

DESENVOLVIMENTO DE SISTEMAS COM C#

Cleverson Lopes Ledur



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS



Utilizar o Entity Framework para persistência em banco de dados – III

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Programar a camada DAL (Data Access Layer) no projeto para fazer o acesso aos dados do banco.
- Aplicar dados no banco.
- Praticar as possíveis exceções que podem ocorrer ao inserir uma informação no banco.

Introdução

Desenvolver sistemas sem transformar o código em uma grande bagunça é algo importante. Afinal, muitas vezes várias pessoas irão fazer modificações no mesmo projeto, não é? Por isso, o uso de camadas para o desenvolvimento é um conceito muito utilizado atualmente. Uma das camadas é chamada de DAL ou camada de acesso aos dados. Como o próprio nome já diz, ela é a camada em que ficarão as classes usadas para acessar o banco de dados.

Neste capítulo, você vai conhecer e criar uma camada DAL para acessar o banco de dados. Também vai ver como inserir dados no banco e evitar que possíveis exceções não esperadas sejam recebidas pelo usuário.

A camada DAL

No desenvolvimento de sistemas, geralmente há grande quantidade de códigos. Muito desse código fica armazenado de acordo com o tipo de entidade com que você está trabalhando. A abordagem recomendada, no entanto,

é separar a lógica de acesso a dados da camada de apresentação. Essa camada separada é referida como a camada de acesso a dados ou DAL. Essa é uma classe curta e normalmente é implementada como um projeto de biblioteca de classes separado ou como uma simples pasta no sistema (MITCHELL, 2006).



Fique atento

Os benefícios da arquitetura em camadas são diversos. Entre eles, você pode considerar os listados a seguir.

- Melhor organização do código.
- Facilidade de manutenção.
- Facilidade de novas implementações.
- Encapsulamento e reutilização de código.
- Maior controle sobre as alterações no banco de dados.
- Facilidade de trabalho em equipe no projeto.

Todo o código específico da fonte de dados — como criar uma conexão ao banco de dados, emitir comandos `select`, `insert`, `update`, `delete` e assim por diante — deve estar localizado na DAL (HAMIDREZA-GHA-SEMKHAH, 2007). A camada de apresentação não deve conter nenhuma referência a esse código de acesso a dados. Deve, em vez disso, fazer chamadas na DAL para todas e quaisquer solicitações de dados. As camadas de acesso a dados geralmente contêm métodos para acessar o banco de dados subjacente. Com o uso do Entity Framework, você não precisa criar chamadas com código SQL, mas tem a possibilidade de utilizar, dentro da camada DAL, as chamadas dos métodos do próprio Entity Framework.

Considere um projeto que já possui o Entity Framework e o uso de migrations habilitado. Suponha também que nesse projeto há um objeto de domínio chamado `Usuário`. A adição de um novo usuário a partir de um método de execução se daria com um código semelhante ao que você pode ver a seguir.

```

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Digite o nome do usuário: ");
        String nome = Console.Read().ToString() ;
        Console.WriteLine("Digite o apelido do usuário: ");
        String apelido = Console.Read().ToString();
        Console.WriteLine("Digite a idade do usuário: ");
        int idade = Console.Read();

        Context con = new Context();

        Usuario user = new Usuario();
        user.Nome = nome;
        user.Apelido = apelido;
        user.Idade = idade;

        con.Usuarios.Add(user);
        con.SaveChanges();
        con.Dispose();
    }
}

```

Você teria, inicialmente, que criar uma instância da classe de contexto. Então, a partir dela, precisaria fazer a adição do usuário. Portanto, deveria utilizar o método `Dispose` para finalizar a conexão com o banco. E se antes de adicionar o usuário você precisar colocar algumas verificações para evitar problemas no banco? No próximo exemplo, você pode ver como ficaria o seu código com essas verificações. Você pode adicionar uma verificação para garantir que não vai receber uma idade negativa.

```

class Program{
    static void Main(string[] args){

        Console.WriteLine("Digite o nome do usuário: ");
        String nome = Console.Read().ToString() ;
        Console.WriteLine("Digite o apelido do usuário: ");
        String apelido = Console.Read().ToString();
        Console.WriteLine("Digite a idade do usuário: ");
        int idade = Console.Read();

        if(idade<0){
            throw new Exception("A idade do usuário deve ser um número positivo.");
        }

        Context con = new Context();

        Usuario user = new Usuario();
        user.Nome = nome;
        user.Apelido = apelido;

        con.Usuarios.Add(user);
        con.SaveChanges();
        con.Dispose();
    }
}

```

Agora, imagine que você precisa inserir mais três usuários. Você deve repetir todo esse código? Pensando nisso e no reuso de código, foi definido o uso de software em camadas. Uma dessas camadas é a DAL. A seguir, você vai ver como criar uma DAL que irá ajudá-lo na organização desse código.

Primeiro, você deve criar diretórios no seu projeto. Você pode utilizar também bibliotecas de classes, mas, para focar nos conceitos da camada DAL, o exemplo que você vai ver aqui utilizará diretórios. Então, pressione o botão direito do mouse no nome do projeto e clique em **Adicionar > Nova Pasta**. Veja esse procedimento na Figura 1.

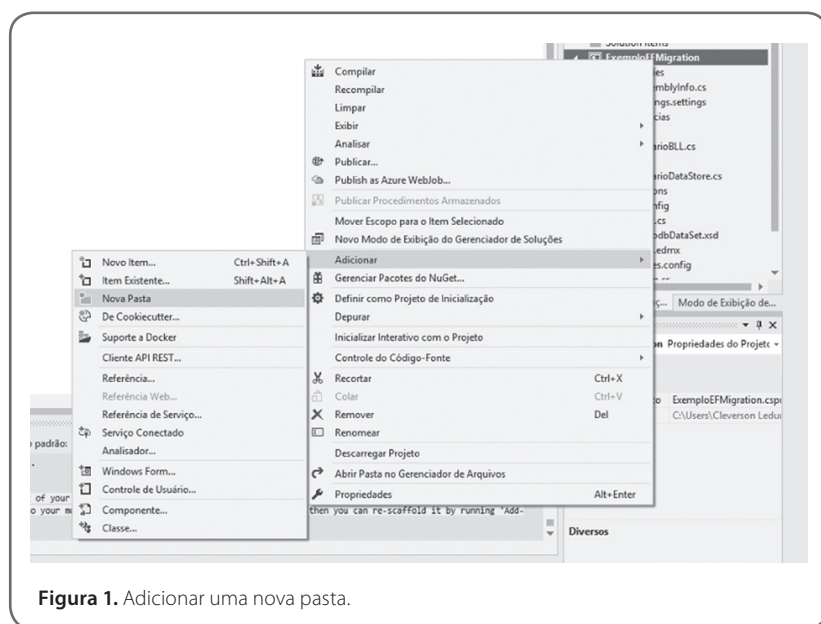


Figura 1. Adicionar uma nova pasta.

Digite o nome da pasta quando ela for inserida. Você deve criar uma pasta chamada DAL e outra chamada **BLL (Business Logic Layer)**. A pasta DAL irá conter suas classes que irão acessar o banco de dados. Já a classe **BLL** é onde ficarão seus métodos de regras de negócio. Logo mais, você irá entender melhor o que a BLL fará. Na Figura 2, você pode ver como ficará o projeto com as pastas.

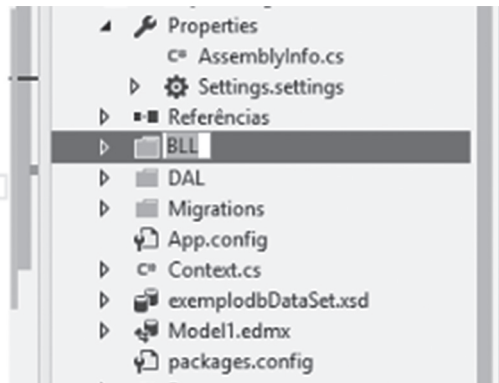


Figura 2. Projeto com pastas.

Agora, você deve criar uma nova classe dentro do seu diretório DAL. Você pode chamá-la de `UsuarioDataStore.cs`, já que ela será a sua classe DAL responsável por acessar os dados do banco relacionados com `Usuario`. Sua classe deverá conter alguns métodos de acesso, conforme a aplicação necessitar. A seguir, você pode ver um exemplo de classe DAL.

```
class UsuarioDataStore{
    Context = new Context();

    ~UsuarioDataStore()
    {
        context.Dispose();
    }

    public List<Usuario> Listar()
    {
        return context.Usuarios.ToList();
    }
}
```

Note que, utilizando uma camada DAL, você precisa apenas instanciar sua classe de contexto uma vez. Nessa classe, insira apenas o método `Listar`. Na próxima seção, você vai ver como inserir um dado por meio da sua camada DAL. Muito simples, não? Então, para cada objeto de modelo do seu projeto, o ideal é que exista uma classe DAL. Assim fica mais fácil saber onde você deve fazer modificações quando uma nova funcionalidade for implementada no sistema.

Inserindo dados no banco

Agora, para que a aplicação funcione, você precisa inserir informações. Logo, a inserção de dados será realizada na classe DAL. De forma geral, quando se trabalha com o Entity Framework, os dados são inseridos por meio do método `Add` originário do objeto `DbSet` dentro da classe contexto. Ou seja, para criar um novo usuário, por exemplo, você teria que chamar `[objeto contexto].[classe Usuário].Add` e passar como parâmetro um objeto do tipo usuário. No próximo exemplo, você pode ver uma forma para entender melhor. No exemplo, já foi criado o método `Adicionar` na classe DAL.

```
public Usuario Adicionar(Usuario usuario){
    Usuario novoUsuario = context.Usuarios.Add(usuario);
    context.SaveChanges();
    return novoUsuario;
}
```

Nesse caso, é recebido um parâmetro `Usuário`, que é então adicionado ao banco de dados por meio do Entity Framework. Muito simples, não? Assim, você não precisa criar um script SQL que realiza a inserção no banco.

Mas agora fica a dúvida: quando utilizar o método `Adicionar`? Você deve se lembrar de que, na seção anterior, aprendeu a criar uma pasta chamada BLL. Lá, você deve criar todas as suas classes e métodos relacionados com as regras de negócio. Precisa, então, criar uma classe dentro de BLL chamada `UsuarioBLL.cs`. Ela deve conter o conteúdo mostrado no próximo código.

```
public Usuario CriarUsuario(String nome, int idade, String apelido)
{
    if(idade<0){
        throw new Exception("A idade do usuário deve ser um número positivo.");
    }

    UsuarioDataStore usuarioDS = new UsuarioDataStore();

    Usuario novoUsuario = new Usuario();
    novoUsuario.Nome = nome;
    novoUsuario.Idade = idade;
    novoUsuario.Apelido = apelido;

    return usuarioDS.Adicionar(novoUsuario);
}
```

Veja só: as suas verificações vão ficar dentro dessa classe, já que é uma regra de negócio do seu sistema não aceitar idades negativas. Bom, mas o

que aconteceria se você passasse um valor não esperado ao banco de dados? Continue lendo este capítulo para ver como lidar com exceções no C#.

Tratamento de exceções

Uma exceção é um problema que ocorre durante a execução de um programa. Uma exceção C# é uma resposta a uma circunstância excepcional que surge enquanto um programa está sendo executado, como uma tentativa de dividir por zero (HILLAR, 2010).

As exceções fornecem uma maneira de transferir o controle de uma parte de um programa para outro. O tratamento de exceção C# é construído com base em quatro palavras-chave: **try**, **get**, **finally** e **catch** (HILLAR, 2010).



Saiba mais

A seguir, você pode conhecer melhor cada uma das palavras-chave utilizadas para o tratamento de exceção em C#.

- **try:** um bloco `try` identifica um bloco de código para o qual determinadas exceções são ativadas. É seguido por um ou mais blocos de captura.
- **catch:** um programa captura uma exceção com um manipulador de exceção no local em um programa no qual você deseja lidar com o problema. A palavra-chave `catch` indica a captura de uma exceção.
- **finally:** o bloco `finally` é usado para executar determinado conjunto de declarações se uma exceção é lançada ou não é jogada. Por exemplo, se você abrir um arquivo, ele deve ser fechado se uma exceção é gerada ou não.
- **throw:** um programa lança uma exceção quando surge um problema. Isso é feito usando a palavra-chave `throw`.

Sintaxe

Suponha que um bloco eleva uma exceção e um método captura uma exceção usando uma combinação das palavras-chave `try` e `catch`. Um bloco `try/catch` é colocado ao redor do código que pode gerar uma exceção. O código dentro de um bloco `try/catch` é referido como código protegido, e a sintaxe para usar `try/catch` parece ser a que você pode ver a seguir.


```

try{
    // declarações causando exceção
} catch (ExceptionName e1) {
    // código de tratamento de erros
} catch (ExceptionName e2) {
    // código de tratamento de erros
} catch (ExceptionName eN) {
    // código de tratamento de erros
} finally{
    // declarações a serem executadas
}

```

Na classe do projeto que você está criando, pode inserir mais algumas verificações para evitar a inserção de valores nulos ou strings vazias. Também pode inserir um bloco `try/catch` ao chamar o método DAL que tentará inserir as informações. Assim, você tem a possibilidade de evitar que um erro seja exibido para o usuário e de informar um erro mais amigável. Veja o próximo código.

```

public Usuario CriarUsuario(String nome, int idade, String apelido)
{
    if (String.IsNullOrEmpty(nome) || String.IsNullOrWhiteSpace(nome))
    {
        throw new Exception("O nome do usuário deve ser informado.");
    }

    if(idade<0){
        throw new Exception("A idade do usuário deve ser um número
positivo.");
    }

    UsuarioDataStore usuarioDS = new UsuarioDataStore();

    Usuario novoUsuario = new Usuario();
    novoUsuario.Nome = nome;
    novoUsuario.Idade = idade;
    novoUsuario.Apelido = apelido;

    try
    {
        return usuarioDS.Adicionar(novoUsuario);
    }
    catch(Exception e)
    {
        throw new Exception("Ocorreu um erro ao inserir o usuário no banco
de dados", e);
    }
}

```

Perfeito! Agora parece que você tem um código bastante protegido contra exceções não esperadas. Também tem uma classe DAL que acessa o banco de dados. Logo, se você precisar implementar uma nova funcionalidade que altera (update) ou remove os usuários, já sabe onde deve criar os métodos.

Observe, no exemplo a seguir, como ficou o seu método principal, que você viu no início do capítulo.

```
static void Main(string[] args)
{
    Console.WriteLine("Digite o nome do usuário: ");
    String nome = Console.Read().ToString() ;
    Console.WriteLine("Digite o apelido do usuário: ");
    String apelido = Console.Read().ToString();
    Console.WriteLine("Digite a idade do usuário: ");
    int idade = Console.Read();

    UsuarioBLL.CriarUsuario(nome, idade, apelido);
}
```

Muito bem, agora você conseguiu organizar seu código utilizando as camadas no projeto e encapsulando o código que faz o acesso ao banco de dados em uma classe específica chamada DAL. Também, com o uso do conceito BLL, conseguiu isolar seu código que possui as regras de negócio. Por exemplo, se a partir de hoje quiser inserir apenas usuários que tenham idade maior que 18 anos, você pode simplesmente adicionar uma condição na classe dentro de BLL, sem precisar modificar todo o código. O mesmo ocorre se alguma regra não for mais utilizada.



Referências

HAMIDREZA-GHASEMKHAH. *DAL class and Transact-SQL generator for C# and VB.NET*. [S.l.]: CodeProject, 2007. Disponível em: <<https://www.codeproject.com/Articles/21507/DAL-Class-and-Transact-SQL-Generator-for-C-and-VB>>. Acesso em: 24 nov. 2017.

HILLAR, G. *Professional parallel programming with C#: master parallel extensions with .NET 4*. Birmingham: Wrox, 2010.

MITCHELL, S. *Tutorial 1: creating a data access layer*. Redmond: Microsoft, 2006. Disponível em: <<https://msdn.microsoft.com/en-us/library/aa581776.aspx>>. Acesso em: 02 nov. 2017.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS