Concetti Fondamentali

Architettura di Rete con ESP32

L'ESP32 può operare in tre modalità principali:

- **STA (Station)**: Funziona come client in una rete WiFi
- **AP (Access Point)**: Crea una rete WiFi a cui altri dispositivi possono connettersi
- **Server + AP**: Combina le funzionalità di server e access point

```
[CL1] <---> [AP] <---> [CL2]
```

Componenti di Rete

- **SSID**: Nome della rete WiFi
- **IP**: Indirizzo che identifica un dispositivo nella rete
- **Porta**: Canale numerico utilizzato per il servizio (es. 80 per HTTP)
- **Socket**: Combinazione di IP + Porta

Librerie Principali

```
#include <WiFi.h>      // Per modalità STA e funzionalità server
#include <WiFiAP.h>    // Per modalità Access Point
#include <WebServer.h> // Per creare un server HTTP
```

Configurazione Base

Modalità Station (Client)

```
// Configurazione rete
const char* ssid = "NomeRete";
const char* password = "Password";

void setup() {
  Serial.begin(115200);

  // Configurazione come client
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  // Attesa connessione
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(500);
    Serial.print(".");
  }

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

Modalità Access Point

```
const char* ssid = "ESP32_AP";
const char* password = "Password123";

void setup() {
  Serial.begin(115200);

  // Creazione Access Point
  WiFi.softAP(ssid, password);

  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP); // Di default 192.168.4.1
}
```

Comunicazione Client-Server

Server Base

```
#include <WiFi.h>

const char* ssid = "NomeRete";
const char* password = "Password123";

// Creazione server sulla porta specificata
WiFiServer server(80);

void setup() {
  Serial.begin(115200);

  // Configurazione nome host
  WiFi.setHostname("ESP32Server");
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }

  // Avvio del server
  server.begin();

  Serial.println("Server started");
  Serial.println(WiFi.localIP());
}

void loop() {
  // Accetta connessioni client
  WiFiClient client = server.accept();

  if (client) {
    // Gestione del client connesso
    while (client.connected()) {
      if (client.available()) {
        // Lettura dei dati
        String request = client.readStringUntil('\r');
        // Elaborazione e risposta
      }
    }
    client.stop();
  }
}
```

Client Base

```
#include <WiFi.h>

const char* ssid = "NomeRete";
const char* password = "Password123";
const char* host = "192.168.1.1"; // IP del server
const int port = 80;              // Porta del server

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("WiFi connected");
  Serial.println(WiFi.localIP());
}
```

```
void loop() {
  WiFiClient client;

  if (client.connect(host, port)) {
    // Invio dati al server
    client.println("GET /path HTTP/1.1");
    client.println("Host: " + String(host));
    client.println("Connection: close");
    client.println();

    // Lettura risposta
    while (client.available()) {
      String line = client.readStringUntil('\r');
      Serial.print(line);
    }

    client.stop();
  }

  delay(5000);
}
```

Protocollo HTTP

Origine e Utilizzo

HTTP è un protocollo di comunicazione basato su testo che consente la trasmissione di risorse attraverso una rete, indipendentemente dal loro formato. Può essere utilizzato per trasmettere documenti HTML, file binari o altre strutture dati più complesse.

In HTTP 1.1 è possibile utilizzare la stessa connessione per inviare più richieste e ricevere più risposte, il che consente di ridurre il numero di connessioni necessarie e migliorare le prestazioni complessive.

Struttura Base HTTP

1. **Richiesta HTTP** (client → server):

```
METODO /risorsa HTTP/versione
Host: www.example.com
User-Agent: Mozilla/5.0 ...
Accept-Language: en-us
...

[corpo del messaggio]
```

2. **Risposta HTTP** (server → client):

```
HTTP/versione CODICE_STATO DESCRIZIONE
Content-Type: text/html
Content-Length: 438
...

[corpo del messaggio]
```

Metodi HTTP Principali

- **GET**: Richiede una risorsa (esempio: pagina web, immagine)
- **POST**: Invia dati al server (esempio: form di registrazione)
- **PUT**: Aggiorna una risorsa esistente
- **DELETE**: Elimina una risorsa

Effettuare Richieste HTTP GET con ESP32

Per inviare richieste HTTP GET da un ESP32, è necessario utilizzare la libreria `HTTPClient`:

```
#include <WiFi.h>
#include <HTTPClient.h>

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("WiFi connected");
}

void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // Specificare l'URL
    String serverPath = "http://example.com/data?param=value";
    http.begin(serverPath.c_str());

    // Invio della richiesta GET
```

```
    int httpResponseCode = http.GET();

    if (httpResponseCode > 0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      String payload = http.getString();
      Serial.println(payload);
    } else {
      Serial.print("Error code: ");
      Serial.println(httpResponseCode);
    }

    // Liberare le risorse
    http.end();
  }

  delay(5000);  // Effettuare richieste ogni 5 secondi
}
```

WebServer HTTP con ESP32

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

const char* ssid = "NomeRete";
const char* password = "Password123";

WebServer server(80);

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Definizione gestori per vari endpoint
  server.on("/", HTTP_GET, handleRoot);
  server.on("/led/on", HTTP_GET, handleLedOn);
  server.on("/led/off", HTTP_GET, handleLedOff);
  server.onNotFound(handleNotFound);

  // Avvio server
  server.begin();
  Serial.println("HTTP server started");
}
```

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

// Gestori richieste
void handleRoot() {
  server.send(200, "text/html", "Pagina principale");
}

void handleLedOn() {
  // Codice per accendere il LED
  server.send(200, "text/plain", "LED acceso");
}

void handleLedOff() {
  // Codice per spegnere il LED
  server.send(200, "text/plain", "LED spento");
}

void handleNotFound() {
  server.send(404, "text/plain", "Risorsa non trovata");
}
```

Comunicazione Client-Server con ESP32

Vantaggi e Svantaggi del Modello Client-Server

Vantaggi

- **Sicurezza centralizzata**: L'architettura centralizzata offre una maggiore protezione dei dati poiché tutte le richieste passano attraverso il server, permettendo controlli di accesso centralizzati.
- **Indipendenza dalla piattaforma**: I protocolli client-server sono indipendenti dalla piattaforma, consentendo l'uso di dispositivi con sistemi operativi diversi.

Svantaggi

- **Sovraccarico**: Il modello è vulnerabile a problemi di sovraccarico quando troppe richieste simultanee vengono inviate al server.
- **Punto singolo di guasto**: Se il server si guasta, tutti i client perdono l'accesso ai servizi.

Protocolli a Basso Livello

- **TCP**: Protocollo orientato alla connessione che garantisce la consegna affidabile dei dati.
- **UDP**: Protocollo senza connessione più veloce ma meno affidabile di TCP.

Esercizi Pratici

Esercizio 1: Server Echo

Crea un server che restituisca al client esattamente il messaggio che ha ricevuto.

```
#include <WiFi.h>

const char* ssid = "NomeRete";
const char* password = "Password123";
WiFiServer server(3000);

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  server.begin();
  Serial.println("Server started");
  Serial.println(WiFi.localIP());
}

void loop() {
  WiFiClient client = server.available();

  if (client) {
    while (client.connected()) {
      if (client.available()) {
        String data = client.readStringUntil('\n');
        Serial.println("Received: " + data);
        client.println(data);
      }
    }
    client.stop();
  }
}
```

Esercizio 2: Client con Autenticazione

Crea un client che invii un codice di autenticazione al server e visualizzi se l'accesso è stato autorizzato.

```
#include <WiFi.h>

const char* ssid = "NomeRete";
const char* password = "Password123";
const char* host = "192.168.1.1";
const int port = 3459;
const int authCode = 123456;

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("Connected");
  Serial.println(WiFi.localIP());
}

void loop() {
  WiFiClient client;

  if (client.connect(host, port)) {
    Serial.println("Connected to server");
    client.println(authCode);

    delay(1000);

    while (client.available()) {
      String response = client.readStringUntil('\n');
      if (response.toInt() == 1337) {
        Serial.println("Authentication successful");
      } else {
        Serial.println("Authentication failed");
      }
    }

    client.stop();
  } else {
    Serial.println("Connection failed");
  }
```

```
  delay(5000);
}
```

Esercizio 3: Server HTTP Semplice con LED

Crea un server HTTP che permetta di controllare un LED tramite browser.

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

const char* ssid = "NomeRete";
const char* password = "Password123";
const int ledPin = 2;

WebServer server(80);

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  server.on("/", handleRoot);
  server.on("/on", handleOn);
  server.on("/off", handleOff);

  server.begin();
  Serial.println("HTTP server started");
  Serial.println(WiFi.localIP());
}

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

void handleRoot() {
  String html = "<html><body>";
  html += "<h1>ESP32 LED Control</h1>";
  html += "<p><a href=\"/on\"><button>ON</button></a></p>";
  html += "<p><a href=\"/off\"><button>OFF</button></a></p>";
  html += "</body></html>";

  server.send(200, "text/html", html);
}
```

```
void handleOn() {
  digitalWrite(ledPin, HIGH);
  server.sendHeader("Location", "/");
  server.send(303);
}

void handleOff() {
  digitalWrite(ledPin, LOW);
  server.sendHeader("Location", "/");
  server.send(303);
}
```

Esercizio 4: Temperatura e Server HTTP

Crea un server HTTP che mostri la temperatura rilevata da un sensore DHT11/DHT22.

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

const char* ssid = "NomeRete";
const char* password = "Password123";
#define DHTPIN 4
#define DHTTYPE DHT11  // DHT11 o DHT22

DHT dht(DHTPIN, DHTTYPE);
WebServer server(80);

void setup() {
  Serial.begin(115200);
  dht.begin();

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  server.on("/", handleRoot);
  server.on("/temperature", handleTemp);

  server.begin();
  Serial.println("HTTP server started");
  Serial.println(WiFi.localIP());
}

void loop() {
```

```
  server.handleClient();
}

void handleRoot() {
  String html = "<html><body>";
  html += "<h1>ESP32 Temperature Server</h1>";
  html += "<p><a href=\"/temperature\">View Temperature</a></p>";
  html += "</body></html>";

  server.send(200, "text/html", html);
}

void handleTemp() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();

  String html = "<html><body>";
  html += "<h1>Sensor Data</h1>";
  html += "<p>Temperature: " + String(temp) + " °C</p>";
  html += "<p>Humidity: " + String(hum) + " %</p>";
  html += "<p><a href=\"/\">Back</a></p>";
  html += "</body></html>";

  server.send(200, "text/html", html);
}
```

Esercizio 5: AP e Controllo Servo

Crea un ESP32 che funzioni come Access Point e server HTTP per controllare un servo motore.

```
#include <WiFi.h>
#include <WebServer.h>
#include <ESP32Servo.h>

const char* ssid = "ESP32_ServoControl";
const char* password = "Password123";

#define SERVO_PIN 13
Servo myServo;
WebServer server(80);

void setup() {
  Serial.begin(115200);

  // Configurazione servo
  myServo.attach(SERVO_PIN);
  myServo.write(90);  // Posizione iniziale
```

```
  // Configurazione AP
  WiFi.softAP(ssid, password);
  IPAddress myIP = WiFi.softAPIP();
  Serial.println("AP IP address: ");
  Serial.println(myIP);

  // Configurazione server
  server.on("/", handleRoot);
  server.on("/setangle", HTTP_GET, handleSetAngle);

  server.begin();
  Serial.println("HTTP server started");
}

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

void handleRoot() {
  String html = "<html><body>";
  html += "<h1>ESP32 Servo Control</h1>";
  html += "<p>Angle: <span id='angle'>90</span></p>";
  html += "<input type='range' min='0' max='180' value='90' id='servo'
oninput='updateServo(this.value)'>";
  html += "<script>";
  html += "function updateServo(angle) {";
  html += "  document.getElementById('angle').innerHTML = angle;";
  html += "  fetch('/setangle?value=' + angle);";
  html += "}";
  html += "</script>";
  html += "</body></html>";

  server.send(200, "text/html", html);
}

void handleSetAngle() {
  String angleValue = server.arg("value");
  int angle = angleValue.toInt();

  if (angle >= 0 && angle <= 180) {
    myServo.write(angle);
    server.send(200, "text/plain", "OK");
  } else {
    server.send(400, "text/plain", "Invalid angle");
  }
}
```

Esercizio 6: Implementazione di Richieste HTTP GET

Crea un ESP32 che invii periodicamente richieste HTTP GET a un server web e stampi la risposta.

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* ssid = "NomeRete";
const char* password = "Password123";
String serverName = "http://example.com/data";  // Sostituisci con il tuo
server

unsigned long lastTime = 0;
const unsigned long timerDelay = 10000;  // 10 secondi

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  Serial.print("Connessione alla rete ");
  Serial.println(ssid);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connesso");
  Serial.println("Indirizzo IP: " + WiFi.localIP().toString());
}

void loop() {
  // Invia richiesta HTTP GET ogni 10 secondi
  if ((millis() - lastTime) > timerDelay) {
    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;

      // Configura la richiesta con parametri
      String serverPath = serverName + "?sensor=temperatura&value=24.5";
      http.begin(serverPath.c_str());

      // Invio richiesta HTTP GET
      int httpResponseCode = http.GET();

      if (httpResponseCode > 0) {
        Serial.print("Codice risposta HTTP: ");
        Serial.println(httpResponseCode);
        String payload = http.getString();
        Serial.println(payload);
```

```
      } else {
        Serial.print("Errore: ");
        Serial.println(httpResponseCode);
      }

      // Libera risorse
      http.end();
    } else {
      Serial.println("WiFi disconnesso");
    }

    lastTime = millis();
  }
}
```

Esercizio 7: Server con Autenticazione

Implementa un server web con autenticazione HTTP basic per proteggere l'accesso.

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

const char* ssid = "NomeRete";
const char* password = "Password123";

// Credenziali per l'autenticazione HTTP
const char* www_username = "admin";
const char* www_password = "esp32admin";

// Porta LED
const int ledPin = 2;

WebServer server(80);

void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connesso");
```

```
  Serial.println("Indirizzo IP: " + WiFi.localIP().toString());

  server.on("/", []() {
    if (!server.authenticate(www_username, www_password)) {
      return server.requestAuthentication();
    }
    server.send(200, "text/html", getPage());
  });

  server.on("/led/on", []() {
    if (!server.authenticate(www_username, www_password)) {
      return server.requestAuthentication();
    }
    digitalWrite(ledPin, HIGH);
    server.sendHeader("Location", "/");
    server.send(303);
  });

  server.on("/led/off", []() {
    if (!server.authenticate(www_username, www_password)) {
      return server.requestAuthentication();
    }
    digitalWrite(ledPin, LOW);
    server.sendHeader("Location", "/");
    server.send(303);
  });

  server.begin();
}

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

String getPage() {
  String state = (digitalRead(ledPin)) ? "ON" : "OFF";
  String html = "<!DOCTYPE html><html><head><title>ESP32 Control</title>";
  html += "<meta name='viewport' content='width=device-width, initial-
scale=1'>";
  html += "<style>body{font-family:Arial;margin:0;padding:20px;text-
align:center;}";
  html += ".button{display:inline-block;padding:15px 30px;margin:10px;";
  html += "background-color:#4CAF50;color:white;font-
size:16px;border:none;";
  html += "border-radius:5px;text-decoration:none;cursor:pointer;}";
  html += ".button.off{background-color:#808080;}";
  html += "h1{color:#333;}p{font-size:18px;}</style></head>";
  html += "<body><h1>ESP32 Web Server</h1>";
  html += "<p>Stato LED: " + state + "</p>";
  if (state == "OFF") {
```

```
    html += "<a href='/led/on'><button class='button'>ON</button></a>";
  } else {
    html += "<a href='/led/off'><button class='button off'>OFF</button>
</a>";
  }
  html += "</body></html>";
  return html;
}
```