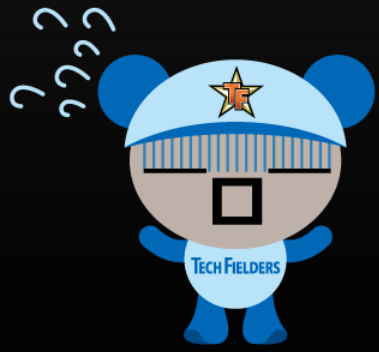
Tech Fielders セミナー

Silverlight を用いた業務アプリケーション開発入門

マイクロソフト株式会社
デベロッパーエバンジェリスト
小高太郎(こだかたろう)
taro.kodaka@microsoft.com
http://blogs.msdn.com/tarok/



アジェンダ

- イントロダクション
- まずはサンプルシステムの説明を・・・
- ポイント解説
  - データサービス実装
  - データバインド
  - データの追加削除
  - (オプション) EDMを利用したWCFサービス呼び出しによるストアドプロシージャ実行とトランザクションの実装

まずは、
サンプルシステムの
説明を・・・

サンプルアプリケーション

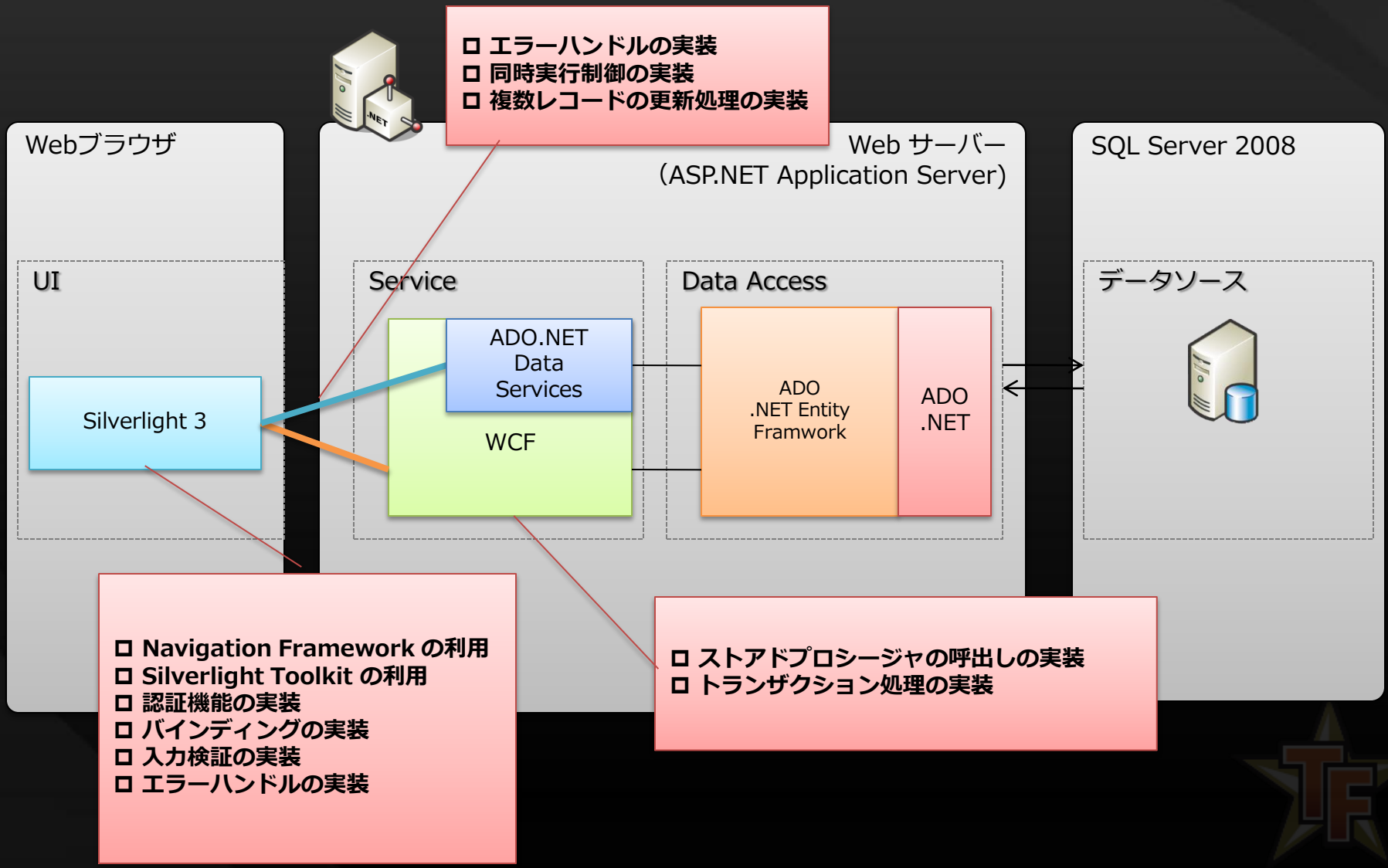
- データ処理を伴う Silverlight 3 アプリケーション
  - ADO.NET Entity Framework
  - ADO.NET Data Services
  - WCF
  - SQL Server 2008
- Visual Studio のみで作成可能
  - 見た目の考慮は最小限
  - Expression Blend 未使用

- Silverlight を用いた業務アプリケーションを構築する場合の典型例として提示
  - 今回は Web ショッピングサイト

開発環境

- Visual Studio 2008 Service Pack1
- Visual Studio 2008 SP1 用 Microsoft Silverlight 3 Tools
- Silverlight Toolkit
- Microsoft SQL Server 2008 (Express 以上のエディション)

Silverlightを使用した Webショッピングサイト概略図



データサービス実装

サービスによるデータアクセス

- Silverlight ではデータアクセスでサービスの利用が必要
- サービスはすべて非同期処理
- 利用できるサービスと選択基準
  - WCF or ADO.NET Data Services

ADO.NET Data Services の利用

- RESTful
  - URI によるアクセス
    - CURD 、ページング、並び変え、フィルタ、変更管理、早期ロード、遅延ロードが可能

メリット

- クライアントの変更管理が可能
- サービス参照のエンドポイントが一つ
  - クライアントの設定ファイル管理が容易

デメリット

- 複雑な処理 (複数テーブルのデータ取得、1:N レコードの一括更新)では実装が冗長

WCF の利用

- SOAP
  - CRUD に関わらず、様々な処理をメソッドとして公開可能

メリット

- 複雑な処理 （異なるデータソースのマージ結果の取得）などが実装可能

デメリット

- クライアントの変更管理を実装する必要がある
- サービス参照のエンドポイントが多くなる可能性がある

画面の構築：ListBox の追加

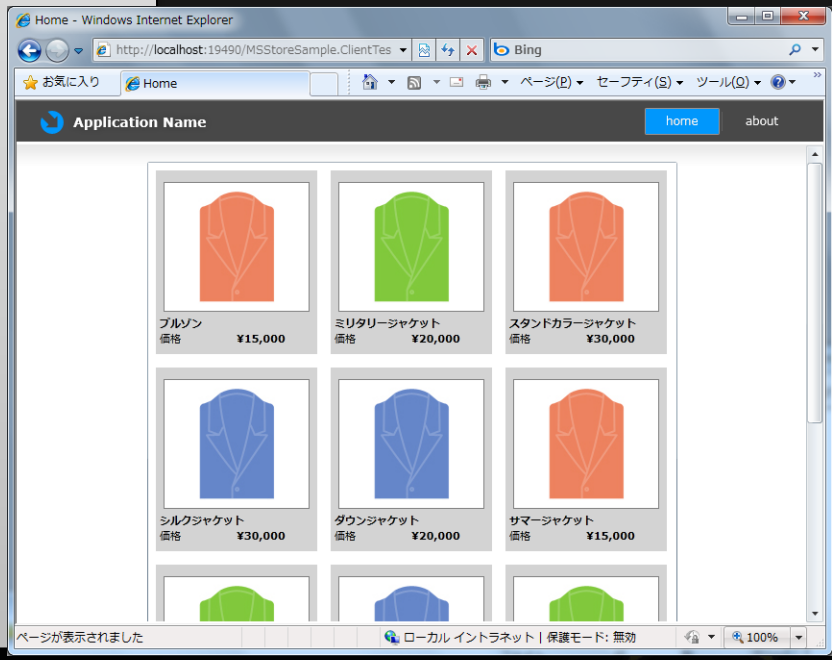
Home.xaml

```
<!--プロダクトリスト-->
<ListBox x:Name="productListBox" Margin="24,0,24,0"
      ItemsPanel="{StaticResource HorizonalWrapPanel}"
      ScrollViewer.HorizontalScrollBarVisibility="Disabled">
   <ListBox.ItemTemplate>
      <DataTemplate>
         <Border Margin="4" Background="LightGray">
            <Grid Width="160" Margin="4,8">
               <Grid.RowDefinitions>
                  <RowDefinition />
                  <RowDefinition />
                  <RowDefinition />
               </Grid.RowDefinitions>

               <Border Margin="4" BorderBrush="Gray" BorderThickness="1" Background="White">
                  <Image x:Name="productImage"
                        Source="{Binding ProductThumbnailUrl,
                                 Converter={StaticResource
                                 UriToBitmapImageConverter}}"
                        Grid.Row="0"
                        Stretch="None"
                        Margin="8"    />
               </Border>

               <TextBlock Text="{Binding Name}"
                        Grid.Row="1"
                        TextWrapping="Wrap"
                        FontWeight="Bold" />

               <Grid Grid.Row="2">
                  <Grid.ColumnDefinitions>
                     <ColumnDefinition/>
                     <ColumnDefinition/>
                  </Grid.ColumnDefinitions>
                  <TextBlock Text="価格" Grid.Column="0" />
                  <TextBlock Text="{Binding Price,
                                 Converter={StaticResource FormattingConverter},
                                 ConverterParameter=¥{0:C¥}}"
                                 Grid.Column="1"
                                 FontWeight="Bold"/>
               </Grid>
            </Grid>
         </Border>
      </DataTemplate>
   </ListBox.ItemTemplate>
</ListBox>
```

画像情報の表示
（相対パス）のフォーマット

商品情報の表示

価格情報の表示
（金額情報）のフォーマット



データの取得

Home.xaml.cs

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    MSStoreSampleEntities context = MSStoreSampleEntities.CreateNoTracking();
    var productsQuery = (from product in context.DefaultColorProducts
                         where product.CategoryId == 1
                         select product) as DataServiceQuery<DefaultColorProducts>;

    productsQuery.BeginExecute(DefaultColorProductsQueryCompleted, productsQuery);
}

private void DefaultColorProductsQueryCompleted(IAsyncResult result)
{
    Dispatcher.BeginInvoke(() =>
    {
        DataServiceQuery<DefaultColorProducts> query = result.AsyncState as DataServiceQuery<DefaultColorProducts>;
        productListBox.ItemsSource = query.EndExecute(result);
    });
}
```

LINQでのデータ取得

非同期処理
コールバックメソッドの実装

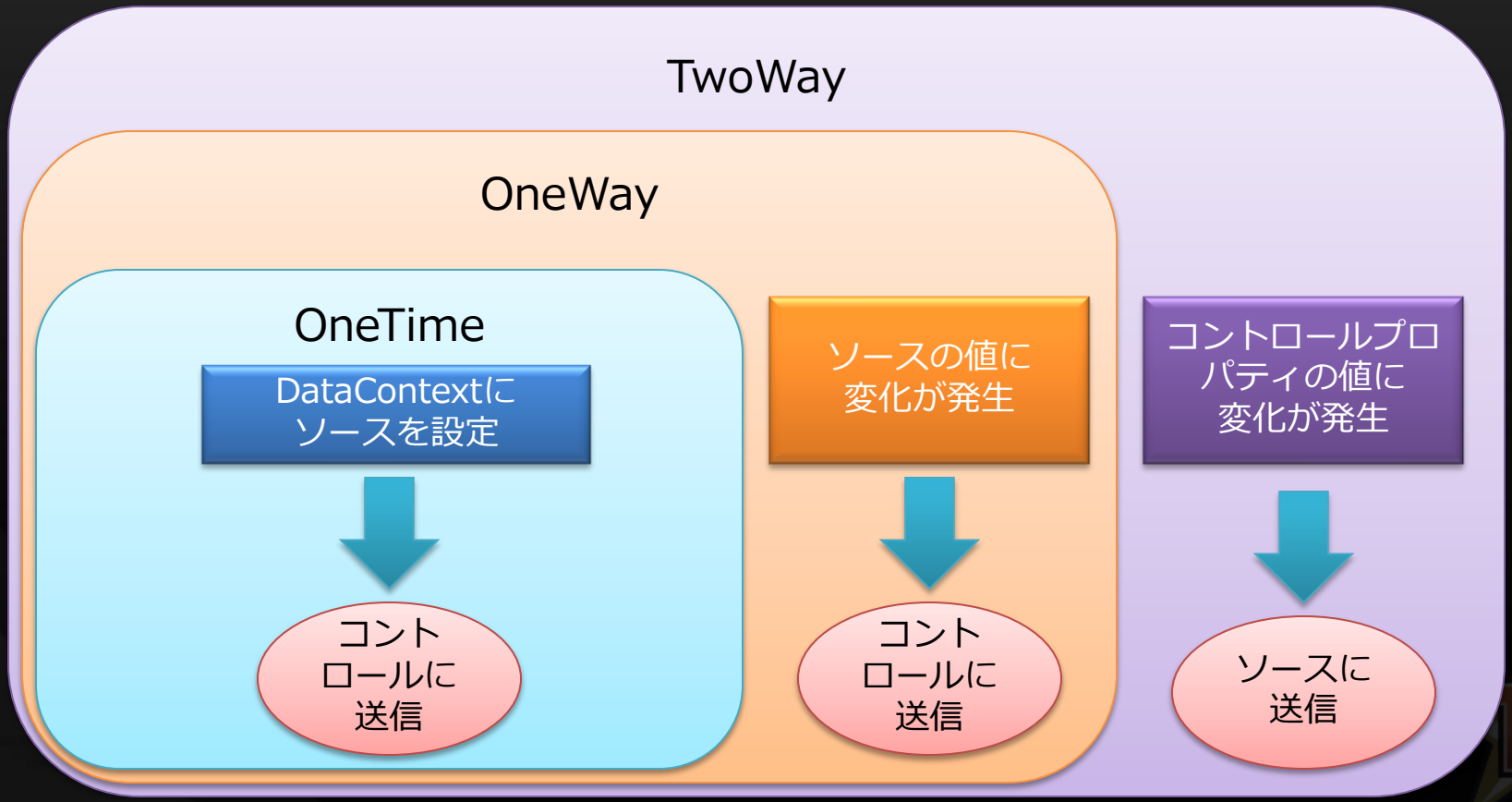
データバインド

データバインド

- XAMLベースのバインド機能
  - System.Windows.FrameworkElement
    - Page, UserControl を含む多くのコントロールで可能
  - 依存関係プロパティ
    - コントロールに設定してあるバインド可能なプロパティ
  - DataContext
    - 上位にあるXAML要素から継承される
  - バインディングモード

バインディングモード

- 変更点の通知
  - INotifyPropertyChanged インターフェイスの実装
    - PropertyChanged イベントの発行で変更点の通知が可能

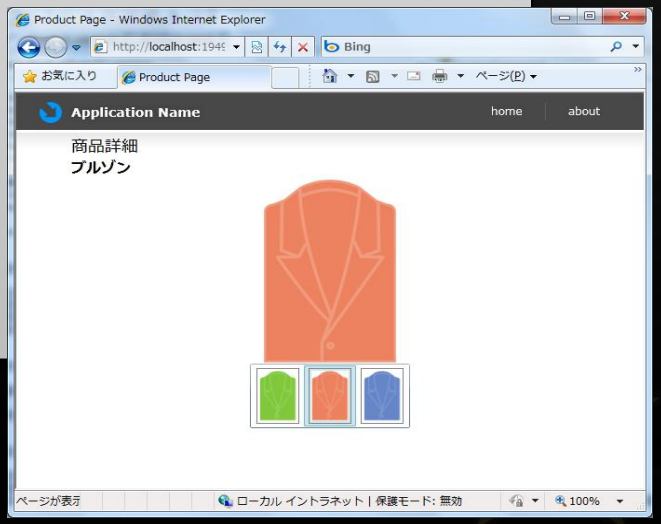


画面の構築：ListBox の追加

Product.xaml

```
<TextBlock Text="商品詳細" FontSize="18" />
<TextBlock x:Name="productNameTextBlock"
        Text="{Binding Product.Name}"
        FontSize="16" FontWeight="Bold" />
<Image x:Name="productDetailImage"
            Source="{Binding ProductImageUrl,
                        Converter={StaticResource UriToBitmapImageConverter}}"/>
<ListBox x:Name="colorListBox"
           ItemsPanel="{StaticResource HorizonalWrapPanel}"
           ScrollViewer.HorizontalScrollBarVisibility="Disabled"
           SelectionChanged="colorListBox_SelectionChanged"
              ItemsSource="{Binding ProductColors}"
              SelectedItem="{Binding ProductColor, Mode=TwoWay}"
              VerticalAlignment="Center"
              HorizontalAlignment="Center"
              Margin="0">
   <ListBox.ItemTemplate>
     <DataTemplate>
       <Border BorderBrush="Gray" BorderThickness="1" Margin="2,0">
         <Image Source="{Binding ProductThumbnailUrl,
                        Converter={StaticResource UriToBitmapImageConverter}}"
                Margin="2" Width="40" />
       </Border>
     </DataTemplate>
   </ListBox.ItemTemplate>
</ListBox>
```

TwoWayバインドの実装



商品詳細エンティティの実装

ProductDetailContainer.cs

インターフェイスの実装

変更の通知

```
public class ProductDetailContainer : INotifyPropertyChanged
{
    private string productImageUrl;
    private Products product;
    private ProductColors productColor;
    private IEnumerable<ProductColors> productColors;
    public event PropertyChangedEventHandler PropertyChanged;

    public IEnumerable<ProductColors> ProductColors
    {
        get { return this.productColors; }
        set
        {
            if (this.productColors != value)
            {
                this.productColors = value;
                PropertyChanged(this, new PropertyChangedEventArgs("ProductColors"));
                this.ProductColor = productColors.FirstOrDefault(color => color.ProductColorId == product.DefaultColorId);
            }
        }
    }

    public ProductColors ProductColor
    {
        get { return this.productColor; }
        set
        {
            if (this.productColor != value)
            {
                this.productColor = value;
                this.ProductImageUrl = productColor.ProductImageUrl;
                PropertyChanged(this, new PropertyChangedEventArgs("ProductColor"));
            }
        }
    }

～後略～
```

データバインドの利用

Product.xaml.cs

```
productDetail = new ProductDetailContainer();
LayoutRoot.DataContext = productDetail;

var productQuery = (from product in context.Products
                    where product.ProductId == productId
                    select product) as DataServiceQuery<Products>;
productQuery.BeginExecute(ProductQueryCompleted, productQuery);

var colorsQuery = (from color in context.ProductColors
                   where color.ProductId == productId
                   select color) as DataServiceQuery<ProductColors>;
colorsQuery.BeginExecute(ProductColorsQueryCompleted, colorsQuery);
```

ProductDetailContainer の利用

取得データの宣言
（OnNavigatedToイベント）

Product.xaml.cs

```
private void ProductQueryCompleted(IAsyncResult result)
{
    Dispatcher.BeginInvoke(() =>
    {
        DataServiceQuery<Products> query = result.AsyncState as DataServiceQuery<Products>;
        productDetail.Product = query.EndExecute(result).FirstOrDefault();
    });
}

private void ProductColorsQueryCompleted(IAsyncResult result)
{
    Dispatcher.BeginInvoke(() =>
    {
        DataServiceQuery<ProductColors> query = result.AsyncState as DataServiceQuery<ProductColors>;
        productDetail.ProductColors = query.EndExecute(result).ToList();
    });
}
```

ProductDetail(Container)
へのデータ充填

上から呼ばれる
非同期メソッド

Product.xaml.cs

```
private void colorListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    productDetail.ProductImageUrl = productDetail.ProductColor.ProductImageUrl;
}
```

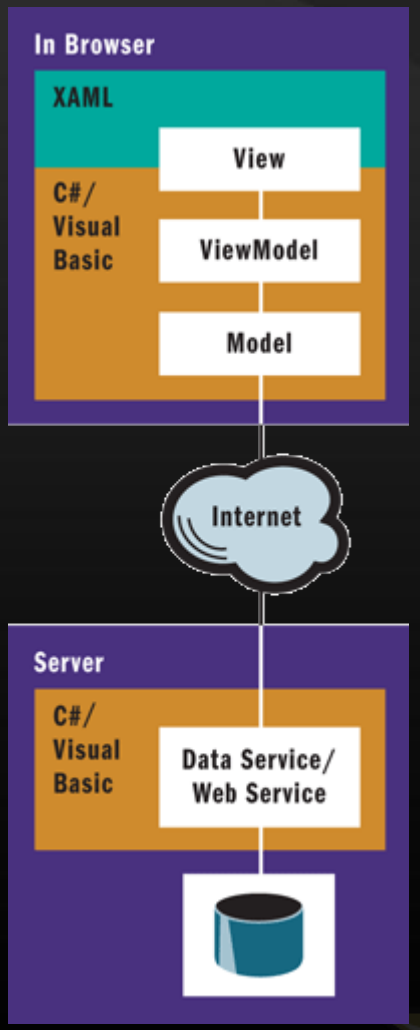
TwoWayバインドの利用

Model-View-ViewModel パターン（参考）

一歩進んだ Silverlight ソリューション

今回のサンプルでは処理の簡略化のために未実装

- WPF、Silverlight の疎結合ソリューションのパターン
- 下記の実装を行う
  - Model
    - Data(Web)Service のエンティティ(をラップする)
  - ViewModel
    - Model を UI に合わせて公開する
  - View
    - ViewModel をXAML等でバインドする



参照：
http://msdn.microsoft.com/ja-jp/magazine/dd458800.aspx

データの追加削除

ADO.NET Data Services でのデータの追加/更新/削除

- コンテキストの異なる Entity を対象にする場合
  - Attach
    - コンテキストの変更管理に含める
  - SetLink
    - エンティティ同士が関連している場合の認識の追加
- 複数レコード更新処理が可能
- 楽観同時実行制御可能

データの追加

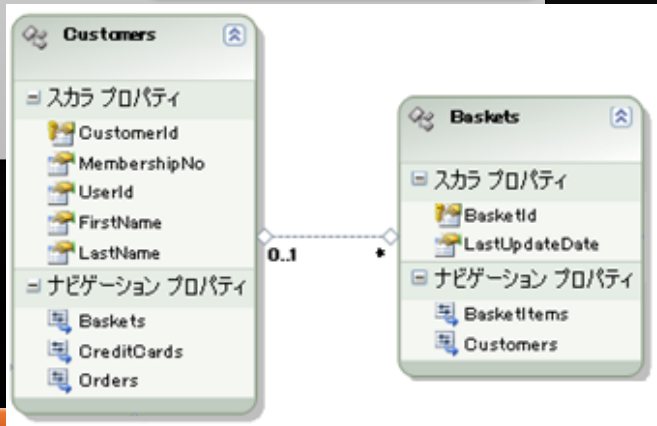
別のコンテキストから取得した Entity を追加対象にするため Attach する

Basket.xaml.cs

Baskets と Customer はオブジェクトグラフ構造をとり、別のコンテキストから取得した Entity を対象にするため SetLink する

```
Baskets  basket = new Baskets()
{
    BasketId = Guid.NewGuid(),
    Customers = AuthenticationContext.Current.User.Customer,
    LastUpdateDate = DateTime.Now
};
context.AddToBaskets(basket);
context.AttachTo("Customers", AuthenticationContext.Current.User.Customer);
context.SetLink(basket, "Customers", AuthenticationContext.Current.User.Customer);

  // Data Services による更新確定
    context.BeginSaveChanges(SaveChangesOptions.Batch, OnAddBasketSaveChanged, context);
}
private void OnAddBasketSaveChanged(IAsyncResult result)
{
    Dispatcher.BeginInvoke(() =>
    {
        MSStoreSampleEntities context = result.AsyncState as MSStoreSampleEntities;
        DataServiceResponse response = context.EndSaveChanges(result);
        Application.Current.Resources.Remove("ProductDetail");
     });
}
```

商品をバスケットに追加
（Basket.xaml
OnNavigatedToイベント）



データの削除

選択された商品情報

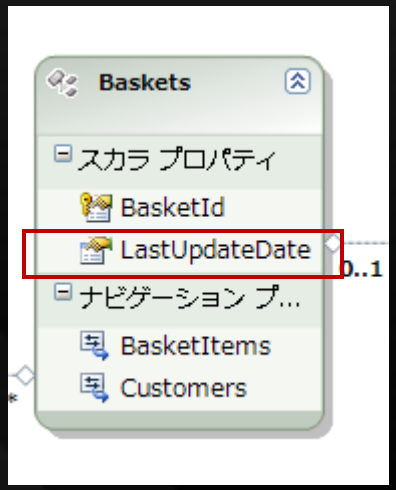
Basket.xaml.cs

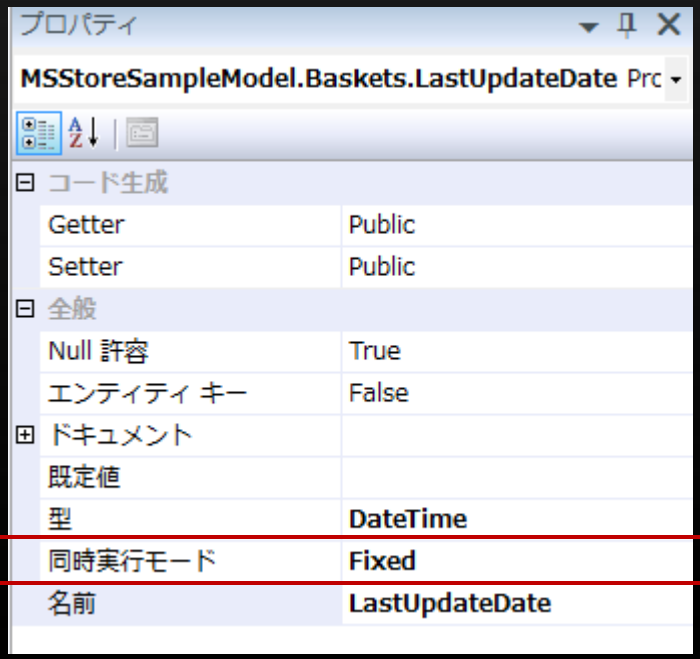
```
private void basketItemDeleteButton_Click(object sender, RoutedEventArgs e)
{
    Button button = sender as Button;
    BasketItems item = button.DataContext as BasketItems;
    context.DeleteObject(item);
    context.BeginSaveChanges(OnDeleteBasketItemSaveChanged, context);
}

private void OnDeleteBasketItemSaveChanged(IAsyncResult result)
{
    Dispatcher.BeginInvoke(() =>
    {
        MSStoreSampleEntities context = result.AsyncState as MSStoreSampleEntities;
        var deletedDescripters = context.GetChanges(EntityStates.Deleted);
        context.EndSaveChanges(result);
        Baskets basket = basketPanel.DataContext as Baskets;
        foreach (var descripter in deletedDescripters)
        {
            BasketItems item = descripter.Entity as BasketItems;
            basket.BasketItems.Remove(item);
        }
        basketPanel.DataContext = null;
        basketPanel.DataContext = basket;
    });
}
```

商品の削除ボタン押下
（Basket.xaml
basketItemDeleteButton_Click
イベント）

楽観同時実行制御（参考）

- エンティティの同時実行モードを Fixed にする
- 内部的にはレスポンスデータの Etag 値を利用
  - 更新処理を行い並列の違反があれば、DataServiceRequest が発生



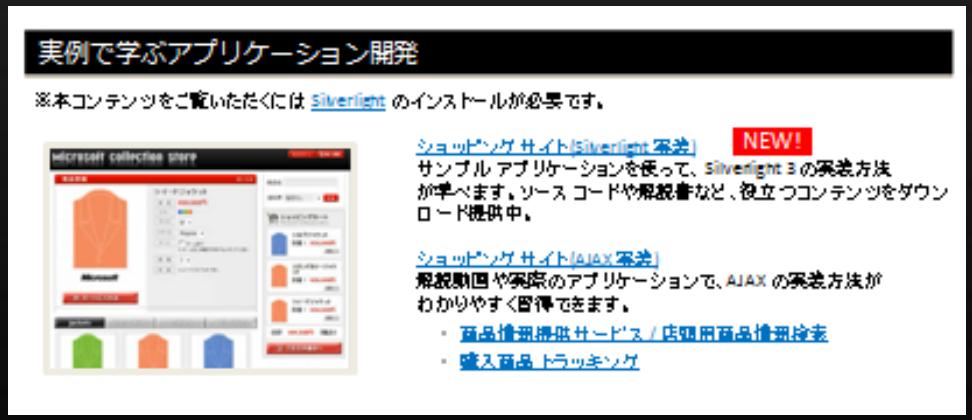


最後に

- 本日のサンプル

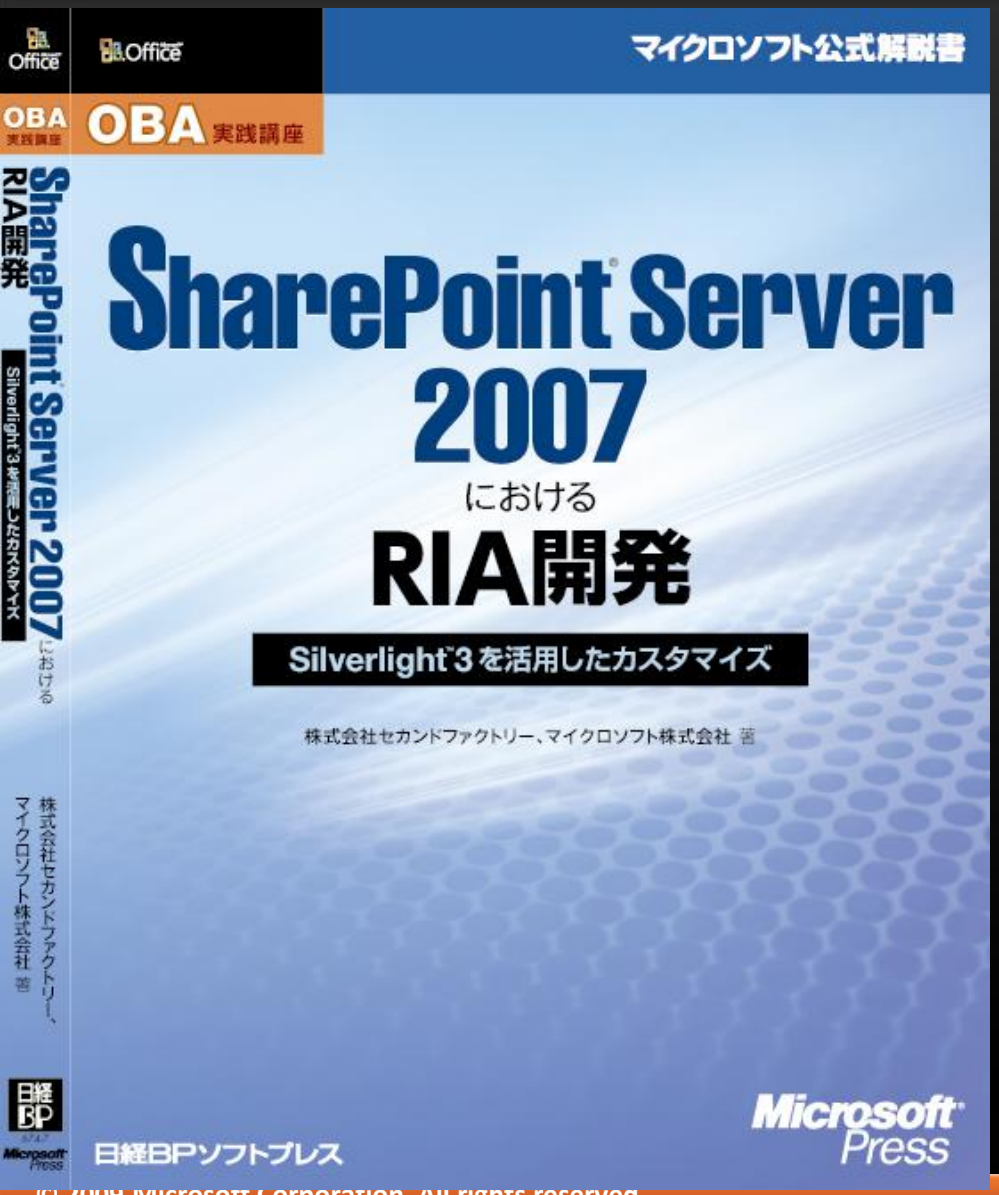
.NET 開発サンプル コード集 Code Recipe

http://msdn.microsoft.com/ja-jp/samplecode.recipe.aspx



- 順次シナリオを公開しています
  - データモデル、データサービスの作成
  - データ取得
  - Silverlight ToolKitの利用
  - データ表示（コンバーター）
  - 画面のナビゲート
  - データバインド
  - 入力データの検証
  - ADO.NET Data ServicesとSilverlightによるエラーハンドリング
  - 関連のある複数エンティティからのデータ取得
  - オンデマンドな認証処理の実装
  - データ処理（追加更新）
  - EDMを利用したWCFサービス呼び出しによるストアドプロシージャ実行とトランザクションの実装

Silverlight 3 と SharePoint 開発



- タイトル：
  OBA実践講座
  SharePoint Server 2007における RIA開発
  *Silverlight3*を活用したカスタマイズ
- 出版：
  日経BPソフトプレス
- ISBN：
  978-4-89100-674-7
- 定価：
  3,570円（税込み）
- ***2010*年*1*月（予定）**

オプション

アプリケーションの開発

EDMを利用したWCFサービス呼び出しによるストアドプロシージャ実行とトランザクションの実装

ストアドプロシージャを呼び出す為に必要な準備

- サーバーサイド
  - Entity Data Model にストアドプロシージャを登録
    - モデルブラウザで関数インポート
  - EntityClient 上でストアドプロシージャを呼び出す
    - 現状 ObjectServices 対応のコードは自動生成されない
    - 自動トランザクションの指定が可能
  - Silverlight 対応の WCF サービスとして公開
- クライアントサイド
  - Silverlight アプリケーションからの呼び出しは非同期の実装

サービスの実装

OrderServices.svc.cs

```
[OperationContract]
public void CreateOrder(string userId, int paymentType, Guid creaditCardId)
{
    using (MSStoreSampleEntities context = new MSStoreSampleEntities())
    {
        context.Connection.Open();
        using (DbTransaction transaction = context.Connection.BeginTransaction())
        {
            // 注文を作成します
            using (DbCommand command = context.Connection.CreateCommand())
            {
                command.Transaction = transaction;
                command.CommandType = CommandType.StoredProcedure;
                command.CommandText = "MSStoreSampleEntities.CreateOrder";
                command.Parameters.Add(
                    new EntityParameter("UserId", DbType.String) { Value = userId });
                command.Parameters.Add(
                    new EntityParameter("PaymentType", DbType.Int32) { Value = paymentType });
                command.Parameters.Add(
                    new EntityParameter("CreditCardId", DbType.Guid) { Value = creaditCardId });
                command.Parameters.Add(
                    new EntityParameter("ReturnValue", DbType.Int32) { Direction = ParameterDirection.ReturnValue });
                command.ExecuteNonQuery();
                int? returnValue = ((int?)command.Parameters["ReturnValue"].Value);
            }
            // 注文が完了したら、バスケットを削除します
            using (DbCommand command = context.Connection.CreateCommand())
            {
                command.Transaction = transaction;
                command.CommandType = CommandType.StoredProcedure;
                command.CommandText = "MSStoreSampleEntities.DeleteBasket";
                command.Parameters.Add(
                    new EntityParameter("UserId", DbType.String) { Value = userId });
                command.Parameters.Add(
                    new EntityParameter("ReturnValue", DbType.Int32) { Direction = ParameterDirection.ReturnValue });
                command.ExecuteNonQuery();
                int? returnValue = ((int?)command.Parameters["ReturnValue"].Value);
            }
            transaction.Commit();
        }
    }
}
```

EntityClient 上での実装

トランザクションの考慮

Silverlight (クライアント) からの呼び出し

Basket.xaml.cs

```
private void paymentButton_Click(object sender, System.Windows.RoutedEventArgs e)
{
    OrderServiceClient client = new OrderServiceClient();
    client.CreateOrderCompleted += CreateOrderCompleted;
    client.CreateOrderAsync(AuthenticationContext.Current.User.UserName, 3, Guid.Empty);
}

void CreateOrderCompleted(object sender, System.ComponentModel.AsyncCompletedEventArgs e)
{
    if (e.Error != null)
        throw e.Error;
    NavigationService.Navigate(new Uri("/Home", UriKind.RelativeOrAbsolute));
}
```

注文処理ボタン押下
(Basket.xaml
paymentButton_Click(
イベント)

非同期処理
コールバックメソッドの実装