01

# Conhecendo pattern .NET para cancelamento

### Transcrição

O objetivo atual consiste em oferecermos ao usuário a oportunidade de cancelar a consolidação de seus dados. Por enquanto, o que acontece é que o usuário inicia o processamento e, caso ele queira cancelá-lo, suas opções se restringem ao clique no " x " da janela. Ou, na pior das hipóteses, ele irá ao Gerenciador de Tarefas, "matando" o processo do ByteBank. Esta não é uma boa abordagem, pois temos várias *threads* trabalhando simultaneamente, e não se sabe em qual estado está cada uma delas quando o usuário "mata" o processo. Todas elas podem estar em um determinado estado que, quando o usuário interfere no processo, torna a aplicação inválida, com consequências tais como perda de dados do cliente e do banco, e com certeza não queremos isto.

Precisamos implementar uma maneira que possibilite o cancelamento de forma amigável pelo usuário, sem que a aplicação entre em um estado inválido. Atualmente, qual é o fluxo da unidade de trabalho? É aquele realizado por uma *task*, ou seja, cada *task* tem um cliente que consolida seus dados, retornando-os.

É necessário pensarmos em um contexto que prevê ao usuário o ato de cancelamento. Quando temos uma tarefa, há uma pergunta a ser feita: "O usuário cancelou a consolidação?" Se ele não cancelou, vamos consolidar os dados. Feito isto, nos perguntamos de novo: "O usuário cancelou?" Antes de entregarmos o trabalho finalizado, é bom perguntar mais uma vez. "Devo entregar? O usuário cancelou?" Se ele não cancelou, o resultado será retornado.

Quando o usuário de fato cancela o processamento, a unidade de trabalho, a *task*, é informada. Podemos reverter algum trabalho, fazer um *log* do ocorrido e, por convenção do .NET, lança-se uma exceção especial, indicando que houve cancelamento de uma tarefa.

Como este *pattern* é representado no .NET? A pergunta "O usuário cancelou a consolidação?" é implementada com a estrutura `CancellationToken`, representando uma potencial requisição de cancelamento, que verificamos pela propriedade pública `IsCancellationRequested;`. Ela é `readonly`, ou seja, possui apenas leitura.

Sendo assim, o que modifica esta estrutura? Todo `CancellationToken` possui, associado à ele, um `CancellationTokenSource` que o origina. Ele pode criar um destes e, a partir de seu método ( `Cancel()` ), modifica-se todos os `CancellationToken`, informando que houve requisição de cancelamento por parte do usuário e, a partir de então, a propriedade `IsCancellationRequested;` se torna `true` (verdadeira). Com isto, tomamos as medidas necessárias para o caso do usuário cancelar a operação.

Podemos também passar este `CancellationToken` à frente, isto significa que estamos usando a extensão `ContinueWith` da classe `Task`, podendo passar a ela, por parâmetro, o mesmo `CancellationToken`, replicando esta estrutura. No fim, todos estarão apontando ao mesmo `CancellationTokenSource`.

Isto é importante pois uma tarefa muitas vezes é composta de várias outras menores, e todas elas devem estar em sincronia em relação ao `CancellationToken`, apontando ao mesmo `CancellationTokenSource`. Vamos implementar este código mais adiante.