

Nesta seção você encontra artigos intermediários sobre a tecnologia .net

Service Pack 1

Visual Studio 2008 e .NET Framework 3.5

Neste artigo veremos

ADO.NET Entity Framework
ADO.NET Data Services
ASP.NET Dynamic Data

Qual a finalidade

Explorar as principais novidades do Service Pack 1 do Visual Studio 2008 e .NET Framework 3.5

Quais situações utilizam esses recursos?

No desenvolvimento de aplicações Web onde a alta produtividade e qualidade do resultado final são pontos fundamentais.

O lançamento do *Service Pack 1* do *Visual Studio 2008* e *.NET Framework 3.5* já não é nenhuma novidade. Ele foi disponibilizado no começo de Agosto/2008, e se você é desenvolvedor .NET já deve estar cansado de ouvir falar do SP1 e de suas novidades.

Na modesta opinião deste autor, as principais features que o Service Pack 1 trouxe são: ADO.NET Entity Framework, ADO.NET Data Services e o ASP.NET Dynamic Data. É claro que estas não são as únicas, mas certamente as mais significativas. Neste artigo vamos abordar cada um destes tópicos de forma mais detalhada, com exemplos práticos de como utilizar cada um deles.



Rodrigo Sendin

rodrigo.sendin@terra.com.br

É Arquiteto de Sistemas e trabalha com desenvolvimento de Software há mais de 12 anos. Tecnólogo formado pela FATEC de Americana e MCP .NET. É consultor da TauNet Consulting e escreve artigos para .net Magazine e WebMobile. Blog: <http://www.algoritma.com.br/rodrigo.sendin>



Resumo do DevMan

O Service Pack 1 do .Net Framework 3.5 trouxe grandes novidades para o framework. Novas funcionalidades como o Entity Framework, um mapeador objeto-relacional, o ASP.Net Data Services, que facilita a cloud computing, e o Dynamic Data, um poderoso modelo de aplicação web baseado em templates, estão entre suas melhores novidades.

Baixando e Instalando o Service Pack 1

Você vai encontrar o download do *Service Pack 1* no seguinte site: [http://msdn.microsoft.com/pt-br/vstudio/cc533448\(en-us\).aspx](http://msdn.microsoft.com/pt-br/vstudio/cc533448(en-us).aspx), em quatro variações possíveis. Como você pode conferir na **Figura 1**, podemos baixar versões para *Visual Studio*, ou somente para o *.Net Framework*.

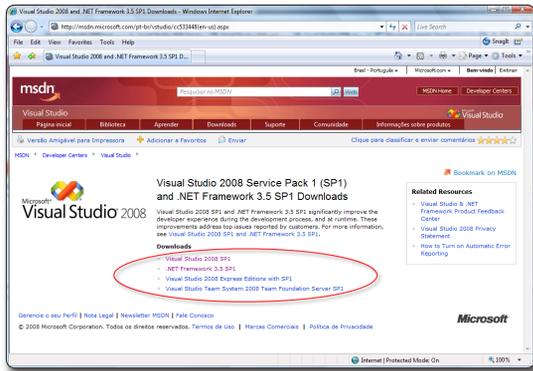


Figura 1. Página de Download do Service Pack

O primeiro link é referente ao SP1 do *Visual Studio 2008*, que naturalmente irá instalar também o SP1 do *.NET Framework 3.5* no mesmo pacote. Portanto, se você tem instalado o *Visual Studio 2008*, acesse este link, baixe e instale o SP1 por aí.

O segundo link irá lhe remeter ao download do SP1 apenas do *.NET 3.5*, que provavelmente será (se já não é no momento da publicação deste artigo) parte das atualizações automáticas do Windows. Um ponto interessante é que este SP1 não traz atualizações apenas para a versão 3.5 do *.NET*, ele também traz atualizações para as versões 2.0 e 3.0 da plataforma.

No terceiro link você poderá baixar as versões *Express* do

Visual Studio, que agora são disponibilizadas junto com o *Service Pack 1*. Se você utiliza as versões *Express*, é este link que deverá acessar para obter uma versão atualizada com o SP1.

E no último link você vai encontrar o SP1 do *Team Foundation Server*, a versão Servidor do *Visual Studio 2008*.

Não há segredo na instalação do SP1, basta baixar, executar e aguardar. Sim, o processo é demorado. Quando for atualizar o SP1 reserve um bom tempo, que deverá variar de acordo com a velocidade da sua internet, pois o download completo é feito durante a instalação. Você pode, no entanto, optar por baixar a imagem do DVD (.iso) em <http://go.microsoft.com/fwlink/?LinkId=122095>, e não precisar aguardar o download durante a instalação. Neste caso, no entanto, o download é completo, o que não acontece com a versão via web, que só baixa o que é necessário.

ADO.NET Entity Framework

Essa é uma das mais controversas novidades do SP1, e como não poderia ser diferente, é a minha preferida. O *ADO.NET Entity Framework (EF)* é o que podemos chamar de Ferramenta de Mapeamento Objeto/Relacional (OR/M). Certamente o EF não é a ferramenta OR/M dos sonhos de todos os desenvolvedores, e obviamente não vai ser. Se já não é fácil agradar dois programadores de uma mesma equipe, imagine agradar a todos os programadores *.NET*?

Porém, apesar de não agradar a todos, é isso o que *ADO.NET Entity Framework* é: uma ferramenta para fazer um mapeamento entre um banco de dados relacional e um modelo de objetos. Eu sempre fui e sou fã de carteirinha dos *DataSets Tipados*, e espero sinceramente que aos poucos o EF venha a ocupar este mesmo espaço, principalmente porque ele é uma



Nota do DevMan

VERSÕES DO .NET FRAMEWORK E VISUAL STUDIO

Visual Studio 2002 / .NET Framework 1.0

O Microsoft *.NET Framework* e o *Visual Studio* compõe o que chamamos de plataforma de desenvolvimento da Microsoft. A primeira versão oficial da plataforma surgiu em fevereiro de 2002, com o *.NET Framework 1.0* e *Visual Studio 2002* (code-nome Rainer). Esse primeiro release da então nova plataforma de desenvolvimento da Microsoft, foi pouco conhecido e utilizado, e logo deu lugar ao:

VISUAL STUDIO 2003 / .NET FRAMEWORK 1.1

Um ano depois, em Abril/2003 surgiu mais uma versão da plataforma, esta chamada de *.NET Framework 1.1* e *Visual Studio 2003* (code-nome Everett). Essa sim foi uma versão amplamente utilizada pela comunidade. (até hoje ainda tem gente que desenvolve ou dá manutenção em aplicativos feitos no VS2003).

VISUAL STUDIO 2005 / .NET FRAMEWORK 2.0

Em Outubro de 2005 a Microsoft lança uma versão mais "madura" da plataforma, é o *.NET Framework 2.0* com o *Visual Studio 2005* (code-nome Whidbey). Muita coisa foi melhorada nessa versão, tanto que a partir daqui os frameworks passaram a se "complementar" ao invés de se "sobreporem".

.NET FRAMEWORK 3.0

No lançamento do Windows VISTA surgiu uma nova versão da Framework, a 3.0. Essa versão foi marcada pela inclusão das tecnologias: Windows Presentation Foundation, Windows Communication Foundation, Windows Workflow Foundation e Windows CardSpace. Apesar de não ter "saído" um novo *Visual Studio*, foi possível utilizar essas tecnologias através de pacotes que eram instalados na versão 2005.

VISUAL STUDIO 2008 / .NET FRAMEWORK 3.5

Em Novembro de 2007 foi o lançamento do *.NET Framework 3.5* e do *Visual Studio 2008* (code-nome Orcas). Essa é a versão atual e que a maioria dos desenvolvedores *.NET* está trabalhando. Ela foi marcada principalmente pela introdução da linguagem LINQ, e por permitir a fácil migração de projetos feitos na versão anterior.

VISUAL STUDIO 2010 / .NET FRAMEWORK 4.0

Em Setembro/2008 a Microsoft publicou a primeira nota a respeito da versão da plataforma que está sendo desenvolvida, e tem previsão de lançamento para o ano de 2010. É o *.NET Framework 4.0* e o *Visual Studio 2010* (code-nome Hawaii). Isso nos mostra o dinamismo com que a plataforma vem sendo e certamente será atualizada constantemente.

ferramenta OR/M pensada para o LINQ, que indiscutivelmente veio para ficar.

Mas chega de conversa e vamos logo ver como o EF funciona na prática. Abra o seu *Visual Studio 2008 (com o SPI atualizado)* e crie um nova solução em branco, chamada *SPITestes*. Vamos utilizar essa *solution* para fazer todos os testes deste artigo.

Dentro dessa *solution* comece criando um novo projeto do tipo *Class Library* com a linguagem C#. O projeto deverá se chamar *SPITestes.ModeloEF*. Apague a *Class1.cs* que é criada automaticamente no projeto, e em seguida clique com o botão direito sobre o projeto na *Solution Explorer* e escolha a opção *Add / New Item*.

Como você pode ver na **Figura 2**, na esquerda selecione a categoria *Data*, e em seguida selecione o template *ADO.NET Entity Data Model*. Informe *DM_Northwind.edmx* em *Name* e clique em *Add*.

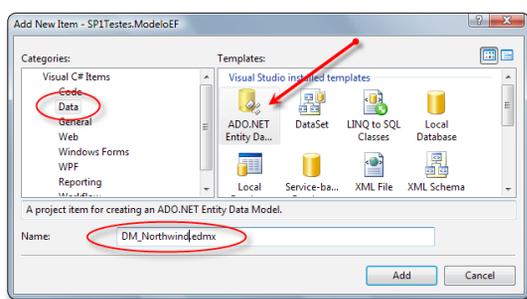


Figura 2. Criando um novo modelo com o ADO.NET Entity Framework

Para a criação do modelo será iniciado um *Wizard* para nos ajudar. Na primeira tela desse *Wizard*, que você vê aqui na **Figura 3**, temos que escolher se o nosso modelo será criado a partir de um *database* pré-existente, ou se vamos criar um modelo em branco.

Essa é uma questão interessante, pois o EF está nos dando duas possibilidades de mapeamento aqui. Podemos criar um modelo de “entidades” vazio, baseado nas regras da orientação a objetos, e somente depois definir a estrutura de um banco de dados relacional, baseado neste modelo. No final basta fazer o mapeamento.

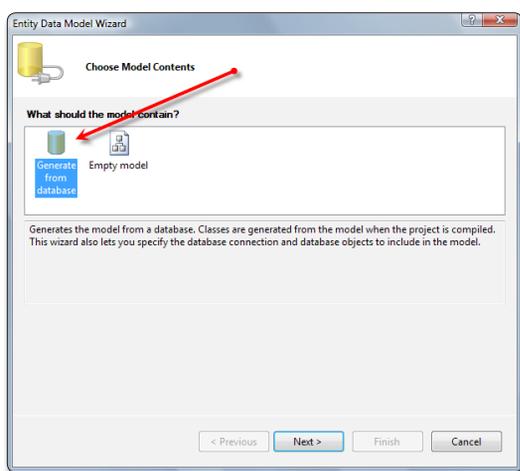


Figura 3. Definindo a geração de um modelo a partir do banco de dados

Ou podemos simplesmente utilizar um *database* já existente e criar um modelo de “entidades” baseado neste modelo. São duas formas distintas de ver o mesmo problema, que é mapear o banco de dados com um modelo de entidades. Fica ao gosto do freguês.

Vamos escolher a primeira opção (*generate from database*) por ser a mais simples, e servirá para colocarmos o EF em prática. Clique em *Next* para prosseguir.

Na segunda parte do *Wizard* (**Figura 4**) devemos definir o banco de dados que será utilizado para a criação do modelo. Veja que temos um botão chamado *New Connection*, que pode ser utilizado para a criação dessa *connection string*.

Neste exemplo estamos criando uma conexão com o *database* de testes da Microsoft *Northwind* (veja na nota como baixar este *database*). Note que a *connection string* é gerada em um formato diferente do que estamos acostumados, e ela será armazenada no arquivo *App.config* do Projeto.

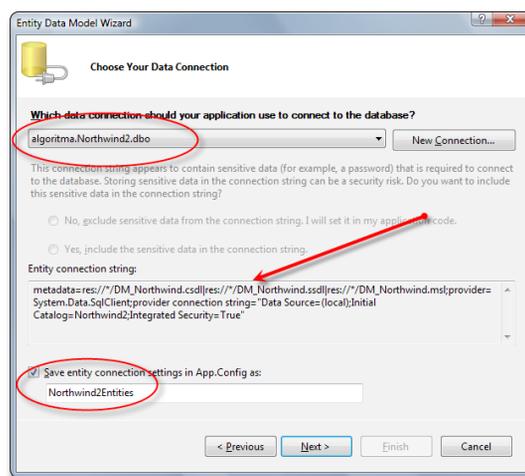


Figura 4. Definindo a connection string para acessar o database

Clique em *Next* novamente para prosseguirmos com o *Wizard*. Na terceira etapa do *Wizard* devemos escolher os objetos do banco de dados que serão mapeados no modelo. Veja na **Figura 5** que você pode mapear *Tabelas*, *Views* e/ou *Stored Procedures*.

Neste exemplo vamos escolher apenas as tabelas *Categories* e *Products* para criarmos um pequeno modelo de testes. Em seguida clique em *Finish* para finalizar a criação do modelo.

Após o término da criação do modelo, será aberta uma janela no *Visual Studio* parecida com a da **Figura 6**. Note que no centro desta janela você terá um diagrama de classes representando o modelo de entidades criado a partir das tabelas escolhidas no banco.

Você irá notar que aqui, em vez de tabelas e relacionamentos temos entidades e associações. Uma diferença crucial que você deve notar, é que na entidade *Products* não temos a propriedade *CategoryID*, que existe na tabela *Products* do *database Northwind*. Ao invés dessa propriedade, temos uma “propriedade de navegação” chamada *Categories*, que é uma referência ao objeto da classe *Categories* que estiver associado ao produto.

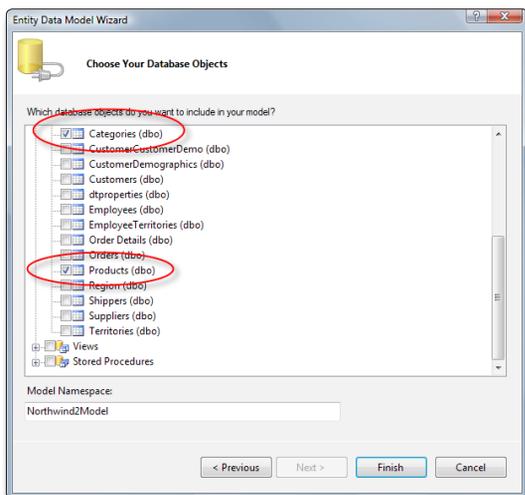


Figura 5. Escolhendo objetos do database que farão parte do modelo

Além das associações, que representam as relações entre as classes de um modelo de classes, temos também a possibilidade de definir a herança de classes em um modelo de entidades. Neste modelo não temos essa representação, mas esta é uma característica bastante importante do EF, que você pode encontrar no meu artigo *ADO.NET Entity Framework*, publicado na edição 54 da .NET Magazine.

Outra funcionalidade muito importante de um modelo de entidades é a janela de mapeamento. Se você ainda não a encontrou, clique com o botão direito na entidade *Categories* e escolha a opção *Show in Model Browser*. Você verá uma janela no canto inferior do *Visual Studio*, assim como mostra a própria Figura 6.

Veja que cada propriedade da entidade *Categories* aponta para uma coluna da tabela no banco de dados. A partir dessa mesma janela você faz o mapeamento das *Associações* também. Essa é a funcionalidade principal de uma ferramenta OR/M.

Adicionalmente, no canto direito da Figura 6, você pode conferir a janela *Model Browser*, que dá uma visão completa de todos os itens que integram o modelo, inclusive o *Store* que é onde temos os objetos do banco de dados utilizados na criação do modelo.

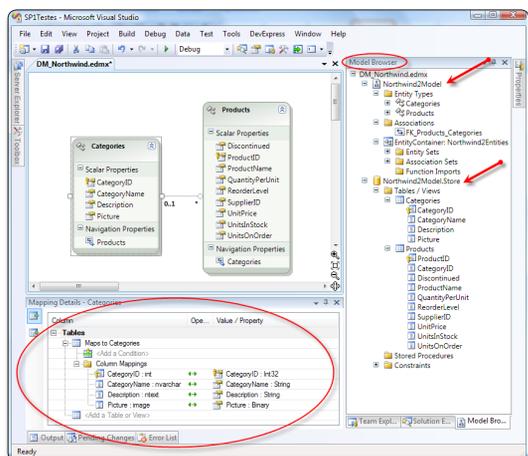


Figura 6. Janelas e opções de um modelo de entidades do EF



Nota do DevMan

O Database Northwind é uma banco de dados do SQL Server para testes e com estrutura definida para uma aplicação de vendas, onde temos Produtos, Clientes, Fornecedores, Pedidos, etc. A Microsoft disponibiliza este database já com dados através do seguinte link:
<http://tinyurl.com/northwinddb>

Faça o download e execute o arquivo de instalação. Ao término da instalação, os databases de exemplo do SQL Server 2000 serão instalados na pasta *C:\SQL Server 2000 Sample Databases*. Você pode obviamente movê-los para o local da sua conveniência.

Agora que já vimos como criar o modelo, vamos ver como podemos utilizá-lo. Crie um novo projeto na Solução, desta vez utilizando o template *ASP.NET Web Application*, novamente com a linguagem C#. O Projeto deverá se chamar *SPITestes.WebEF*.

Para que este projeto possa utilizar o modelo de entidades que foi criado, precisamos adicionar uma referência ao projeto *class library*. Para isso clique com o botão direito sobre o projeto *SPITestes.WebEF* na *Solution Explorer* e escolha a opção *Add Reference*. Vá na *Aba Projects*, selecione o projeto *SPITestes.ModeloEF* e clique em *OK*.

Você também vai precisar incluir mais uma referência a este projeto, agora ao namespace *System.Data.Entity*, para que possamos utilizar as funcionalidades do *ADO.NET Entity Framework* neste projeto. Escolha novamente a opção *Add New Reference*, e agora selecione o *System.Data.Entity* na *Aba .NET*, como você confere na Figura 7.

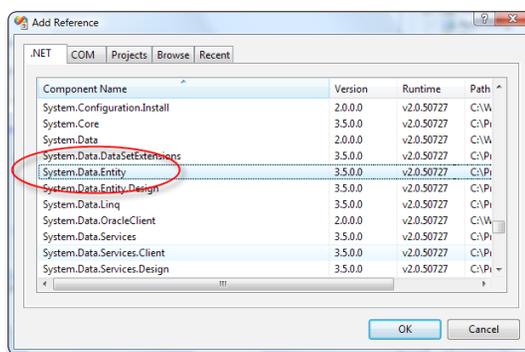


Figura 7. Adicionando a referência ao System.Data.Entity

Listagem 1. ConnectionStrings no Web.config

```
<connectionStrings>
<add name="Northwind2Entities"
connectionString="metadata=res://*/DM_Northwind.
csdl|res://*/DM_Northwind.
ssdl|res://*/DM_Northwind.msl;provider=System.Data.
SqlClient;provider connection
string="Data Source=(local);Initial Catalog=
Northwind2;Integrated
Security=True;MultipleActiveResultSets=True";
providerName="System.Data.EntityClient" />
</connectionStrings>
```

Em seguida precisamos configurar a *ConnectionString* do modelo no arquivo *Web.config*. Basta abrir o *App.config* do projeto *class library*, e copiar a *connectionString* para dentro do *Web.config*, a exemplo da **Listagem 1**.

Abra agora a página *Default.aspx* e desenhe uma interface bem simples, utilizando os controles do AJAX (*ScriptManager* e *UpdatePanel*), um *GridView* e dois *Buttons*. Veja na **Figura 8** como deve ficar a interface.

Em seguida dê um duplo-clique no primeiro botão para codificarmos a pesquisa. A primeira coisa que você deve fazer é importar o namespace do projeto onde está nosso modelo, para isso inclua a seguinte linha na seção *using* da classe:

```
using SPITestes.ModeloEF;
```

Agora, no evento *Click* do *Button1*, inclua o código da **Listagem 2**, que irá retornar todas as categorias de produtos através do nosso modelo de entidades. Faça o mesmo para o *Button2*, só que agora inclua nele o código da **Listagem 3**, para recuperar todos os produtos do modelo.

Com isso já podemos fazer o nosso primeiro teste do *ADO*.

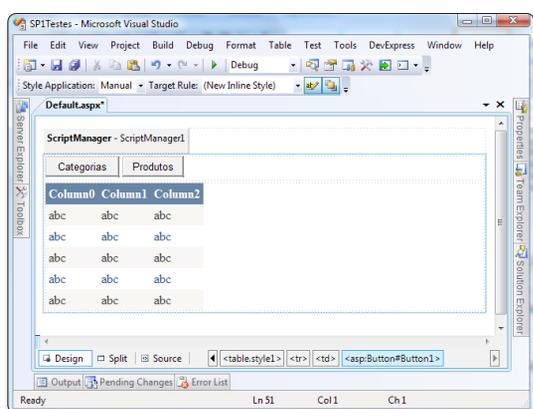


Figura 8. Interface para testes do Modelo EF

Listagem 2. Recuperando todas as categorias do modelo de entidades

```
protected void Button1_Click(object sender, EventArgs e)
{
    using (Northwind2Entities model = new Northwind2Entities())
    {
        GridView1.DataSource = model.Categories;
        GridView1.DataBind();
    }
}
```

Listagem 3. Recuperando todos os produtos do modelo de entidades

```
protected void Button2_Click(object sender, EventArgs e)
{
    using (Northwind2Entities model = new Northwind2Entities())
    {
        GridView1.DataSource = model.Products;
        GridView1.DataBind();
    }
}
```

NET Entity Framework. Marque o projeto *SPITestes.WebEF* como projeto de startup, e a página *Default.aspx* como página inicial. Em seguida salve, compile e execute o projeto.

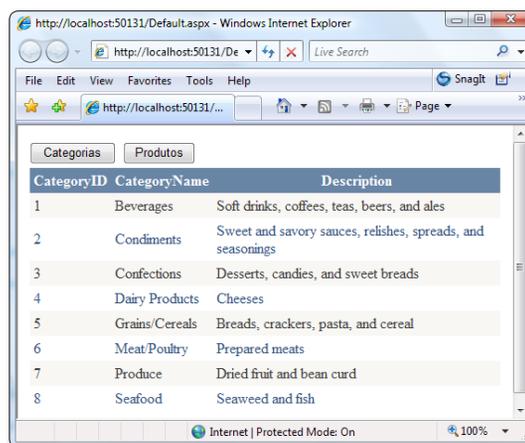


Figura 9. Testando o ADO.NET Entity Framework em uma página ASP.NET

Veja na **Figura 9** todas as categorias listadas no *GridView*. Clique no segundo *Button* para listar todos os produtos.

Quer brincar um pouco com o LINQ? Substitua o código que está no evento *Click* do *Button2*, pelo código da **Listagem 4**. O resultado dessa pesquisa você confere na **Figura 10**.

Listagem 4. Utilizando LINQ para pesquisa complexa na entidade Productcts

```
protected void Button2_Click(object sender, EventArgs e)
{
    using (Northwind2Entities model = new Northwind2Entities())
    {
        GridView1.DataSource = model.Products
            .Where(p => p.Categories
                .CategoryId == 1)
            .OrderBy(p => p.ProductName)
            .Skip(4)
            .Take(5);
        GridView1.DataBind();
    }
}
```

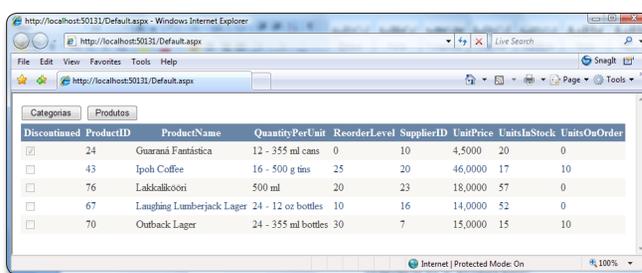


Figura 10. Resultado da query LINQ na entidade de produtos

Note que a query LINQ foi construída com uma seqüência de métodos chamados a partir da entidade *Products*. Isso é possível graças aos *Extension Methods* adicionados por conta do *LINQ to Entities*, parte do *ADO.NET Entity Framework*.

Veja que primeiramente estamos fazendo uma condição *WHERE*, filtrando apenas os produtos da categoria 1. Em seguida estamos ordenando a pesquisa pelo nome do produto. O método *Skip* está “pulando” os quatro primeiros registros do resultado, e o método *Take* está fixando a pesquisa em apenas 5 registros. Já deu pra notar que dá pra ir longe com isso, não é?

ADO.NET Data Services

O termo do momento é *Cloud Computing*, e ele basicamente se refere a uma tendência natural das aplicações e seus dados estarem cada vez mais na *nuvem*, ou *rede*, ou *internet*.

O termo é novo e vem sendo utilizado com frequência em artigos, vídeos e apresentações. É difícil definir o que é exatamente o *Cloud Computing*, como isso tudo funciona pra valer. Veja nesse ótimo vídeo <http://www.youtube.com/watch?v=6PNuQHUiV3Q>, como não há um consenso definitivo sobre o assunto.

Na nota do DevMan você vai encontrar a definição que temos sobre isso, mas o que *Cloud Computing* tem a ver com Service Pack 1? Acontece que outra *feature* que veio no SP1 é o *ADO.NET Data Services*, que está sempre relacionado à *Cloud Computing*.

A idéia básica do *ADO.NET Data Services* é disponibilizar dados na Web. Até aí nada de novo, já podíamos fazer isso desde os *WebServices* nas primeiras versões do *.NET*. O que acontece é que o *ADO.NET Data Services* é uma evolução do *WebServices*, com o objetivo claro de expor um modelo de dados, como é o caso do *Entity Framework*, como serviço na web.

Vamos ver como isso fica na prática. Clique com o botão direito no projeto *SPITestes.WebEF* e escolha a opção *Add New Item*. Como você pode conferir na **Figura 11**, selecione a categoria *Web* e escolha o template *ADO.NET Data Service* que você vai encontrar a direita. Informe *EFService.svc* em *Name* e clique em *Add*.

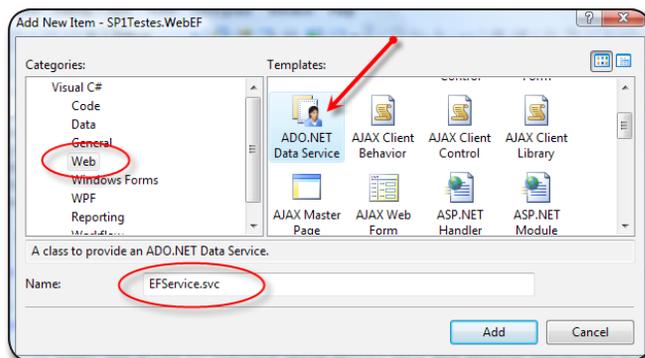


Figura 11. Criando novo serviço com ADO.NET Data Services

Além de uma série de referências que foram adicionadas ao seu projeto, um objeto chamado *EFService.svc* foi adicionado, com o seu *code-behind* aberto no *Visual Studio*, como você pode conferir na **Figura 12**.

Observe que a classe *EFService* herda de *DataService<>*, e este comentário *TODO* indica que precisamos definir aqui qual o modelo de dados que vamos expor com esse serviço. Note mais abaixo que dentro do método *InitializeService* temos que configurar as regras de acesso que definirão o que poderá ser acessado pelo serviço.

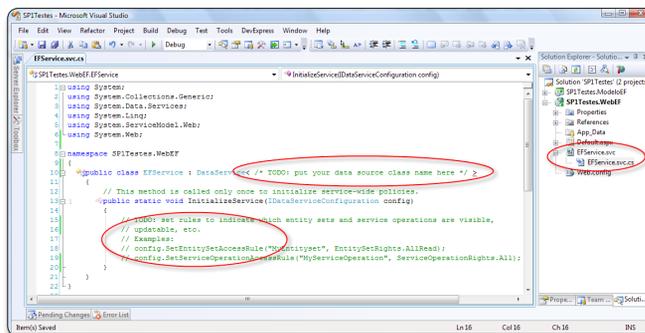


Figura 12. Codificando o Serviço EFService.svc

Sendo assim, para expormos o nosso modelo de entidades através deste serviço, modifique seu código de forma que ele fique como o demonstrado na **Listagem 5**. Note que em primeiro lugar estamos importando o namespace *SPITestes.ModelEF* para acessarmos o nosso modelo de entidades que está em outro projeto.

Em seguida, na definição da classe foi declarado que o nosso serviço *EFService* herda de *DataService<NorthWindEntities>*. Essa simples definição é o suficiente para que o serviço possa expor os dados do modelo.

E para finalizar, inclua no método *InitializeService* a configuração que dará direito de leitura à todas as entidades existentes no modelo.

Já podemos fazer nosso primeiro teste com o *ADO.NET Data Services*. Salve, compile e execute o seu projeto. Na barra de endereços do browser, troque o *Default.aspx* por *EFService.svc*, como você pode conferir na **Figura 13**.



Nota do DevMan

Cloud Computing

A computação em nuvem ou cloud computing é um modelo de computação em que dados e aplicações residem em servidores físicos ou virtuais, acessíveis por meio de uma rede em qualquer dispositivo compatível. Basicamente, consiste em compartilhar ferramentas computacionais pela interligação dos sistemas, semelhantes às nuvens no céu, ao invés de ter essas ferramentas localmente (mesmo nos servidores internos). O uso desse modelo (ambiente) é mais viável do que o uso de unidades físicas.

Um problema originado dentro das corporações é o alto custo com Tecnologia da Informação (TI). "As organizações de TI gastam hoje 80% de seu tempo com a manutenção de sistemas e não é seu objetivo de negócio manter dados e aplicativos em operação. É dinheiro jogado fora, o que é inaceitável nos dias de hoje", defende Clifton Ashley, diretor do Google para a América Latina.

Dentro desse contexto, o PC será apenas um chip ligado à internet, a "grande nuvem" de computadores. Não há necessidade de instalação de programas, serviços e armazenamento de dados, mas apenas os dispositivos de entrada (teclado, mouse) e saída (monitor) para os usuários.

Uma arquitetura em nuvem é muito mais que apenas um conjunto (embora massivo) de computadores. Ela deve dispor de uma infra-estrutura para gerenciamento, que inclua funções como provisionamento de recursos computacionais, balanceamento dinâmico do workload e monitoração do desempenho.

Definição retirada do Wikipédia

Listagem 5. Classe *EFService* que expõe o modelo *NorthwindEntities* como um serviço

```
using System;
using System.Collections.Generic;
using System.Data.Services;
using System.Linq;
using System.ServiceModel.Web;
using System.Web;

using SP1Testes.ModeloEF;

namespace SP1Testes.WebEF
{
    public class EFService : DataService
    {
        <Northwind2Entities>

        public static void InitializeService(IDataServiceConfiguration config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
        }
    }
}
```

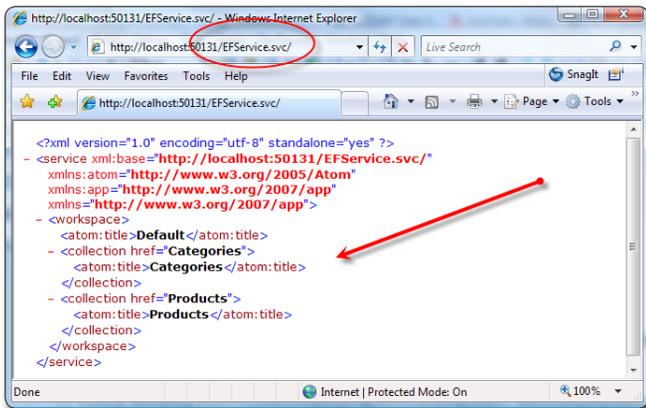


Figura 13. Acessando o *EFService.svc*

Note que o resultado é um XML nos mostrando que temos acesso às duas entidades do modelo. Vá agora em *Tools / Internet Options* do *Internet Explorer*. Na aba *Content* clique no botão *Settings* da seção *Feeds*, e desmarque a opção *Turn on Feed reading view* se esta estiver marcada.

Em seguida substitua a URL no browser por `http://localhost:50131/EFService.svc/Categories` e atualize o browser. Veja agora que você tem acesso diretamente aos dados de todas as Categorias da entidade *Categories*.

Você pode acessar uma única categoria adicionando ao final da URL o seguinte: `"/Categories(3)`". Veja o resultado na **Figura 14**.

É claro que podemos acessar esse serviço através de uma aplicação. O ideal para este teste é criarmos uma nova aplicação Web, que dessa vez irá se chamar *SP1Testes.WebDS*. Depois de criado o novo projeto, a primeira coisa que devemos fazer é adicionar uma referência ao serviço que criamos anteriormente.

Para adicionar uma referência a um serviço do *ADO.NET Data Services*, clique com o botão direito sobre o projeto na *Solution Explorer* e escolha a opção *Add Service Reference*. Essa é uma nova opção que temos no *Visual Studio 2008*, e que você pode conferir aqui na **Figura 15**.

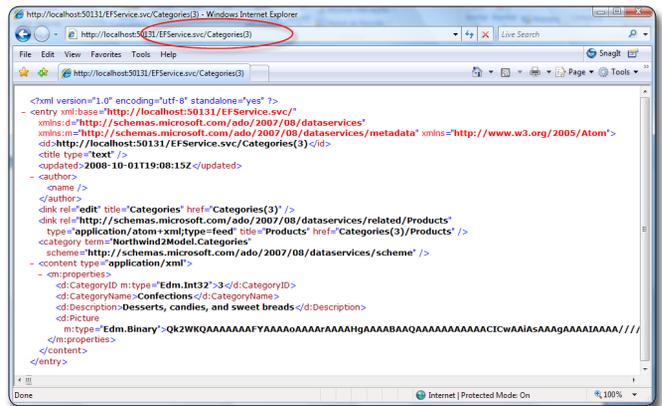


Figura 14. Acessando os Dados da Categoria 3 através do *EFService.svc*

Em seguida irá aparecer uma janela para adicionarmos a referência ao serviço. Você pode digitar a URL do serviço no campo *Address*, ou simplesmente clicar em *Discover / Services in Solution*, como mostra a **Figura 16**. Essa opção irá exibir o nosso serviço *EFService.svc*. Em seguida, clique em *OK* para que a referência seja adicionada ao nosso projeto.

Para testarmos o acesso ao *EFService*, crie uma interface na página *Default.aspx* igual à que fizemos no projeto *SP1Testes.WebEF*, pode inclusive copiar a mesma interface. Em seguida, no *code-behind*, inclua o código da **Listagem 6**.

Veja nessa listagem que estamos disparando exatamente as mesmas pesquisas que fizemos direto no modelo no exemplo anterior. A diferença é que estamos acessando o modelo através do *EFService.svc*.

Salve, compile e execute o seu projeto. Veja que os resultados são os mesmos obtidos nas **Figuras 9** e **10**. Isso é o básico que o *ADO.NET Data Services* nos oferece para tornar nossos dados disponíveis na nuvem!

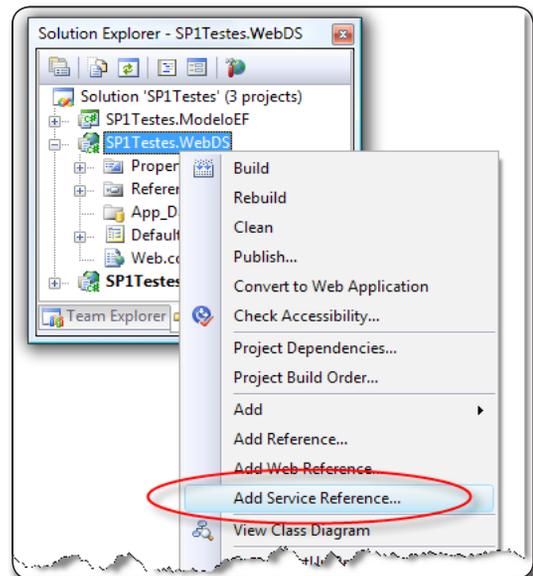


Figura 15. Opção *Add Service Reference*

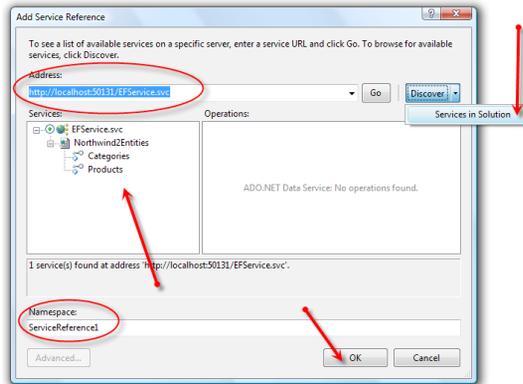


Figura 16. Adicionando uma referência ao Serviço EFService.svc

Listagem 6. Código para consultar dados através do EFService.svc

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace SPITestes.WebDS
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            ServiceReference1.Northwind2Entities model = new
                SPITestes.WebDS.ServiceReference1.Northwind2Entities(
                    new Uri("http://localhost:50131/EFService.svc/"));

            GridView1.DataSource = model.Categories;
            GridView1.DataBind();
        }

        protected void Button2_Click(object sender, EventArgs e)
        {
            ServiceReference1.Northwind2Entities model = new
                SPITestes.WebDS.ServiceReference1.Northwind2Entities(
                    new Uri("http://localhost:50131/EFService.svc/"));

            GridView1.DataSource = model.Products.
                Where(p => p.Categories.CategoryID == 1).
                OrderBy(p => p.ProductName).
                Skip(4).
                Take(5);

            GridView1.DataBind();
        }
    }
}
```

ASP.NET Dynamic Data

O *ASP.NET Dynamic Data* é uma ferramenta de alta produtividade que teve seu *release* no Service Pack 1. Com ele podemos criar as funcionalidades CRUD (funcionalidades para manutenção de uma tabela, como inserir, alterar etc.) de uma aplicação Web em questão de segundos.

O *Dynamic Data* já teve alguns *previews* antes do SP1, mas somente agora podemos utilizá-lo em conjunto com o *ADO.NET Entity Framework*. Para isso você deve criar um novo projeto na *Solution*, e escolher o template *Dynamic Data Entities Web Application*.

Note, como mostra a **Figura 17**, que temos dois *templates* que levam o nome *Dynamic Data*: o *Dynamic Data Web Application*, que deve ser utilizado com um modelo de classes do *LINQ to SQL*, e o *Dynamic Data Entities Web Application*, que é utilizado juntamente com o *ADO.NET Entity Framework*. Nós vamos utilizar a segunda opção. Informe *SPITestes.WebDynamic* em *Name* e clique em *OK* para criar o novo projeto.

.Net ModelKit Suite



Componentes incluídos:

► **Report Sharp-Shooter** - é um gerador de relatórios .Net nativo e permite ao desenvolvedor criar relatórios ricos em recursos e integra-los a aplicações WinForms e WebForms.

► **OLAP ModelKit** - o produto fornece a habilidade de se processar, analisar e representar os dados sob forma de relatórios OLAP quando você desejar.

► **Instrumentation ModelKit** - um componente arredondador de bordas para a criação de painéis digitais, KPI's e outros aplicativos de BI para monitorar em tempo real dados críticos.

► **Chart ModelKit** - a primeira solução gráfica com o real WYSIWYG designer e um modelo de vinculação de dados avançado que não requer codificação.

A assinatura de 1 ano do .NETModelKit Suite inclui:

- Quatro componentes .NET: Instrumentation ModelKit, Chart ModelKit, OLAP ModelKit e Report Sharp-Shooter
- Atualizações do produto, melhorias e correções
- Novas versões do produto
- Suporte técnico gratuito
- **DESCONTO PARA BRASILEIROS 40 % !!!!!**



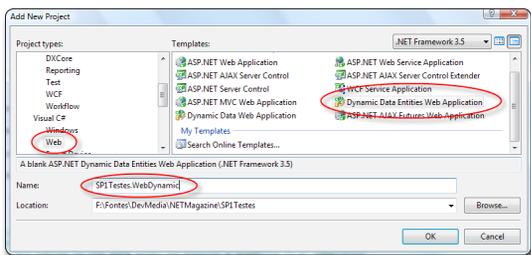


Figura 17. Criando um novo Projeto com o template Dynamic Data Entities Web Application

Listagem 7. Global.asax do SP1Testes.WebDynamic

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;
using System.Xml.Linq;
using System.Web.Routing;
using System.Web.DynamicData;

namespace SP1Testes.WebDynamic
{
    public class Global : System.Web.HttpApplication
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            MetaModel model = new MetaModel();
            model.RegisterContext(typeof(SP1Testes.ModeloEF.Northwind2Entities),
                new ContextConfiguration() { ScaffoldAllTables = true });

            routes.Add(new DynamicDataRoute("{table}/{action}.aspx")
            {
                Constraints = new RouteValueDictionary(new { action = "List|Details|Edit|Insert" }),
                Model = model
            });

            routes.Add(new DynamicDataRoute("{table}/ListDetails.aspx")
            {
                Action = PageAction.List,
                ViewName = "ListDetails",
                Model = model
            });
        }

        void Application_Start(object sender, EventArgs e)
        {
            RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Nestes exemplo poderemos aproveitar o modelo de entidades que foi criado inicialmente, no projeto *Class Library*. Para isso será necessário adicionar uma referência a este projeto, no projeto *SP1Testes.WebDynamic* que acabamos de criar. Também será necessário incluir uma referência ao namespace *System.Data.Entity*.

E por fim, no arquivo *Web.config* vamos precisar configurar a *connectionString*, a mesma que configuramos no primeiro exemplo deste artigo.

Com tudo isso feito, basta apenas configurarmos a nossa aplicação Web para que acesse o modelo de entidades. Isso deve ser feito

no arquivo *Global.asax*, que você confere aqui na **Listagem 7**.

Note que a principal configuração é feita na chamada do método *RegisterContext*, que identifica o nosso modelo como o "contexto" a ser utilizado pelo *Dynamic Data*. Nesta mesma linha temos configurado a propriedade *ScaffoldAllTables* igual a *true*. Isso é importante para que as funcionalidades CRUD sejam geradas para todas as tabelas do modelo.

É isso, já podemos testar! Configure este projeto como o projeto de startup, salve, compile e execute. Na primeira página que será aberta você vai encontrar um menu com as opções *Categories* e *Products*, as duas entidades do nosso modelo.

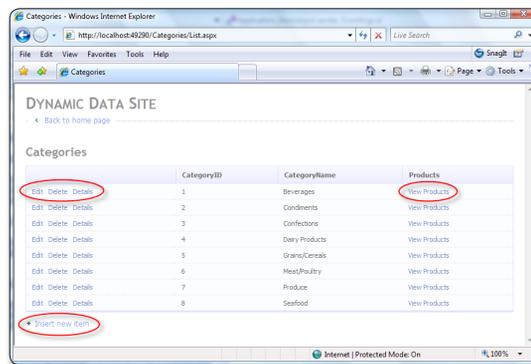


Figura 18. Testando o ASP.NET Dynamic Data

Clique em *Categories*, e veja como mostra a **Figura 18**, que podemos realizar todas as funcionalidades CRUD (*Insert, Update, Delete e Select*).

Divirta-se! Faça todos os testes possíveis, nas duas entidades. Depois, inclua outras entidades ao modelo e veja o que acontece. Esse é só o básico do *ASP.NET Dynamic Data*, junto com isso você ainda vai encontrar uma série de possibilidades para personalizar estes sites que são criados em tempo de execução.

Conclusão

Como eu disse no início deste artigo, estas não são as únicas novidades do Service Pack 1. Eu adicionaria nesta lista o *AJAX Browser History*, e o controle *EntityDataSource* para *ASP.NET*, que é o equivalente do *LinqDataSource* porém para acesso ao modelo de entidades do *ADO.NET Entity Framework*. Espero que tenham gostado do artigo, grande abraço e até a próxima! ●

Dê seu feedback sobre esta edição!

A .NET Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/netmagazine/feedback

