# Core构架的优点

1. 直接支持分布式部署；
2. 直接支持多数据库部署；
3. 直接支持自定义的扩展数据类型，扩展对象实例直接可以存库；
4. 极大程度地减少编码数量。

# 基本使用方法

## 定义实体类

[Serializable] //必须加可序列化特性; 否则无法进行分布式部署和使用MemCache缓存

[Table("UserInfo")] //可选特性; 设置数据库表的名称; 默认为:类名+"Info";

public class User : Core.Data.DataModel //必须继承自该类

{

/// <summary>

/// 用户名

/// </summary>

[Column("Name")] //可选特性; 设置数据库表中的字段名称; 默认与属性名相同

public string Name { get; set; }

/// <summary>

/// 年龄

/// </summary>

[Column("Age")] //可选特性; 设置数据库表中的字段名称; 默认与字段名相同

public int Age = 0;

}

**注意事项:**

1. 实体类必须继承自Core.Data.DataModel类
2. 实体类必须添加[Serializable]特性

## 定义数据入口类

/// <summary>

/// User的数据入口层

/// </summary>

public class UserDataPort : Core.Data.DataPortal<UserDataPort, User>

{

}

**注意事项:**

1. 数据入口类必须继承自Core.Data.DataPortal类
2. Core.Data.DataPortal<UserDataPort, User> 中User表示对应的实体类

## 开始使用

User userA = new User() { Name = "张三", Age = 16 };

User userB = new User() { Name = "李四", Age = 16 };

//增 方法1

UserPortal.InsertToDb(userA);

//增 方法2

UserPortal udp = new UserPortal();

udp.Insert(userB);

//改 方法1

userA.Age = 18;

UserPortal.UpdateToDb(userA);

//改 方法2

userB.Age = 18;

udp.Update(userB);

//根据主键Uid查询;

User a = UserPortal.GetModel(userA.Uid);

//根据列查询;返回单一实体

User b = UserPortal.GetModelByField("Name", userB.Name);

//根据列查询

List<User> users = UserPortal.GetModels("Age", 18);

//获取全部

users = UserPortal.GetAllModels();

//自定义SQL查询

users = udp.Query("SELECT \* FROM UserInfo WHERE Age>16 ");

//删 方法1

UserPortal.DeleteFromDb(userA);

//删 方法2

udp.Delete(userB);

# 高级使用方法

## 使用自定义数据类型

1. 自定义数据的类型，需要实现Core.Data.IExtendDbType接口；该接口定义了用户需要实现的序列化与反序列化方法，以及对应的数据库存储格式。
2. Core构架内提供了Core.Data.ExtendDbType抽象类，已经实现了Core.Data.IExtendDbType接口；该类使用xml格式对对象进行序列化；通常自定义的数据类型只需要继承自Core.Data.ExtendDbType类即可，如果需要使用二进制或者Json类型的序列化，可以自行实现Core.Data.IExtendDbType接口。

示例:

//自定义数据类型

[Serializable] //必须可序列化

public class UserExtend : Core.Data.ExtendDbType

{

public string CountryName { get; set; }

public int PropertyA { get; set; }

public int PropertyB { get; set; }

}

[Serializable]

public class User : Core.Data.DataModel

{

//使用自定义数据类型

public UserExtend ExtendObject { get; set; }

public string Name { get; set; }

public int Age = 0;

}

## 忽略某个属性/字段

因为并不是类内的公共属性/字段，在数据库内都有对应的字段的,所在这里添加了忽略某个字段或者属性的标识[Ignore]

示例:

[Serializable]

public class User : Core.Data.DataModel

{

public string Name { get; set; }

public int Age = 0;

[Ignore] //表示忽略该字段;该字段在数据库中没有对应的列

public int Total { get; set; }

}

## 多数据库部署

在单台数据库服务器，无法承受数据访问的需求的时候，可以将部分表，分离到不同的数据库中，进行多数据库部署。这里提供了一个特性[ConnectionSetting]，用来标识该类使用那一个数据库连接串；仅需要简单在在类上添加[ConnectionSetting]特性，即可进行分库。

示例:

[Serializable]

[ConnectionSetting("DbConnectionString")] //定义该类使用的数据库连接串名称

public class User : Core.Data.DataModel

{

public string Name { get; set; }

public int Age = 0;

}

## 分布式部署

有时候，仅需要将部分数据入口进行分布式部署，部分留在本地执行；这时候，仅需要在数据入口类中，添加[DataPortalSetting]特性，设置数据入口所需用的配置即可。默认的采用本地数据入口。

示例:

/// <summary>

/// User的数据入口层

/// </summary>

[DataPortalSetting("DataPortalClient", "RemotingPortalUrl")] //设置数据入口所使用的配置

public class UserPortal : Core.Data.DataPortal<UserPortal, User>

{

}

其中:

第一个参数是数据入口使用的入口类型设置；"DataPortalClient"表示使用的配置文件中appSettings / DataPortalClient的配置，生成相应的数据入口实例;

第二个参数是数据入口使用的远程Url设置；"RemotingPortalUrl"表示使用的配置文件中appSettings / RemotingPortalUrl的配置, 作为远程对象的地址；如果DataPortalClient为本地配置，则Url值为空

配置文件中的设置:

<add key="DataPortalClient" value="Core.Client.RemotingPortalClient, Core"/>

<add key="RemotingPortalUrl" value="http://localhost:9999/RemotingPortal.rem"/>

# 构架原理

## Core的层级关系图

1. DataPortal层，提供一个抽象的基类，实现了对下一层不同数据入口客户端的调度；提供很多抽象的方法，以实现在增删改查后所需要触发的操作；
2. DataPortal Client层，实现了基于Remoting、WebService、Local(本地调用)的数据调用的客户端，其中Remoting/WebService客户端，还实现了本地基于线程的缓存，防止频繁多次调用同一远程入口，提高本地性能；
3. DataPortal Server层，实现了基于Remoting、WebService服务端数据入口的服务端；
4. DataAccess层，实现上一层数据入口服务端所需要的数据操作类型，基本的增删改查，
5. DbMapper模块，主要实现基本的ORM操作，目前支持SQL Server/MySql/Oracle数据库；
6. Cache模块，主要实现了缓存的进一步封装，目前支持基于.NET Cache和MemCache两种

## 构架使用的层级关系图

1. User-defined DataPortal层，为用户定义实现的层；该层所有类，继承自DataPortal层提供的抽象类Core.Data.DataPortal;
2. Business Layer是用户实现的自身的业务逻辑

## 构架全部模块层级关系

1. Model对应基本使用方法用的User类，继承自Core.Data.DataModel；
2. User-defined DataPortal对应基本使用方法用的UserPortal类， 继承自Core.Data.DataPortal；其中可以在这里重写基类的方法，以实现在增删改查后所需要触发的操作
3. Business Layer即为我们具体业务所需要的增删改查等一些操作

## 简化的构架图

## 单机部署结构

## Remoting分布式部署