Core 2.0

Itaú bba | aadanelon

documento de arquitetura

arquiteto: Alexandre Alves Danelon

Rev. 1.0

Core 2.0

O Core consiste em um **start kit** construído para **.NET Framework 4.x** com objetivo de padronizar as necessidades comuns de uma grande aplicação Web como:

* Persistência de Dados em uma base relacional
* Materialização dos Dados
* Controle transacional e gerenciamento de conexões ao Banco de Dados
* Suporte a Fluxo de negócio utilizando múltiplos Bancos de Dados
* Suporte à conexão com serviços externos e legado
* Log e Diagnósticos referente a erros e auditoria da aplicação
* Interfaces para validação de regras de negocio

O Core foi pensado para trabalhar em um cenário distribuído e escalável orientado a serviços que atenda aos requisitos mínimos de performance e Log.

A seguir temos um conjunto de Interfaces e classes abstratas criadas no Core 1.x que tornaram-se obsoletas no Core 2.0.

* **~~EntityData~~** (utilizada para mapeamento da tabela)
* **~~CollectionData~~** (utilizada para retorno de uma lista do tipo EntityData)
* **~~EntityManager~~** (utilizada para persistência de uma tabela)
* **~~CollectionManager~~** (utilizada para materialização de uma tabela)

E substituição as interfaces **obsoletas**, foram criadas novas interfaces que trazem mais recursos com menos esforço de implementação.

# Componente responsável pelo gerenciamento de Dados

Classes responsáveis pela comunicação e gerenciamento de dados com **SQL Server.**

## IEntityDataModel

Interface comum de mapeamento entre Objeto Entidade e Tabela de Banco de Dados, no qual deve ser obrigatoriamente utilizada para materialização ou e persistência de dados.

A seguir temos um exemplo pratico de utilização.

|  |
| --- |
| public class tbagi017IpiCamo\_ENT : EntityDataManager, IDataEntity, IEntityDataModel  {  #region data  public int IPICampo { get; set; }  public string CodTipoCampo { get; set; }  public string NomeIPILayoutCampo { get; set; }  public string TextoValorPadrao { get; set; }  public string TextoFormatoCampo { get; set; }  public int TamanhoCampo { get; set; }  public int NumeroOrdemCampo { get; set; }  public string CodigoIdentificacaoCampo { get; set; }  public string IndicadorCampoComplementar { get; set; }  public string IndicadorDominioCorporativo { get; set; }  public string TextoNomeDominio { get; set; }  public string NomeIPILayoutCampoTrabalho { get; set; }  #endregion      public string MapperTable()  {  return "TBAGI017\_IPI\_CAMO";  }  public Dictionary<string, string> MapperColumn()  {  return new Dictionary<string, string>() {  {"IPICampo", "IDT\_IPI\_CAMO"},  {"CodTipoCampo", "COD\_TIPO\_CAMO"},  {"TamanhoCampo", "NUM\_TAMA\_CAMO"},  {"NomeIPILayoutCampo", "NOM\_IPI\_LAYO\_CAMO"},  {"TextoValorPadrao" , "TXT\_VLR\_PADR"},  {"TextoFormatoCampo", "TXT\_FORT\_CAMO"},  {"NumeroOrdemCampo", "NUM\_ORDE\_CAMO"},  {"CodigoIdentificacaoCampo", "COD\_IDEF\_CAMO"},  {"IndicadorCampoComplementar", "IND\_CAMO\_COPL"},  {"IndicadorDominioCorporativo", "IND\_DOMN\_CORO"},  {"TextoNomeDominio", "TXT\_NOM\_DOMN"},  {"NomeIPILayoutCampoTrabalho", "NOM\_IPI\_LAYO\_CAMO\_TRAB"}  };  }  public Dictionary<string, string> MapperKeys()  {  return new Dictionary<string, string>() {  {"IPICampo", "IDT\_IPI\_CAMO"}  };  }  public bool MapperKeyIsNotIdentity()  {  return false;  } |

## EntityDataManager

Utilizada para dar suporte a comportamento de persistência sobre os métodos de **Save** e **Delete** para uma Entidade que já implementa a **IEntityDataModel**.

**OBS.:** O **Save** se encarrega automaticamente de determinar se o objeto enviado será inserido ou atualizado na base de dados diante das propriedades chaves informadas.

A seguir temos um exemplo pratico de utilização:

|  |
| --- |
| public class tbagi017IpiCamo\_ENT : EntityDataManager, IDataEntity, IEntityDataModel  {  public override void Save(TransactionManager transactionContext)  {  Save(this, transactionContext);  }  public override void Save()  {  Save(this);  }  public override void Delete(TransactionManager transactionContext)  {  Delete(this, transactionContext);  }  public override void Delete()  {  Delete(this);  }  } |

## CollectionDataManager

Utilizada para recuperação e materialização de uma coleção de Objetos que implementam a interface **IEntityDataModel** na qual expõe os métodos **Find** e **Get** sobre implentação dos Critérios de pesquisa equivalente a serem implementados pelo utilizador.

**Find** : Utilizado para materializar uma coleção de objeto que implementa **IEntityDataModel**.

**Get** : Utilizado para obter um objeto que implementa **IEntityDataModel.**

A seguir temos um exemplo de utilização.

|  |
| --- |
| public class tbagi019ParmPrsoIpi\_COL : CollectionDataManager<tbagi019ParmPrsoIpiKey, tbagi019ParmPrsoIpi\_ENT>  {  protected override Criteria<CriteriaInfo, object> CriteriaFind(tbagi017IpiCamoKey objectKey)  {  var \_pesquisa = new Criteria<CriteriaInfo, object>(new tbagi017IpiCamo\_ENT());  if (!String.IsNullOrEmpty(objectKey.IndicadorDominioCorporativo))  {  \_pesquisa.Add(new CriteriaInfo()  {  LikePattern = LikePattern.Both,  LogicalType = LogicalOperator.AND,  NameProperty = "TextoNomeDominio",  SearchType = LogicalComparator.EQUAL  }, objectKey.IndicadorDominioCorporativo);  }  return \_pesquisa;  }  protected override Criteria<CriteriaInfo, object> CriteriaGet(tbagi017IpiCamoKey objectKey)  {  var \_pesquisa = new Criteria<CriteriaInfo, object>(new tbagi017IpiCamo\_ENT());  switch (objectKey.consulta)  {  case tbagi017IpiCamoKey.enTipoConsulta.Um:  \_pesquisa.Add(new CriteriaInfo()  {  LikePattern = LikePattern.Both,  LogicalType = LogicalOperator.AND,  NameProperty = "IPICampo",  SearchType = LogicalComparator.EQUAL  }, objectKey.idtIpiCamo);  break;  default:  break;  }  return \_pesquisa;  }  } |

## CollectionQueryableManager

Além de todos os recursos e funcionalidades herdadas da **CollectionDataManager®,**  a **CollectionQueryableManager** permite a implementação de uma Query complexa com múltiplas tabelas e joins, na qual permite ser recuperada através do método **FindQuery.**

A seguir temos um exemplo de utilização.

|  |
| --- |
| public class tbagi017IpiCamo\_COL : CollectionQueryableManager<tbagi017IpiCamoKey, tbagi017IpiCamo\_ENT>  {  protected override string Query(tbagi017IpiCamoKey objectKey)  {  var complexSQL = new StringBuilder("SELECT \n");  complexSQL.Append("tbagi017.IDT\_IPI\_CAMO IDT\_IPI\_CAMO, \n");  complexSQL.Append("tbagi017.COD\_TIPO\_CAMO COD\_TIPO\_CAMO, \n");  complexSQL.Append("tbagi017.NOM\_IPI\_LAYO\_CAMO NOM\_IPI\_LAYO\_CAMO, \n");  complexSQL.Append("tbagi017.NUM\_TAMA\_CAMO NUM\_TAMA\_CAMO, \n");  complexSQL.Append("tbagi017.TXT\_VLR\_PADR TXT\_VLR\_PADR, \n");  complexSQL.Append("tbagi017.TXT\_FORT\_CAMO TXT\_FORT\_CAMO, \n");  complexSQL.Append("tbagi017.NUM\_ORDE\_CAMO NUM\_ORDE\_CAMO, \n");  complexSQL.Append("tbagi017.COD\_IDEF\_CAMO COD\_IDEF\_CAMO, \n");  complexSQL.Append("tbagi017.IND\_CAMO\_COPL IND\_CAMO\_COPL, \n");  complexSQL.Append("tbagi017.IND\_DOMN\_CORO IND\_DOMN\_CORO, \n");  complexSQL.Append("tbagi017.TXT\_NOM\_DOMN TXT\_NOM\_DOMN, \n");  complexSQL.Append("tbagi017.NOM\_IPI\_LAYO\_CAMO\_TRAB NOM\_IPI\_LAYO\_CAMO\_TRAB \n");  switch (objectKey.consulta)  {  case tbagi017IpiCamoKey.enTipoConsulta.porCarteira:  complexSQL.Append("FROM dbo.TBAGI017\_IPI\_CAMO tbagi017, \n");  complexSQL.Append("dbo.TBAGI005\_IPI\_CAMO\_SIST\_PROD tbagi005 \n");  complexSQL.Append("WHERE tbagi017.IDT\_IPI\_CAMO=tbagi005.IDT\_IPI\_CAMO \n");  complexSQL.AppendFormat("AND tbagi005.IDT\_CRTR={0}", objectKey.IdentificadorCarteira);  break;  case tbagi017IpiCamoKey.enTipoConsulta.Um:  break;  case tbagi017IpiCamoKey.enTipoConsulta.In:  complexSQL.Append("FROM dbo.TBAGI017\_IPI\_CAMO tbagi017 \n");  var inIPI = "";  objectKey.idtIpiCamo.ForEach(id =>  {  inIPI = String.Concat(inIPI, id.ToString(), ",");  });  inIPI = inIPI.Substring(0, inIPI.Length - 1);  complexSQL.AppendFormat("WHERE tbagi017.IDT\_IPI\_CAMO IN ({0})", inIPI);  break;  default:  break;  }  return complexSQL.ToString();  }  } |

## IDataKey

Interface comum utilizada para implementação de critérios de busca de uma **CollectionDataManager** ou **CollectionQueryableManager.**

## TransactionManager

Utilizado para gerenciar conexões e Transações executadas no banco de Dados.

|  |
| --- |
| using (transactionDelete = new TransactionManager("RemoverCampoIPITrans", "EntityModel"))  {  listData005.ForEach(data005 => { data005.Delete(transactionDelete); });  listData007.ForEach(data007 => { data007.Delete(transactionDelete); });  listData017.ForEach(data017 => { data017.Delete(transactionDelete); });  listData027.ForEach(data027 => { data027.Delete(transactionDelete); });  listData028.ForEach(data028 => { data028.Delete(transactionDelete); });  listData030.ForEach(data030 => { data030.Delete(transactionDelete); });  transactionDelete.Commit();  } |

# Componente responsável pelo Monitoramento de Erros e Trace da Aplicação

## Componente de Monitoramento

Componente interno no qual é responsável por registro de Assíncrono de Mensagens enviadas através do **Serviço de Monitoramento** a serem registradas no LOG de Eventos do Windows ou em uma base de dados destinada para LOG.

## Serviço de Monitoramento

Responsável por utilizar o componente de monitoramento e receber mensagens do cliente para registro em LOG de Eventos do Windows ou Banco de Dados.

## Cliente de Monitoramento

Responsável pelo envio de mensagens :

|  |
| --- |
| var monitor = MonitorClientFactory.Create();  monitor.TraceError(new monitor.service.MonitorData { OperationName = \_Operation, TraceMessage = String.Format("Error in operation {0} \n on point {1} \n with message {2} \n and default {3}", \_Operation, \_CheckPoint, \_CustomMessage, base.Message) + " >> " + \_stackTrace }); |

# Componente comum de interface entre serviço e Fluxo de Negocio

## DataTransferObject

Responsável pelo recebimento de dados providos pelas **DataEntity** e exposição através de um contrato de serviço.

Todo objeto DTO ( Data Transfer Object ), terá suporte automático de transferência de preenchimento e recebimento de informação providos pela **EntityData** através dos métodos **Fill** e **Bind**.

Podemos ter também suporte a serialização em **XML®** ou **JSON®**.

A seguir temos um exemplo de utilização:

|  |
| --- |
| protected override List<ParserData> ParseDataList(IDataEntity data)  {  return new List<ParserData>()  {  new ParserData { DataPropertyIdentifier = "IPICampo", DTOPropertyIdentifier = "IPICampo" },  new ParserData { DataPropertyIdentifier = "TamanhoCampo", DTOPropertyIdentifier = "TamanhoCampo" },  new ParserData { DataPropertyIdentifier = "CodTipoCampo", DTOPropertyIdentifier = "CodTipoCampo" },  new ParserData { DataPropertyIdentifier = "NomeIPILayoutCampo", DTOPropertyIdentifier = "NomeIPILayoutCampo" },  new ParserData { DataPropertyIdentifier = "TextoValorPadrao", DTOPropertyIdentifier = "TextoValorPadrao" },  new ParserData { DataPropertyIdentifier = "TextoFormatoCampo", DTOPropertyIdentifier = "TextoFormatoCampo" },  new ParserData { DataPropertyIdentifier = "NumeroOrdemCampo", DTOPropertyIdentifier = "NumeroOrdemCampo" },  new ParserData { DataPropertyIdentifier = "CodigoIdentificacaoCampo", DTOPropertyIdentifier = "CodigoIdentificacaoCampo" },  new ParserData { DataPropertyIdentifier = "IndicadorCampoComplementar", DTOPropertyIdentifier = "IndicadorCampoComplementar" },  new ParserData { DataPropertyIdentifier = "IndicadorDominioCorporativo", DTOPropertyIdentifier = "IndicadorDominioCorporativo" },  new ParserData { DataPropertyIdentifier = "TextoNomeDominio", DTOPropertyIdentifier = "TextoNomeDominio" },  new ParserData { DataPropertyIdentifier = "NomeIPILayoutCampoTrabalho", DTOPropertyIdentifier = "NomeIPILayoutCampoTrabalho" }  };  } |

## IDataEntity

Utilizada em conjunto com uma **DataTranferObject** especializada para viabilizar o **Fill** e **Bind** automáticos.

## BusinessFlow

Utilizada para determinar captura de validações comuns ao fluxo de negocio estabelecido.