipo di metodo di richiesta HTTP più semplice; quello che il browser usa ogni volta che cliccate su un link o inserite una URL nella barra degli

GET	indirizzi. Ordina al server di trasmettere al client le informazioni identificate nella URL. Le informazioni lato server non dovrebbero essere modificate in risultato ad una richiesta `GET`.
POST	`POST` viene utilizzato quando si desidera che il processo che avviene sul server possa essere ripetuto, in caso la richiesta `POST` venga ripetuta. Inoltre, le richieste `POST` potrebbero causare l'elaborazione del corpo come un subordinato della URL a cui vi riferite.
PUT	Una richiesta `PUT` viene usata quando si vuole creare o aggiornare la risorsa identificata dalla URL.
PATCH	Aggiorna tutte le rappresentazioni della risorsa o può creare la risorsa se non esiste, usando le istruzioni nel corpo della richiesta.
DELETE	Il `DELETE`, dovrebbe funzionare inversamente al `PUT`; dovrebbe essere usato quando volete cancellare la risorsa identificata dalla URL della richiesta.

Quando usi un verbo rispetto ad un altro la cosa che cambia è il trasferimento dati, il comportamento sarà comunque determinato dal tuo codice.

```
void getHelloWord() {
        server.send(200, "text/json", "{\"name\":
\"Hello world\"}");
}

void restServerRouting() {
    server.on("/", HTTP_GET, []() {
        server.send(200, F("text/html"),
            F("Welcome to the REST Web Server"));
    });
    server.on(F("/helloWorld"), HTTP_GET,
getHelloWord);
}
```

Qui sopra 2 operazioni in GET, la prima ha come endpoint `/` e ritorna ogni volta un `text/html` come mime type con "Welcome to the REST Web Server" come contenuto.

Ma che significa?

L'uso del verbo GET è solo il tipo di richiesta e il server REST lo utilizza per capire come gestire la richiesta.
Il mimetype viene utilizzato dal browser per comprendere il tipo di contenuto del risultato e capire come deve essere elaborato il risultato, in questo caso text/html viene analizzato dal browser come testo html e mostrato nella pagina.
Il contenuto è il risultato della richiesta.

Per l'esp32 devi modificare questi header

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
```

a

```
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>
```

Ecco un sottoinsieme di tipi mime:

Mimetype

Ecco il codice completo

```
/*
 *      Simple hello world Json REST response
  *  by Mischianti Renzo <https://mischianti.org>
 *
 *  https://mischianti.org/
 *
 */

#include "Arduino.h"
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
```

```
const char* ssid = "<your-ssid>";
const char* password = "<your-passwd>";

ESP8266WebServer server(80);

// Serving Hello world
void getHelloWord() {
        server.send(200, "text/json", "{\"name\":
\"Hello world\"}");
}

// Define routing
void restServerRouting() {
    server.on("/", HTTP_GET, []() {
        server.send(200, F("text/html"),
            F("Welcome to the REST Web Server"));
    });
    server.on(F("/helloWorld"), HTTP_GET,
getHelloWord);
}

// Manage not found URL
void handleNotFound() {
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ?
"GET" : "POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i = 0; i < server.args(); i++) {
    message += " " + server.argName(i) + ": " +
server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);
}

void setup(void) {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
```

```
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    // Activate mDNS this is used to be able to
  connect to the server
    // with local DNS hostmane esp8266.local
    if (MDNS.begin("esp8266")) {
      Serial.println("MDNS responder started");
    }

    // Set server routing
    restServerRouting();
    // Set not found response
    server.onNotFound(handleNotFound);
    // Start server
    server.begin();
    Serial.println("HTTP server started");
  }

  void loop(void) {
    server.handleClient();
  }
```

Dopo aver fatto l'upload del tuo sketch (Ecco la guida WeMos D1 mini (esp8266): caratteristiche e configurazione dell'Arduino IDE) devi identificare l'ip o il nome locale.

Puoi ottenere l'IP dall'output seriale.

```
Connected to ...
IP address: 192.168.1.122
MDNS responder started
HTTP server started
```

Quindi puoi inseriresul tuo browser direttamente l'URL REST in GET http://192.168.1.122/helloWorld per ricevere il messaggio di benvenuto. Il browser elabora tutte le chiamate come GET, quindi questo end-point funziona bene.

Puoi provare a chiamare il tuo dispositivo con il nome http://esp8266.local, questo nome è gestito dall'mDNS.

mDNS

> *Nelle reti di computer, il* ***protocollo DNS*** *(* ***mDNS*** *)* ***multicast*** *risolve i nomi host in indirizzi IP all'interno di piccole reti che non includono un server dei nomi locale. È un servizio a configurazione zero, che utilizza essenzialmente le stesse interfacce di programmazione, formati di pacchetti e semantica operativa del sistema DNS (unicast). Sebbene Stuart Cheshire abbia progettato mDNS come protocollo autonomo, può funzionare di concerto con i server DNS standard.*
> *Per funzionare sono necessari i servizi di stampa Bonjour per Windows o Avahi per Linux.*

Quando si chiama l'endpoint REST helloWorld, se si fa clic su `F12` nel browser e si passa a Networking e si seleziona la chiamata, è possibile vedere che è stato elaborato come formato text/json.

Grazie

1. Server REST su esp8266 o esp32: introduzione
2. Server REST su esp8266 o esp32: GET e formattazione JSON
3. Server REST su esp8266 o esp32: POST, PUT, PATCH, DELETE
4. Server REST su esp8266 o esp32: richieste CORS, OPTION e GET
5. Server REST su esp8266 o esp32: richieste CORS, OPTION e POST

Etichette: esp32 esp8266 REST API Wemos D1 mini

LASCIA UN COMMENTO

Login with your Social ID

I agree to my personal data being stored and used as per Privacy Policy

Commento *

Nome *

Email *

Sito web

Subscribe newsletter!

Invia commento