Aula Teste

Código Assíncrono em JavaScript

Código Assíncrono em Java Script

- O que é código assíncrono?
    - Código que não bloqueia a execução de outras tarefas enquanto espera uma operação ser concluída.
    - Fundamental para aplicações que fazem requisições de rede, manipulação de arquivos ou esperam eventos externos.
- Por que é importante?
    - Mantém as aplicações rápidas e responsivas.
    - Evita o famoso "travamento" quando há operações demoradas.

Execução de código assíncrono em JavaScript

- Single-threaded
  - JavaScript tem uma única thread de execução, mas consegue lidar com tarefas assíncronas graças ao Event Loop.

- Event Loop
  - Mecanismo que gerencia a execução de tarefas assíncronas.
  - Move funções pendentes da Callback Queue para a Call Stack quando ela está vazia.

Abordagens

- Callback
  - Uma função passada como argumento para outra, para ser chamada quando a tarefa estiver pronta.
- Promises
  - Objetos que representam a eventual conclusão (ou falha) de uma operação assíncrona.
- Async/Await
  - Sintaxe moderna baseada em Promises.
  - Torna o código assíncrono mais parecido com o síncrono.

Callback

- O que é um callback?
  - É uma função que você passa como argumento para outra função.
  - Essa função será chamada (executada) depois que uma certa tarefa terminar.
  - Muito útil para operações assíncronas, como ler arquivos, fazer requisições web, temporizadores etc.

Callback (Exemplo 1)

javascript

```javascript
function saudacao(nome, callback) {
    console.log("Olá " + nome);
    callback();
}

saudacao("Maria", () => {
    console.log("Seja bem-vinda!");
});
```

Callback (Exemplo 2)

javascript

```javascript
function esperarEExecutar(callback) {
  setTimeout(() => {
    console.log("Esperou 2 segundos...");
    callback();
  }, 2000);
}

esperarEExecutar(() => {
  console.log("Callback executado após espera");
});
```

Callback (Problemas)

javascript

```javascript
fazerAlgo1(() => {
  fazerAlgo2(() => {
    fazerAlgo3(() => {
      console.log("Tudo pronto!");
    });
  });
});
```

Promise

- É um objeto que representa uma operação assíncrona que pode:
    - Resolver (sucesso)
    - Rejeitar (erro)
- Permite encadear ações e melhorar a legibilidade do código assíncrono.

Promise (Exemplo 1)

javascript

```javascript
const promessa = new Promise((resolve, reject) => {
    setTimeout(() => resolve("Dados prontos!"), 2000);
});

promessa.then(msg => console.log(msg));
```

Promise (Exemplo 2)

javascript

```javascript
const promessa = new Promise((resolve, reject) => {
  setTimeout(() => {
    const sucesso = true;
    if (sucesso) {
      resolve("Deu tudo certo!");
    } else {
      reject("Algo deu errado.");
    }
  }, 2000);
});

promessa
  .then(resultado => console.log("Sucesso:", resultado))
  .catch(erro => console.error("Erro:", erro));
```

async/await

- São palavras-chave que facilitam o uso de Promises.

- Servem para escrever código assíncrono de uma forma que parece síncrona, tornando o código mais legível e fácil de entender.

- Evitam o uso de .then() e .catch() para lidar com resultado das Promises

async/await (Exemplo 1)

javascript

```javascript
async function executar() {
    const resposta = await new Promise((resolve) => {
        setTimeout(() => resolve("Finalizado!"), 2000);
    });
    console.log(resposta);
}

executar();
```

async/await (Exemplo 2)

javascript Copy Edit

```javascript
function buscarDadosDoServidor() {
    return new Promise((resolve, reject) => {
        console.log("Buscando dados...");
        setTimeout(() => {
            resolve({ nome: "João", idade: 30 });
        }, 2000);
    });
}

async function processarDados() {
    console.log("Iniciando processamento...");

    const dados = await buscarDadosDoServidor();
    console.log("Dados recebidos:", dados);

    console.log(`Nome: ${dados.nome}, Idade: ${dados.idade}`);

    console.log("Processamento concluído.");
}

processarDados();

console.log("Este log aparece antes dos dados serem recebidos!");
```

Resumo

- **Callback**
  - ✓ Função passada como argumento para outra função
  - ✓ Executada após uma operação terminar
  - ✓ Pode causar código difícil de entender com muitas funções aninhadas (Callback Hell)
- **Promise**
  - ✓ Objeto que representa uma operação assíncrona
  - ✓ Permite encadear operações usando .then() e tratar erros com .catch()
  - ✓ Código mais organizado que callbacks aninhados
- **Async/Await**
  - ✓ Sintaxe moderna para trabalhar com Promises
  - ✓ Código fica parecido com síncrono, fácil de ler e entender
  - ✓ Usa try/catch para tratamento de erros