

Introdução ao Servidor Web no ESP32 e Protocolos de IoT

Objetivo da Aula:

- Compreender o funcionamento do ESP32 como servidor web.
- Explicar os protocolos de comunicação envolvidos em IoT.
- Analisar a estrutura de um servidor web embarcado.
- Demonstrar a implementação de um servidor web usando MicroPython no ESP32.

1. Introdução aos Servidores Web no ESP32

O ESP32 é um microcontrolador potente com conectividade Wi-Fi e Bluetooth, tornando-se uma excelente opção para aplicações IoT. Uma das principais funcionalidades do ESP32 é a capacidade de atuar como um **servidor web**, permitindo a interação com sensores e atuadores por meio de uma interface acessível via navegador.

2. Protocolos de IoT Utilizados na Aplicação

O desenvolvimento de sistemas IoT requer o uso de protocolos eficientes para a comunicação entre dispositivos. No caso do ESP32 como servidor web, destacam-se:

- **HTTP (HyperText Transfer Protocol)**: Utilizado para a comunicação entre o servidor (ESP32) e o cliente (navegador web). O ESP32 recebe requisições HTTP e responde com páginas HTML dinâmicas.
- **TCP/IP**: Protocolo base da comunicação em redes Wi-Fi, garantindo a transmissão dos pacotes de dados entre dispositivos.
- **AJAX (Asynchronous JavaScript and XML)**: Permite a atualização dinâmica da página web sem precisar recarregar completamente a interface.

3. Estrutura de um Servidor Web no ESP32

O servidor web no ESP32 é construído utilizando o protocolo HTTP e responde a requisições feitas por clientes conectados à mesma rede. O fluxo de funcionamento segue os seguintes passos:

- O ESP32 se conecta à rede Wi-Fi.
- Um socket é criado para escutar conexões na porta 80.
- Quando um cliente acessa o servidor via navegador, o ESP32 responde com uma página HTML.
- O navegador faz requisições periódicas via AJAX para atualizar os dados do sensor.

4. Explicação do Código Implementado

O seguinte código foi utilizado como base para o servidor web do ESP32:

```

import network
import socket
import machine
import time

# Configuração do Wi-Fi
SSID = "SEU_SSID"
PASSWORD = "SUA_SENHA"

# Conectar ao Wi-Fi
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(SSID, PASSWORD)

while not station.isconnected():
    pass

print("Conectado ao Wi-Fi")
print("Endereço IP:", station.ifconfig()[0])

# Configuração do sensor LM35
adc = machine.ADC(machine.Pin(34))
adc.atten(machine.ADC.ATTN_11DB) # Configura a atenuação para leitura de 0 a 3,3V

# Configuração do LED
led = machine.Pin(2, machine.Pin.OUT)

# Função para ler temperatura com média de 10 amostras
def read_temperature():
    total = 0
    samples = 10 # Número de leituras para a média
    for _ in range(samples):
        time.sleep(0.05) # Pequeno atraso entre leituras
        voltage = adc.read() * (3.3 / 4095) # Converte leitura ADC para tensão
        temperature = voltage * 100 # Conversão para temperatura em Celsius (LM35: 10mV/°C)
        total += temperature
    return round(total / samples, 2) # Retorna a média das leituras

# Página HTML com AJAX para atualização dinâmica
def web_page():
    html = """
<!DOCTYPE html>
<html>
<head>
    <title>Monitoramento de Temperatura</title>
    <meta charset="UTF-8">
    <script>
        function updateTemperature() {
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function() {
                if (xhr.readyState == 4 && xhr.status == 200) {
                    document.getElementById("temp_value").innerHTML = xhr.responseText;
                }
            };
            xhr.open("GET", "/temp", true);
            xhr.send();
        }
        setInterval(updateTemperature, 1000);
    </script>
    <style>
    
```

```

    body { font-family: Arial, sans-serif; text-align: center; }
    .temp { font-size: 50px; color: #ff6600; }
</style>
</head>
<body>
    <h1>Monitoramento de Temperatura - ESP32</h1>
    <p class="temp" id="temp_value">-- °C</p>
</body>
</html>
"""
return html

```

Criar servidor web

```

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(("0.0.0.0", 80))
s.listen(5)

print("Servidor Web Iniciado...")
# Laço principal
while True:
    led.on() # Liga o LED indicando a atualização da página
    conn, addr = s.accept()
    print("Conexão de:", addr)
    request = conn.recv(1024).decode()

    if request.startswith("GET /temp"):
        temp = read_temperature()
        response = "HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nConnection: close\r\n\r\n" + str(temp) + " °C"
    else:
        response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nConnection: close\r\n\r\n" + web_page()

    conn.sendall(response.encode())
    conn.close()
    led.off() # Desliga o LED após a atualização da página
    time.sleep(1) # Aguarda 1 segundo antes da próxima atualização

```

Explicação do Código

Principais Métodos Utilizados

- `network.WLAN(network.STA_IF)`: Ativa a interface de rede Wi-Fi no modo estação (STA), permitindo que o ESP32 se conecte a uma rede existente.
- `network.active(True)`: Habilita a interface de rede para iniciar a conexão Wi-Fi.
- `network.connect(SSID, PASSWORD)`: Conecta o ESP32 ao roteador Wi-Fi especificado.
- `station.isconnected()`: Retorna True se o ESP32 estiver conectado a uma rede Wi-Fi.
- `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`: Cria um socket TCP/IP para comunicação entre dispositivos na rede.
- `socket.bind(("0.0.0.0", 80))`: Associa o socket à porta 80, tornando o ESP32 um servidor HTTP.
- `socket.listen(5)`: Define o servidor para escutar até 5 conexões simultâneas.
- `socket.accept()`: Aguarda conexões de clientes e retorna um objeto de conexão.
- `socket.recv(1024)`: Recebe até 1024 bytes de dados enviados pelo cliente.

- `socket.sendall(response.encode())`: Envia a resposta HTTP ao cliente, garantindo que todos os dados sejam transmitidos corretamente.
- `socket.close()`: Fecha a conexão com o cliente após o envio da resposta.
- `adc.read()`: Lê o valor analógico do sensor LM35.

Bibliotecas Utilizadas

network: Responsável pela configuração e gerenciamento da conexão Wi-Fi do ESP32. Permite ativar a interface de rede, conectar-se a uma rede Wi-Fi e obter informações sobre a conexão.

socket: Implementa a comunicação entre dispositivos na rede utilizando o protocolo TCP/IP. No código, essa biblioteca é usada para criar o servidor web que escuta conexões HTTP e responde aos clientes.

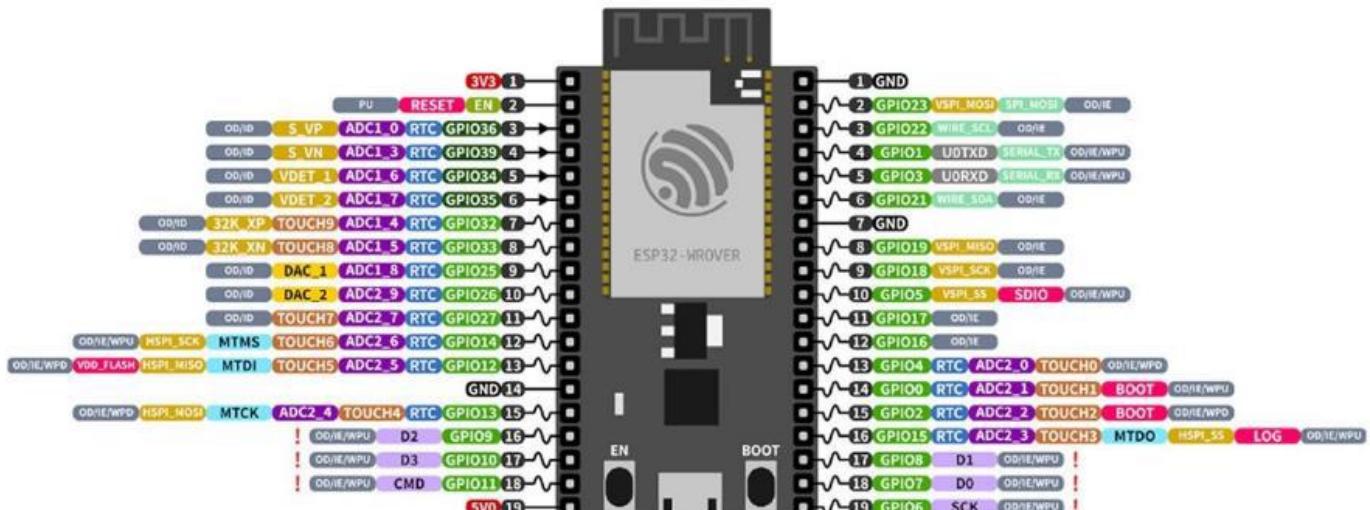
- **Conexão Wi-Fi:** O ESP32 se conecta à rede configurada e obtém um endereço IP.
- **Leitura da Temperatura:** O ESP32 realiza 10 leituras do sensor LM35 e calcula a média para estabilizar o valor.
- **Configuração do Servidor Web:** Um socket é criado para escutar conexões na porta 80.
- **Respostas HTTP:** O ESP32 responde a requisições:
 - /temp → Retorna apenas o valor da temperatura.
 - Página principal → Exibe a temperatura em uma interface simples.
- **Indicação por LED:** O LED pisca sempre que uma nova leitura é realizada.

5. Importância da Atualização em Tempo Real (AJAX)

Para evitar a necessidade de recarregar a página, utilizamos o AJAX no código HTML. Ele faz requisições assíncronas ao servidor para obter apenas a temperatura, reduzindo o consumo de rede e melhorando a experiência do usuário.

6. Conclusão

Nesta aula, exploramos como utilizar o ESP32 como um servidor web para aplicações IoT. Compreendemos a estrutura do protocolo HTTP, a configuração do ESP32 para redes Wi-Fi e a implementação de uma página web interativa com leitura de sensores. O servidor web permite a coleta e exibição remota de dados, sendo uma solução eficiente para sistemas de monitoramento IoT.

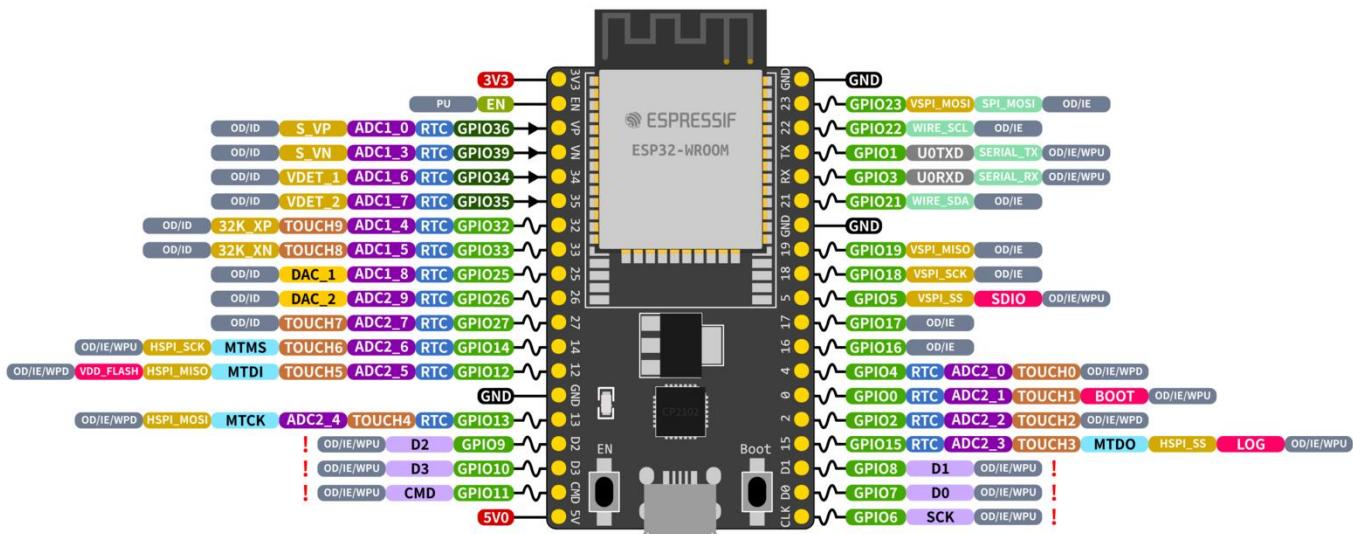


ESP32 Specs

32-bit Xtensa® dual-core @240MHz
 Wi-Fi IEEE 802.11 b/g/n 2.4GHz
 Bluetooth 4.2 BR/EDR and BLE
 520 KB SRAM (16 KB for cache)
 448 KB ROM
 34 GPIOs, 4x SPI, 3x UART, 2x I2C,
 2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
 1 slave SDIO/SPI, TWA™, 12-bit ADC, Ethernet

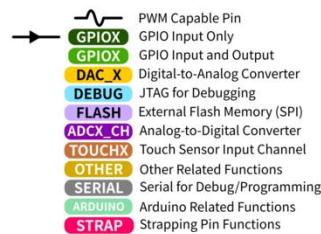


GPIO STATE
WPU: Weak Pull-up (Internal)
WPD: Weak Pull-down (Internal)
PU: Pull-up (External)
IE: Input Enable (After Reset)
ID: Input Disabled (After Reset)
OE: Output Enable (After Reset)
OD: Output Disabled (After Reset)



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
Bluetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet



RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE

WPU: Weak Pull-up (Internal)
WPD: Weak Pull-down (Internal)
PU: Pull-up (External)
IE: Input Enable (After Reset)
ID: Input Disabled (After Reset)
OE: Output Enable (After Reset)
OD: Output Disabled (After Reset)