19 缓存

一提到缓存，往往联想到的是 B/S 架构上的缓存技术，本文所述的 Phenix丶缓存技术，在基本功能上是和这些普通的缓存技术差不多的，只是实现的方式不一样：

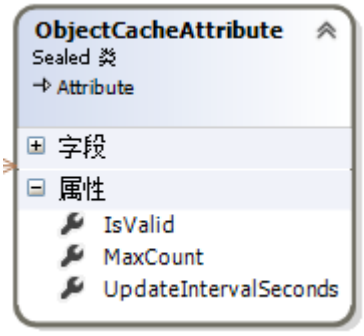
- 性能：将相应数据存储起来以避免数据的重复创建、处理和传输，可有效提高性能。比如将不改变的数据缓存起来，例如国家列表等，这样能明显提高应用系统的反应速度；
- 稳定性：同一个应用中，对同一数据、逻辑功能和用户界面的多次请求是经常发生的，当用户基数很大时，如果每次请求都进行处理，消耗的资源是很大的浪费，也同时造成系统的不稳定。而缓存数据也能降低对数据库的访问次数，降低数据库的负担和提高数据库的服务能力；
- 可用性：有时，提供数据信息的服务可能会意外停止，如果使用了缓存技术，可以在一定时间内仍正常提供对最终用户的支持，提高了系统的可用性。

Phenix丶缓存技术是专用于业务对象的。相对于普通的缓存技术，它更进了一步，只要系统应用了它，每次通过它检索到的业务数据，都是已经和数据库实现了同步的，调用方无需关心业务数据是从本地拿的、还是从应用服务程序上拿的、或者是从数据库上拿的，反正 Phenix丶缓存技术返回给调用方的业务数据始终是最新的，和数据库是保持一致的。

缺省情况下，只要是全景数据的业务类都会被自动缓存到本地，但也可以干预缓存。

19.1 干预缓存

19.1.1为业务类打上 Phenix.Core.Cache.ObjectCacheAttribute 标签



属性	说明	备注
UpdateIntervalSeconds	更新间隔频率(秒数)	缺省为 60 秒；对于基础数据的缓存，希望在整个进程生命周期内不再维持本地业务数据的最新版本，可设置为 int.MaxValue；
MaxCount	缓存对象数量	缺省为 10；按照查询条件缓存业务（集合）对象，如果缓存

		数超出 MaxCount 则会清空缓存；
IsValid	是否有效	缺省为 true；可强制禁用缓存技术；

19.1.2申明是否需要使用缓存

19.1.2.1 查询业务对象时申明

见“11.业务对象生命周期及其状态”中“从数据库获取业务对象”章节，在 Fetch 时提供了 cacheEnabled 参数。但是，是否真正被用到缓存，还受到ObjectCacheAttribute标签IsValid 属性值的限制（见前文）。

19.1.2.2 通过 Phenix.Windows.BarManager 组件申明

见“10.BarManager 组件”中“BindingSources”章节，其实质是设置当 Fetch 业务（集合）对象时如何提供 cacheEnabled 参数。

19.2 非常规开发模式下如何通知 Phenix丶已更新数据

Phenix丶缓存技术是透明的，只要你不绕道走，基本无需关心业务数据的版本问题。也就是说，只要业务数据是映射到业务类且通过业务对象 Save()函数来提交的，就无需操心被缓存的业务数据是否已更新，所检索到的业务数据是否已经和数据库同步，因为这些问题 Phenix丶都帮你自动处理了。

19.2.1当持久层中执行了自己编写的表记录增删改 SQL 语句

Phenix 丶是允许直接通过 Phenix.Core.Data.DbConnectionHelper 、 DbCommandHelper 、DataSetHelper 等 ADO 帮助类直接操作数据库的（不允许直接操作 ADO.NET 库），这种情况下 Phenix丶是无法感知到这些业务数据的变更，那么，请你在提交数据的代码之后添加上类似以下的代码，来通知到 Phenix丶：

```
Phenix.Core.Cache.ObjectCache.RecordHasChanged("PH_PROCESSLOCK");
```

这用到了 Phenix.Core.Cache.ObjectCache 提供的函数：

```
/// <summary>
/// 声明某表记录发生更改
/// </summary>
/// <param name="tableName">表名</param>
public static void RecordHasChanged(string tableName)
```

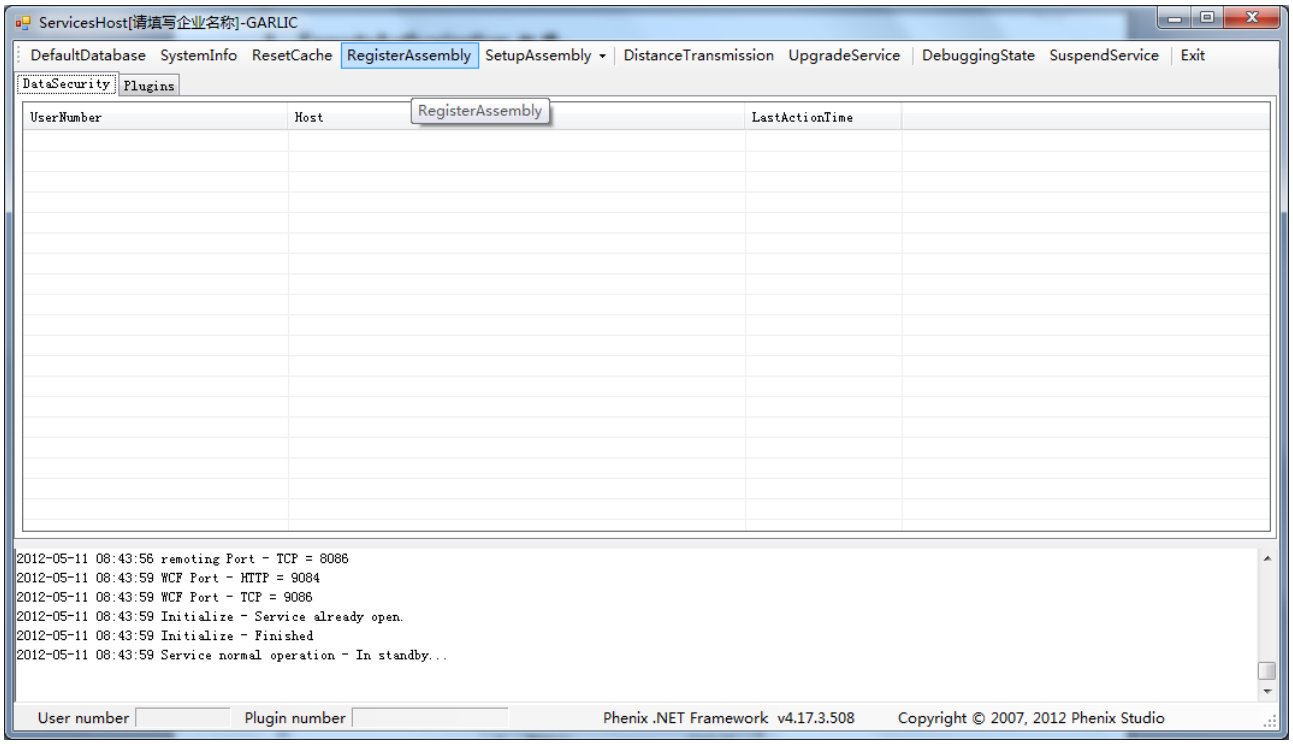
19.2.2当存储过程、触发器中执行了表记录增删改、或通过数据库工具手工修改了表记录

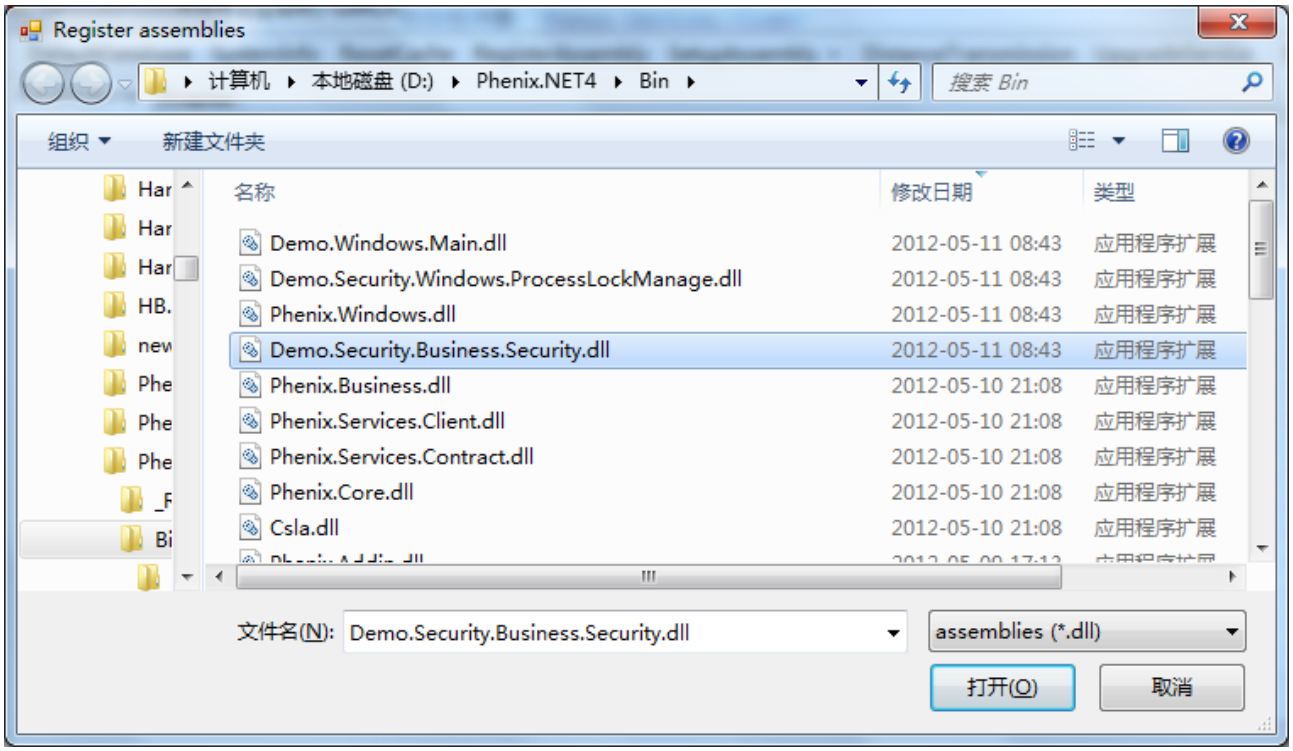
当直接操作数据库完成数据更新后，应该在存储过程、触发器或者数据库工具中，直接运行一次下例中的存储过程（传入参数为被修改数据表的表名），来通知 Phenix丫：

```
PH_Record_Has_Changed("PH_PROCESSLOCK");
```

19.3 注册业务组件

虽然做了前述中的一些操作，但只有事先通过Phenix.Services.Host.exe注册了业务组件，才能正常使用到缓存技术，否则无法自动维持本地业务数据的最新版本：





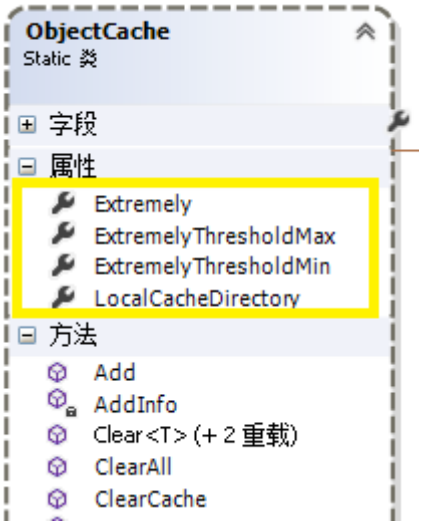
找到业务组件的程序集，打开后注册的提示信息类似于：

```
2012-05-11 08:47:47 Initialize failure - If without registration procedures set components can be omitted the tip.：Demo.Security.Business.Security, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null未找到 IPlugin 接口
2012-05-11 08:47:49 Initialize - Register class: Demo.Security.Business.Security.ProcessLock
2012-05-11 08:47:49 Initialize - Register class: Demo.Security.Business.Security.ProcessLockList
2012-05-11 08:47:49 Initialize - Registered assembly over.
```

如果系统开发中有升级了业务组件的程序集，需再次注册该程序集的最新版本。

19.4 全局配置参数

以下是 ObjectCache 类的静态全局变量，可调整系统默认的行为：



属性	说明	备注
Extremely	极端方式：将缓存保存到本地	缺省为 !AppConfig.AutoMode；注意：一般默认情况下是客户端上会将业务类数据自动写到本地硬盘里，待下次登录时，如果数据库没有发生变化，就直接读取本地数据，这对于性能好的客户端机器是能提速的，但对 IO 性能实在太差的机器，建议关掉本功能；
ExtremelyThresholdMin	极端方式最小阈值：缓存流长度	缺省为 1024 * 1024 / 2
ExtremelyThresholdMax	极端方式最大阈值：缓存流长度	缺省为 1024 * 1024 * 4
LocalCacheDirectory	本地缓存目录	