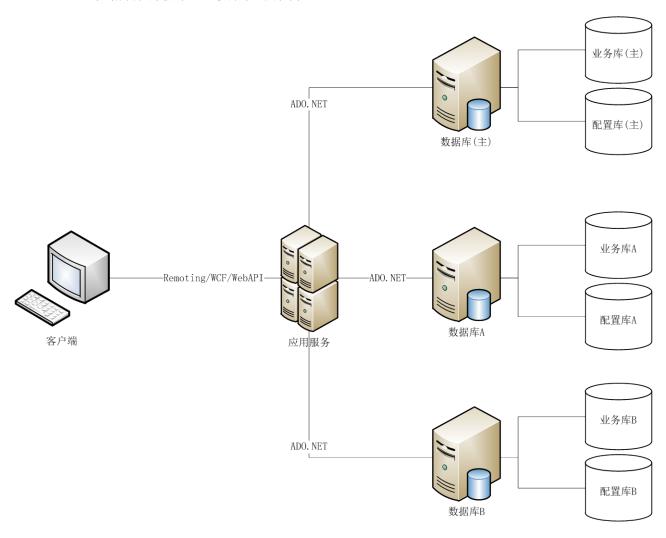
## 22 数据库集群

Phenix 、数据库集群技术,可实现如下架构:



只要在项目实施时,实施人员通过应用服务的数据库连接串配置界面,设定好 N 个数据库的连接串及对应的数据源键(DataSourceKey)值,就可以实现一个应用服务同时操作 N 个数据库的设计目的,为客户端提供透明的数据持久化服务。

如果 DataSourceKey 为空,代表本连接串指向的数据库为主数据库。每个应用系统只有唯一一个主数据库,主数据库将存放应用系统完整的配置数据、完整的数据字典(包括所有分数据库的数据字典),同时也是客户端用户登录验证和操作数据的默认数据库。除此之外的数据库连接串(DataSourceKey 为非空),都指向的是分数据库。分数据库仅用于存储业务数据,其数据字典也要在主数据库中有构建,而且互相要保证一致。

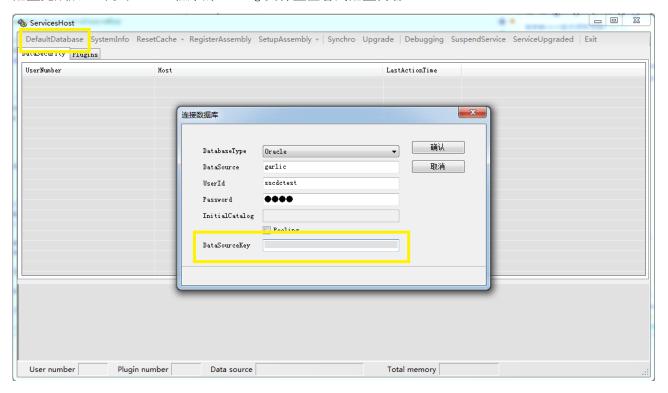
## 22.1 配置方法

应用服务 Host 程序上,提供了数据库连接串配置界面,方法如下。

首先,应该配置的是主数据库连接串,不用填写 DataSourceKey (输入框保持为空)。然后,就可以

随意打开配置界面,配置上分数据库的连接串(比如,将"other"值填写到 DataSourceKey 输入框)。

配置完成后,可以在 Host 程序的 config 文件里查看到配置内容:



```
- - X
Phenix.Services.Host.x64.exe.config - 记事本
 文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
K?xml version="1.0" encoding="utf-8"?>
 <configuration>
         <appSettings>
                 Settings/
<add key=".DatabaseType" value="D20AEBA5EA0B450F" />
<add key=".DataSource" value="D6D12DE2482CC271" />
<add key=".UserId" value="488F1CF48E765727ACC6F3494A1A2141" />
<add key=".InitialCatalog" value="8ACB9A40AB57555A" />
<add key=".Pooling" value="F6256BCFE2E22996" />
<add key=".CommectionString"
<a href="".CommectionString"
<a href="".CommectionStrin
value="A326DEBF29DA8F1D17B9FF7B8E9E555870A80D36384694144DDD0F03499ED27B8A61F0E5B55CDBAE66287B751BDD110DE80F865DD3
89D8B1D1ACD01EC92B316DFC6807D3FD931BBF" />
                  add key="Phenix.Core.Dictionary.DataDictionaryHub.DepartmentInfoChangedLocalTime.garlic/sscdctest" (
                          value="2015-10-22 21:22:33" />
                  add key="Phenix.Core.Dictionary.DataDictionaryHub.PositionInfoChangedLocalTime.garlic/sscdctest" (حام
                          value="2015-10-22 21:22:33"
                  <add key="Phenix.Core.Dictionary.DataDictionaryHub.TableFilterInfoChangedLocalTime.garlic/sscdctest"</p>
                          value="2015-10-22 21:22:33"
                  "Add key="Phenix.Core.Dictionary.DataDictionaryHub.RoleInfoChangedLocalTime.garlic/sscdctest"
                          value="2015-10-22 21:22:33" />
                  <add key="Phenix.Core.Dictionary.DataDictionaryHub.SectionInfoChangedLocalTime.garlic/sscdctest"
value="2015-10-22 21:22:33" />
                  <add key="Phenix.Core.Dictionary.DataDictionaryHub.AssemblyInfoChangedLocalTime.garlic/sscdctest"</p>
                  value="2015-10-22 21:22:33" />
<add key="CslaAuthentication" value="Phenix" />
                  <add key="Phenix.Core.Net.RemotingConfig.TcpPort." value="8086" />
                  <add key="Phenix.Core.Net.RemotingConfig.EnsureSecurity."</p>
                                                                                                                                             ′value=″False″/>
                  <add key="Phenix.Core.Net.RemotingConfig.CompressionSupported.</pre>
                          value="False"
                  value= raise //
<add key="Phenix.Core.Net.RemotingConfig.CompressionThresholdMax."
value="8388608" />
                  <add key="Phenix.Core.Dictionary.DataDictionaryHub.TableInfoChangedLocalTime.garlic/sscdctest"
value="2015-10-22 21:22:33" />
                 value="A326DEBF29DA8F1D608629E68853629C64DA8647A09DA7352A80757785190D9CEEA3E915B7E8E8E514FB334652D7E32E83CFCBE1E0
9495D372873A176ACE103B"/>
          √app5ettings/
 </configuration>
```

现在有了一个分数据库的 DataSourceKey 值 "other"。请注意大小写,Phenix 《是区分大小写的, 会代表不一样的连接串。

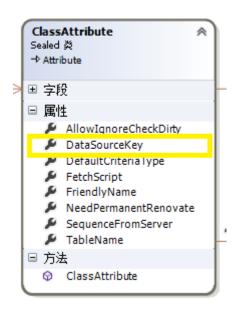
如果 Phenix 、找不到 DataSourceKey 对应的数据库连接串配置项,会采用默认的主数据库连接串。

## 22.2 开发接口

上述设置的数据源键(DataSourceKey)值,应该与应用系统的代码保持一致,DataSourceKey 就是 开发和实施之间的一个契约。这样,开发阶段没必要将数据库连接串写死在代码里,而实施阶段也不必 做二次开发。

应用系统操作数据库,无非是 Fetch 和 Save 两个动作。所以,DataSourceKey 如何在代码里设定,主要与这两个动作相关。

默认的 DataSourceKey,可标记在所需 Fetch 根 (Root) 对象的 ClassAttribute 标签上:



这是静态的设定方法。如需动态设定,可以覆写 BusinessBase、BusinessListBase、CommandBase 的 DataSourceKey 属性。见以下示例代码(请注意,注释里说明了如果不被覆写的话,该属性如何从上到下遍历采纳为返回值的优先级):

```
/// 

/// 数据源键

/// 缺省为 Root. DataSourceKey

/// 缺省为 Criterions. DataSourceKey

/// 缺省为 T、TBusiness 上的 ClassAttribute. DataSourceKey

/// </summary>

[System. ComponentModel. Browsable (false)]

[System. ComponentModel. DataAnnotations. Display (AutoGenerateField = false)]

public override string DataSourceKey

{
   get { return "other"; }
}
```

上述示例很简单。实际场景下,可通过编写代码(实现业务逻辑)返回不同的 DataSourceKey 值。 另外一种方法,还可以在 Fetch 用的(继承自 CriteriaBase 的)查询类里覆写 DataSourceKey 属性, 写法同上:



如果是用 Linq 表达式 Fetch 的话,BusinessBase、BusinessListBase 的 Feth()函数提供了 DataSourceKey 参数:

```
/// <summary>
/// 构建业务对象集合
/// </summary>
/// <param name="dataSourceKey">数据源键</param>
/// <param name="criteriaExpression">条件表达式</param>
/// <param name="orderByInfos">数据排列顺序队列</param>
public static T Fetch(string dataSourceKey, Expression<Func<TBusiness, bool>> criteriaExpression,
params OrderByInfo[] orderByInfos)
```

这些动态设定的 DataSourceKey,优先级比静态的(ClassAttribute. DataSourceKey)高,查询条件的 DataSourceKey 优先级最高。Phenix × 在 Fetch 时,如果发现它返回 null(如果查询条件的目的是连接主数据库的话,返回值应该是 String. Empty),会退而求其次,采纳 Root 对象的 DataSourceKey 属性值。

主从结构里 GetDetail 从对象的时候,必定会采用 Root 对象的 DataSourceKey 属性值。同样,Save 时也会采用 Root 对象的 DataSourceKey 属性值。

Root 对象的 DataSourceKey 属性值,默认返回的是查询条件上的 DataSourceKey。所以,如果希望动态来设定 DataSourceKey 的话,一般情况下还是设定在查询类上较为简便。

## 22.3 建议

作为软件产品,当开发阶段结束后,请将代码中的 DataSourceKey 做一次汇总说明,写到实施手册里,以便实施人员可独立完成自己的工作。