えムナウのC#
プログラミングのページ

Microsoft MVP Most Valuable Professional

ineta

Linq
プラットホームベンダーの情熱(を感じる)

えムナウ(児玉宏之)

Microsoft MVP for Visual-Developer C# 2005/01-2007/12

アジェンダ

- はじめに
- Linqの概要
- C#3.0とLinqの関係
- Entity Data Model
- まとめ

はじめに

- データを簡単にオブジェクトとして扱いたい

Microsoftはこれまでに何をやったか

- DataSetを作ってみた
  - データベースをメモリ上で再現できたがObjectじゃない
- DataSetデザイナとか作ってみた
  - 手軽で便利になったけどObjectじゃない
- Partial で拡張できるようにした
  - DataSet、DataTable、DataRow、TableAdapterに、拡張できてプロパティとかメソッドとか作れるようにはなってObjectっぽくなってきたけど作るのは大変。

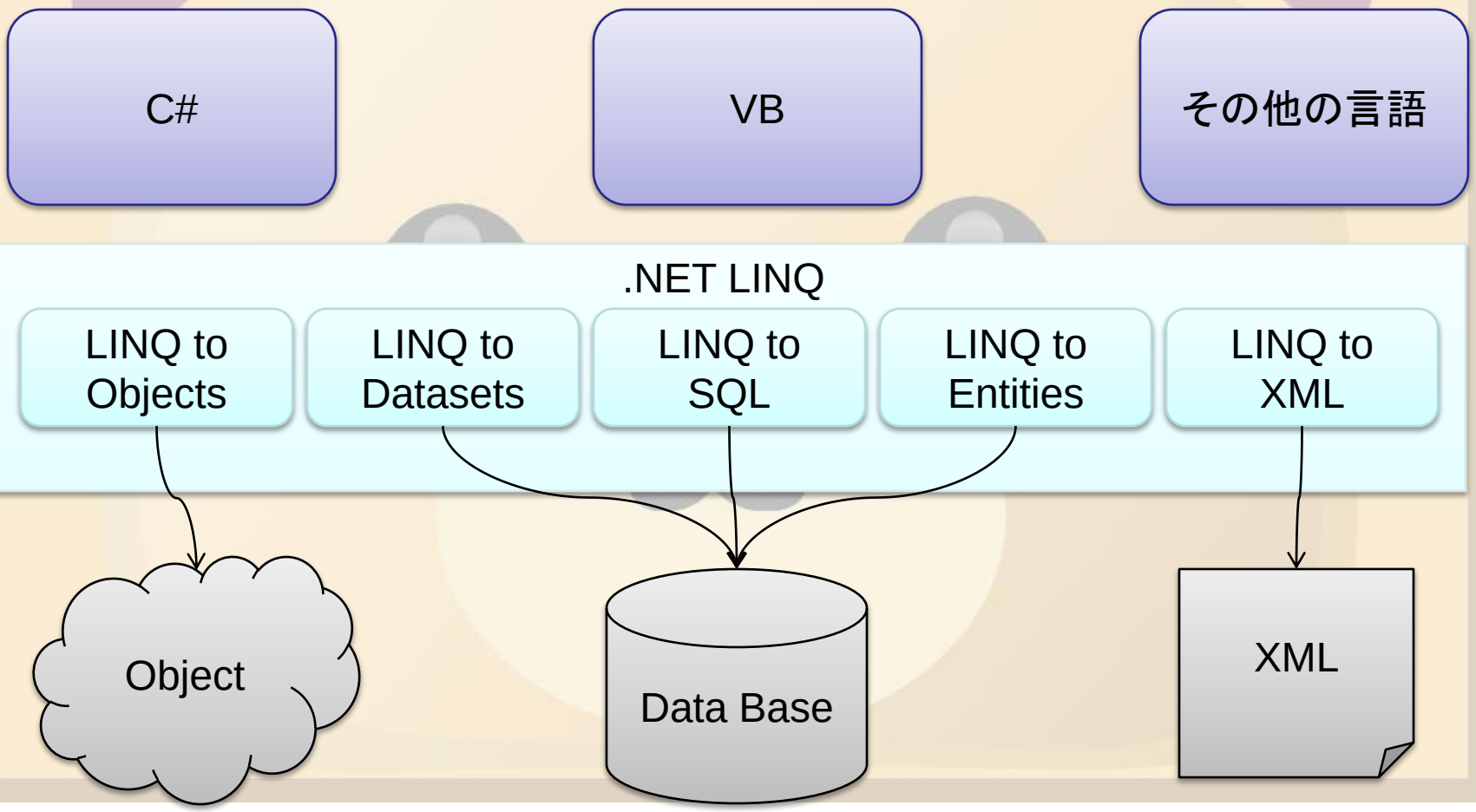
Linq概要

- Linqとは

  .Net Framework上の「言語に統合されたクエリ」(Language-INtegrated Query)の拡張セットを指すコードネームの名称。

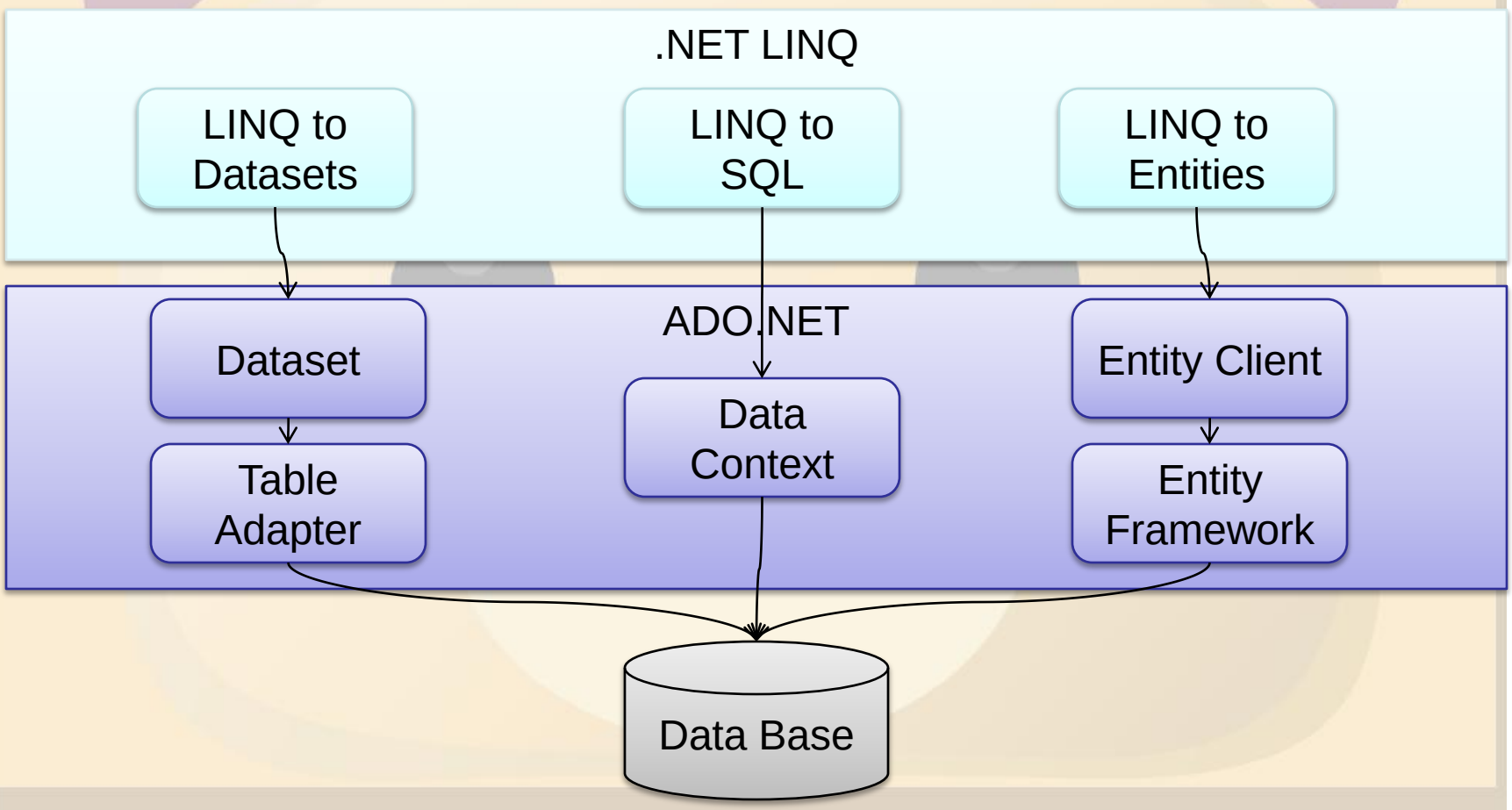
Linq概要

- Linqの種類



Linq概要

- データベース周りのLinq



Linq概要

- Linq to Objects

  IEnumerable<T> ベースのすべての情報ソースにクエリを適用

```
var al = new [] {
    new {Name="hnaka",ZipCode="553-0001",Prefecture="大阪府"},
    new {Name="hkodama",ZipCode="168-0064",Prefecture="東京都"}
};
var accounts = from a in al
               where a.ZipCode == "168-0064"
               select new { a.Name, a.ZipCode };
foreach (var account in accounts)
{
    Console.WriteLine(account.Name+ "(" + account.ZipCode + ")");
}
```

Linq概要

- Linq to DataSet

  従来のADO.NETのDataSetのすべての情報ソースにクエリを適用

```
DataSet al = new DataSet();
testTableAdapter.Fill(al);
var accounts = from a in al.Account
               where a.ZipCode == "168-0064"
               select new { Name = a.Name, ZipCode = a.ZipCode };
foreach (var account in accounts)
{
    Console.WriteLine(account.Name+ "(" + account.ZipCode + ")");
}
```

Linq概要

- Linq to SQL

  SQLサーバーのデータベースのすべての情報ソースにクエリを適用

```
using (DataClasses1DataContext db = new DataClasses1DataContext
            (Linq1.Properties.Settings.Default.TESTConnectionString))
{
    var accounts = from a in db.Account
                where a.ZipCode == "168-0064"
                select a;
     foreach (Account account in accounts)
    {
        Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
    }
}
```

Linq概要

- Linq to Entities

  Entity Framework を通した概念エンティティの情報ソースにクエリを適用

```
var accounts = from a in textContext.Account
               where a.ZipCode == "168-0064"
               select a;
foreach (Account account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

Linq概要

- Linq to XML

  XMLのXElementのすべての情報ソースにクエリを適用

```
var al = new XElement(“Account”, new XAttribute(“CanCode”, true),
new XElement(“Name”, “ hkodama”), new XElement(“ZipCode”, “168-0064”),
new XElement(“Prefecture”, “東京都”));
var accounts = from a in al
               where a.ZipCode == "168-0064"
               select new Account { Name = a.Attribute(“Name”),
                ZipCode = a.Attribute("ZipCode") };
foreach (var account in accounts)
{
   Console.WriteLine(account.Name+ "(" + account.ZipCode + ")");
}
```

Linq概要

- まとめ
  - 一般に.NetFrameworkでデータとして扱うすべてが対象

名称	対象
Linq to Objects	IEnumerable<T>
Linq to DataSet	ADO.NETのDataSet
Linq to SQL	SQLサーバーのデータベース
Linq to Entities	Entity Framework を通した概念エンティティ
Linq to XML	XMLのXElement

C#3.0とLinqの関係

- C#3.0の言語拡張
  - 暗黙に型付けされたローカル変数
  - 拡張メソッド
  - ラムダ式
  - オブジェクト初期化子および コレクション初期化子
  - 匿名型
  - クエリ式
  - パーシャルメソッド

C#3.0とLinqの関係

- C#2.0で書いてみた

```
List<MyAccount> al = new List<MyAccount>();
al.Add(new MyAccount("hnaka", "553-0001", "大阪府"));
al.Add(new MyAccount("hkodama", "168-0064", "東京都"));

IEnumerable<MyAccount2> accounts =
    EnumerableExtensions<MyAccount, MyAccount2>.Select(
        EnumerableExtensions<MyAccount, MyAccount2>.Where
            (al, delegate(MyAccount a) { return a.ZipCode == "168-0064"; }),
        delegate(MyAccount a) { return new MyAccount2(a.Name, a.ZipCode); }
    );

Console.WriteLine("C#2.0");
foreach (MyAccount2 account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

C#3.0とLinqの関係

```
public class MyAccount
{
    public MyAccount()
    {
        _name = "";
        _zipcode = "";
        _prefecture = "";
    }
    public MyAccount(string name, string
zipcode, string prefecture)
    {
        _name = name;
        _zipcode = zipcode;
        _prefecture = prefecture;
    }
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    private string _zipcode;

    public string ZipCode
    {
        get { return _zipcode; }
        set { _zipcode = value; }
    }
    private string _prefecture;

    public string Prefecture
    {
        get { return _prefecture; }
        set { _prefecture = value; }
    }
}
```

C#3.0とLinqの関係

```
public class MyAccount2
{
    public MyAccount2()
    {
        _name = "";
        _zipcode = "";
    }
    public MyAccount2(string name, string
zipcode)
    {
        _name = name;
        _zipcode = zipcode;
    }
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    private string _zipcode;

    public string ZipCode
    {
        get { return _zipcode; }
        set { _zipcode = value; }
    }
}
```

C#3.0とLinqの関係

```
static class EnumerableExtensions
<TS, TD>
{
    public delegate TD SelectFunc(TS t);

    public static IEnumerable<TD> Select
     (IEnumerable<TS> e, SelectFunc f)
    {
        foreach (TS i in e)
        {
            yield return f(i);
        }
    }
```

```
    public static IEnumerable<TS> Where
    (IEnumerable<TS> e, Predicate<TS> p)
    {
        foreach (TS i in e)
        {
            if (p(i)) yield return i;
        }
    }
}
```

C#3.0とLinqの関係

- 暗黙に型付けされたローカル変数

```
var al = new List<MyAccount>();
al.Add(new MyAccount("hnaka", "553-0001", "大阪府"));
al.Add(new MyAccount("hkodama", "168-0064", "東京都"));

var accounts =
    EnumerableExtensions<MyAccount, MyAccount2>.Select(
        EnumerableExtensions<MyAccount, MyAccount2>.Where
            (al, delegate(MyAccount a) { return a.ZipCode == "168-0064"; }),
        delegate(MyAccount a) { return new MyAccount2(a.Name, a.ZipCode); }
    );

Console.WriteLine("C#3.0 暗黙に型付けされたローカル変数");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

C#3.0とLinqの関係

- 拡張メソッド

```
var al = new List<MyAccount>();
al.Add(new MyAccount("hnaka", "553-0001", "大阪府"));
al.Add(new MyAccount("hkodama", "168-0064", "東京都"));


var accounts = al
    .Where(delegate(MyAccount a) { return a.ZipCode == "168-0064"; })
    .Select(delegate(MyAccount a) { return new MyAccount2(a.Name, a.ZipCode); });

Console.WriteLine("C#3.0 拡張メソッド");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

C#3.0とLinqの関係

```
static class EnumerableExtension
{
    public delegate TD SelectFunc<TS, TD>(TS t);
    public static IEnumerable<TD> Select<TS, TD>
                (this IEnumerable<TS> e, SelectFunc<TS, TD> f)
    {
        foreach (TS i in e)
        {
            yield return f(i);
        }
    }
    public static IEnumerable<TS> Where<TS>
                (this IEnumerable<TS> e, Predicate<TS> p)
    {
        foreach (TS i in e)
        {
            if (p(i)) yield return i;
        }
    }
}
```

C#3.0とLinqの関係

- ラムダ式

```
var al = new List<MyAccount>();
al.Add(new MyAccount("hnaka", "553-0001", "大阪府"));
al.Add(new MyAccount("hkodama", "168-0064", "東京都"));


var accounts = al
    .Where(a => a.ZipCode == "168-0064")
    .Select(a => new MyAccount2(a.Name, a.ZipCode));

Console.WriteLine("C#3.0 ラムダ式");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

C#3.0とLinqの関係

- オブジェクト初期化子および コレクション初期化子

```
var al = new List<MyAccount> {
    new MyAccount{Name="hnaka",ZipCode="553-0001",Prefecture="大阪府"},
    new MyAccount{Name="hkodama",ZipCode="168-0064",Prefecture="東京都"}
};

var accounts = al
    .Where(a => a.ZipCode == "168-0064")
    .Select(a => new MyAccount2 { Name = a.Name, ZipCode = a.ZipCode });

Console.WriteLine("C#3.0 オブジェクト初期化子および コレクション初期化子");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

C#3.0とLinqの関係

- 匿名型

```
var al = new [] {
    new {Name="hnaka",ZipCode="553-0001",Prefecture="大阪府"},
    new {Name="hkodama",ZipCode="168-0064",Prefecture="東京都"}
};

var accounts = al
    .Where(a => a.ZipCode == "168-0064")
    .Select(a => new { Name = a.Name, ZipCode = a.ZipCode });

Console.WriteLine("C#3.0 匿名型");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

C#3.0とLinqの関係

- クエリ式

```
var al = new[] {
    new {Name="hnaka",ZipCode="553-0001",Prefecture="大阪府"},
    new {Name="hkodama",ZipCode="168-0064",Prefecture="東京都"}
};

var accounts = from a in al
               where a.ZipCode == "168-0064"
               select new { Name = a.Name, ZipCode = a.ZipCode };
Console.WriteLine("C#3.0 クエリ式");
foreach (var account in accounts)
{
    Console.WriteLine(account.Name + "(" + account.ZipCode + ")");
}
```

C#3.0とLinqの関係

- パーシャルメソッド
  - Linqでカスタムプロパティ検証や挿入・更新・削除メソッドの検証に使用する

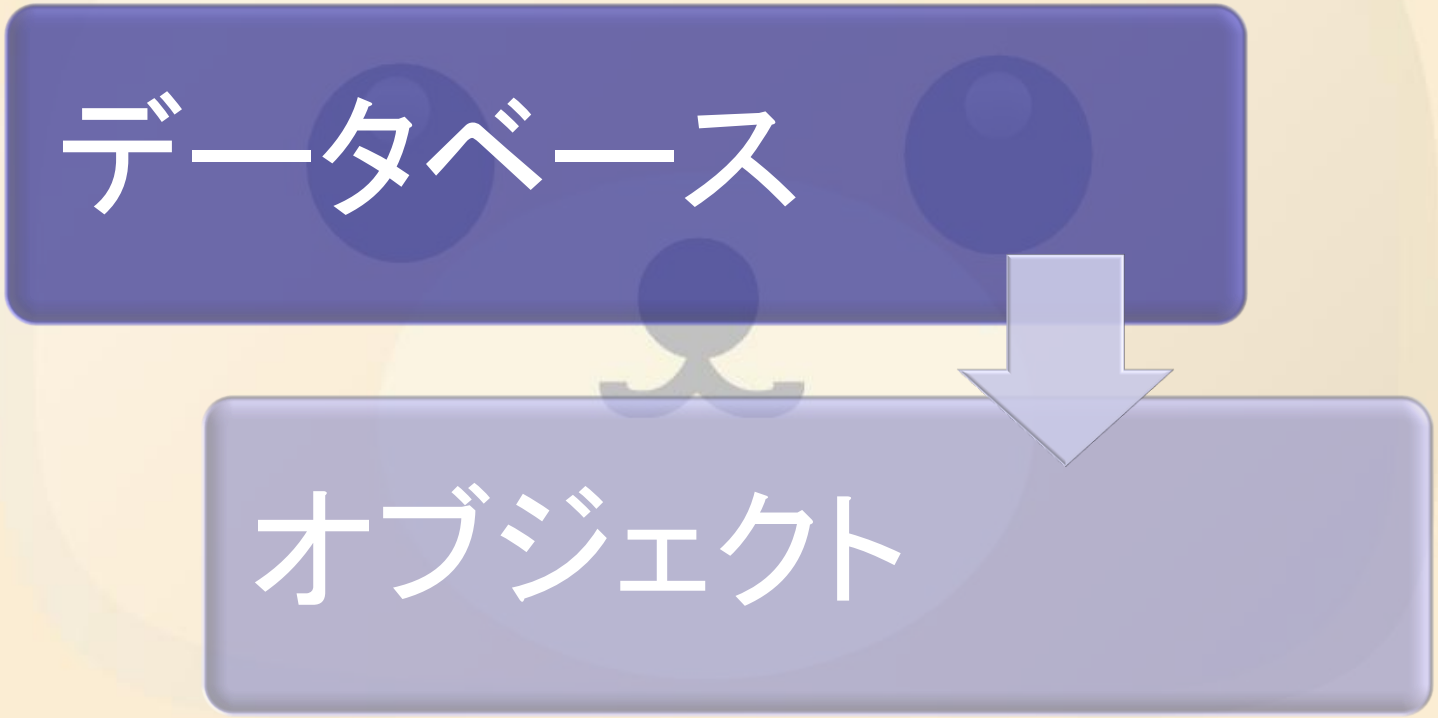
```
public partial class DataClasses1DataContext
{
      public partial class Account
     {
            partial void OnZipCodeChanging(string value)
            {
                  Regex zipcheck = new Regex(@“^[0-9]{3}-[0-9]{4}$”);
                  if (!zipcheck.IsMatch(value)) throw new Excepton(“郵便番号エラー”);
            }
     }
}
```

C#3.0とLinqの関係

- 結論
  - 言語拡張はLinqのためだった
  - 便利に短く書けるようになった
  - from から先に書く
  - 型推論が働くので入力が簡単
  - SQL文とは順序が逆なので注意

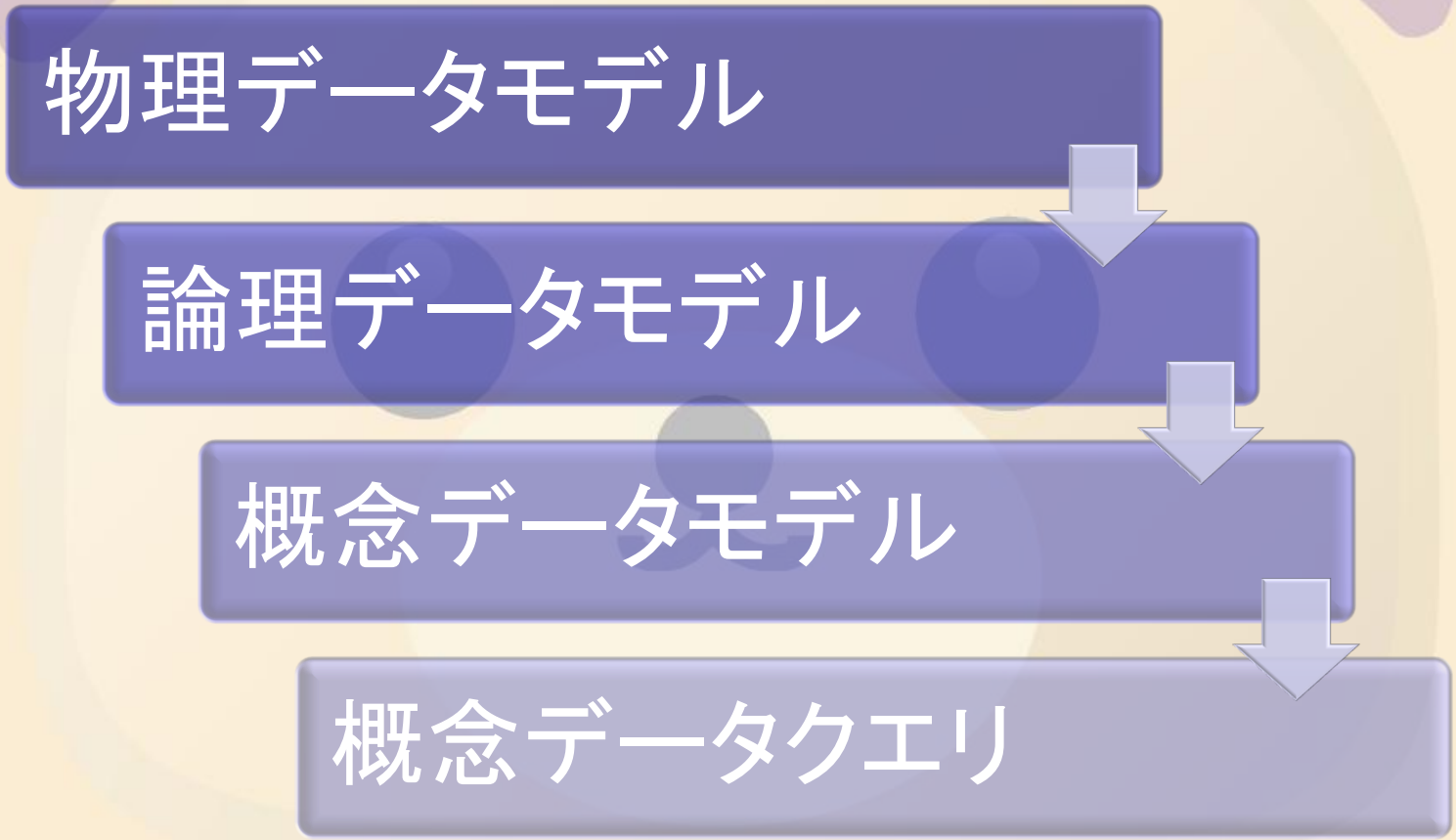
Entity Data Model

- Linq は O/Rマッピングか？
  - O/Rマッピングはこんな感じ、じゃぁそうじゃん。



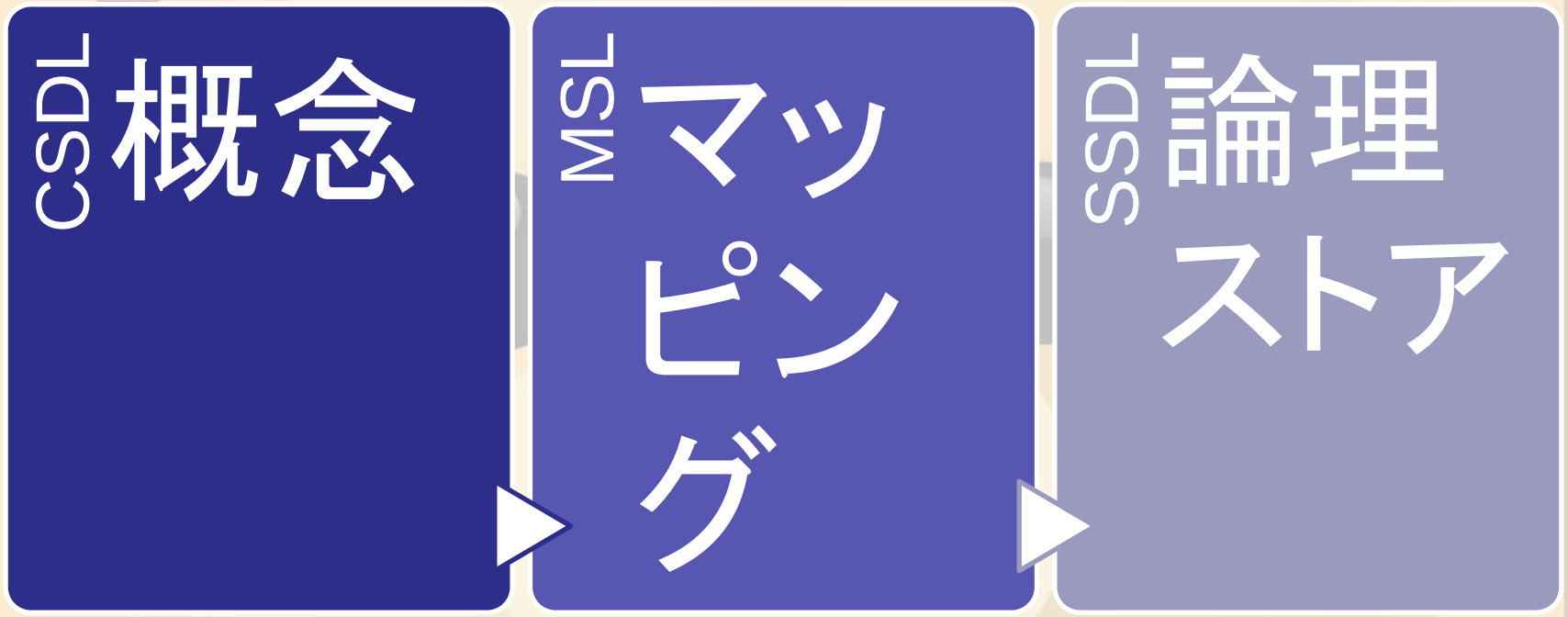
Entity Data Model

– Linq to Entities はこんな感じ



Entity Data Model

- Linq to Entities の構造



Entity Data Model

- 概念モデルは、概念スキーマ定義言語 (CSDL: Conceptual Schema Definition Language) を使用して XML ファイルに定義
- データベース スキーマを表す論理モデルは、ストア スキーマ定義言語 (SSDL: Store Schema Definition Language) を使って XML ファイルに定義
- マッピング層は、マッピング スキーマ言語 (MSL : Mapping Schema Language) を使用して定義

Entity Data Model

- 業務データモデル開発の流れ
  - 業務を分析して概念モデルを作る
    - データの内容
    - データの分類と命名
    - データの操作
  - 実際のデータベースに合わせ論理モデルを作る
    - エンティティ（項目）
    - ドメイン（データ型）
    - キー
    - スーパータイプ・サブタイプ

Entity Data Model

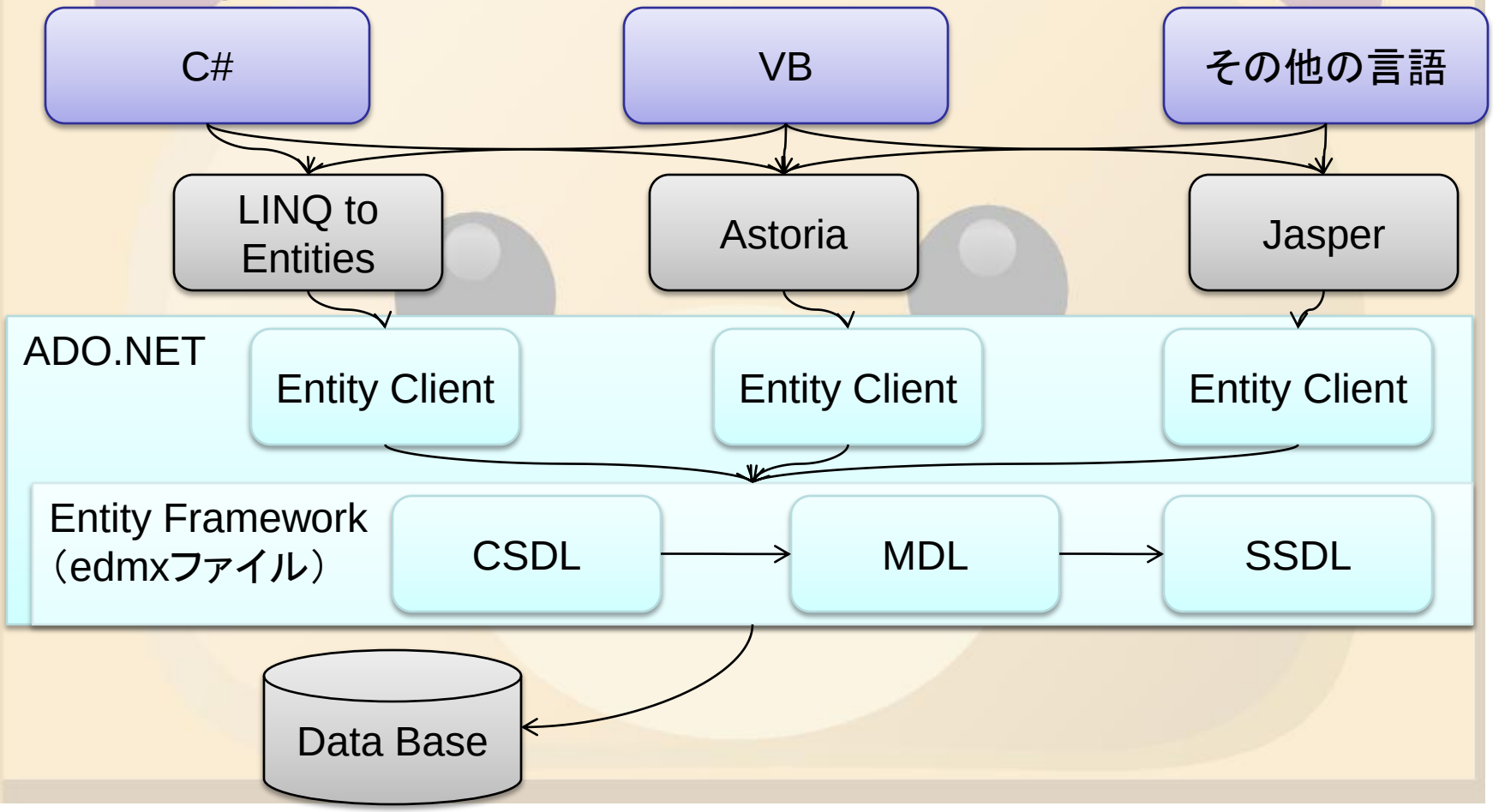
- 業務データモデル開発の流れ
  - 実際のデータベースに適用する
    - プライマリーキー
    - リレーション
    - 基本の型に合わせる
    - インデックス
    - トリガ
    - 容量試算や配置の考慮

Entity Data Model

- ソフトウェアエンジニアはデータベースに対してプログラムを作ってきた
  - これからはデータベースに向き合ってプログラムを作る時代ではなくなった
  - 業務概念そのものに向き合ってプログラムを作る時代になっていく

Entity Data Model

- Entity Framework の新技術たち



Entity Data Model

- 結論
  - 業務概念そのものを相手にした設計が必要
  - 概念レベル設計をしないデータベース設計がまかり通っているが概念レベル設計したほうがいい
  - 概念レベル設計をきちんとやっておくとプログラムも作りやすくなる
  - 新しい技術も生み出されてきている

まとめ

- Microsoftはプラットフォームベンダーとしてデータの扱い方について試行錯誤しながら今日まで来た
- Linq を通じてデータの扱い方を統一することにした
- Linq to SQL を使ってO/Rマッピングについて考えた
- Linq to Entities でその上位概念からとらえることにした

おつかれさまでした

参考になるページ

http://blogs.wankuma.com/chicasharp/

http://msdn.microsoft.com/msdnmag/issues/07/06/CSharp30/default.aspx?loc=jp

http://msdn.microsoft.com/msdnmag/issues/07/07/DataPoints/default.aspx?loc=jp

http://www.event-registration.jp/events/te07/special_session.htm