RESUMO DA AULA

10 Promises

Promises s o uma maneira moderna de lidar com opera es ass ncronas no JavaScript. Elas permitem representar fluxos ass ncronos de maneira mais sequencial e leg vel, al m de facilitarem o tratamento de exce es caso algo d errado nesse processo. O que uma Promise? Uma Promise representa um valor que pode estar dispon vel agora, no futuro ou nunca. Basicamente, ela uma promessa de que algo ser conclu do (com sucesso ou falha) no futuro. Uma Promise pode estar em um dos seguintes estados: - Pendente (pending): Estado inicial, ainda n o resolvida. - Realizada (fulfilled): A opera o foi conclu da com sucesso. - Recusada (rejected): A opera o falhou. - Estabelecida (settled): Foi realizada ou recusada, indicando que a opera o foi conclu da de alguma forma. Criando uma Promise Para criar uma Promise, usamos a sintaxe new Promise(), onde passamos uma fun o callback que possui dois par metros principais: resolve e reject. Esses par metros s o fun es que indicam se a opera o foi conclu da com sucesso (resolve) ou com falha (reject). let ferverAgua = () => { return new Promise((resolve, reject) => { let chaleiraEstaNoFogao = true; let fogaoEstaLigado = true; if (chaleiraEstaNoFogao && fogaoEstaLigado) { console.log('Come ando a ferver a gua...'); resolve(" gua pronta para o caf !"); // Sucesso } else { reject("Erro: necess rio colocar a chaleira no fog o e ligar o fog o."); // Falha } }); }; No exemplo acima, criamos uma fun o ferverAgua que retorna uma Promise. A Promise verifica se a chaleira est no fog o e se o fog o est ligado. Se ambos forem true, a promessa cumprida (resolve). Caso contr rio, ela rejeitada (reject). Usando a Promise com then e catch Para capturar o resultado de uma Promise, usamos os m todos then e catch. O then executado quando a promessa cumprida, e o catch executado quando a promessa rejeitada. ferverAgua() .then((resultado) => { console.log(resultado); // " gua pronta para o caf !" console.log("Continuando a fazer o caf ..."); }) .catch((erro) => { console.error(erro); // "Erro: necess rio colocar a chaleira no fog o e ligar o fog o." }); Evitando o

Callback Hell Antes das Promises, o JavaScript usava fun es de callback para lidar com c digo ass ncrono, o que levava ao problema conhecido como callback hell um encadeamento de fun es aninhadas que tornava o c digo dif cil de ler e manter. Exemplo de Callback Hell: colocarAguaPraFerver(function () { prepararFiltroECafe(function () { passarOCafe(function () { console.log("Caf pronto!"); }); }); }); Usando Promises, o mesmo c digo pode ser reescrito de forma mais leg vel: colocarAguaPraFerver() .then(() => prepararFiltroECafe()) .then(() => passarOCafe()) .then(() => console.log("Caf pronto!")) .catch((erro) => console.error("Erro ao fazer o caf : " + erro)); Como as Promises ajudam? - Facilitam o fluxo ass ncrono: As Promises ajudam a escrever c digo ass ncrono de forma mais organizada e f cil de entender. - Tratamento de Erros: Com as Promises, o tratamento de erros se torna mais eficiente e padronizado usando o .catch. - Encadeamento de Promises: Permite que as opera es ass ncronas sejam encadeadas, tornando a leitura do c digo mais fluida e sem a necessidade de fun es aninhadas. Exemplo Pr tico Vamos construir um exemplo pr tico utilizando Promises para simular o processo de fazer caf : function colocarAguaPraFerver() { return new Promise((resolve, reject) => { console.log('Colocando gua para ferver...'); setTimeout(() => { console.log(' gua ferveu!'); resolve(); }, 2000); }); } function prepararTudoParaOCafe() { return new Promise((resolve) => { console.log('Pegando o p de caf ...'); console.log('Pegando o filtro...'); console.log('Colocando o caf no filtro...'); resolve(); }); } function passarOCafe() { return new Promise((resolve) => { console.log('Passando o caf ...'); setTimeout(() => { console.log('Caf pronto!'); resolve(); }, 1000); }); } colocarAguaPraFerver() .then(() => prepararTudoParaOCafe()) .then(() => passarOCafe()) .then(() => console.log("O caf est servido!")) .catch((erro) => console.error("Erro ao preparar o caf : " + erro)); Como o exemplo funciona: A fun o colocarAguaPraFerver executada e leva 2 segundos para "ferver" a gua. Ap s a conclus o dessa etapa, prepararTudoParaOCafe chamada para preparar os filtros e o p de caf . Em seguida, passarOCafe executada, o que leva mais 1 segundo. Quando todas as etapas s o conclu das, a mensagem final "O caf est servido!" exibida. Caso qualquer uma das Promises falhe, o catch ser acionado e exibir a mensagem de erro.

Conclus o As Promises tornaram a execu o ass ncrona muito mais organizada e eficiente no JavaScript. Elas permitem que a gente fuja do callback hell e crie fluxos ass ncronos que sejam f ceis de ler e manter. Al m disso, elas oferecem uma maneira simples e padronizada de lidar com erros e controlar o fluxo de execu o.
