### Fwk.Caching

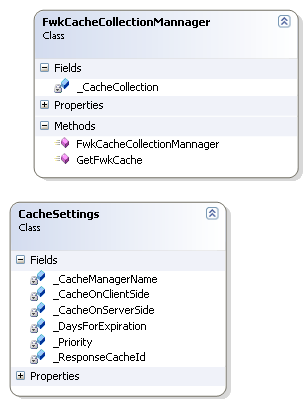
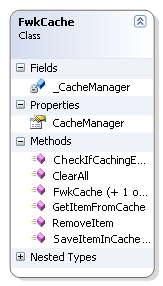
Bloque de caching

El sistema de caching que provee el Framework permite a las aplicaciones (Back-End o Front-End) disponer un conjunto de mecanismos que facilitan el almacenamiento en IsolatedStarages customizables de información persistente que sea requerida en las aplicaciones.

El sistema de caching se basa en lasEnterprise Library Caching Application.

# Modelo de componentes

La ubicación de los componentes de caching es en las Fwk.Caching y tiene el siguiente aspecto:



**Figura 1.0**

El modelo de caching es automáticamente utilizado por los componentes bases de Front-End y Bac-End de modo que es muy poco lo que hay que implementar por parte de los desarrolladores al momento de crear servicios.

### Componentes

**FwkCache**

Proporciona una abstracción al manejo de la cache. Esta clase contiene los métodos para manipular la cache factory y esta adecuada para la funcionalidad de la arquitectura.

Reúne en un punto común la tecnología de caching, de esta manera si se desea dejar de actualizar los App Block de P&P simplemente se cambia la implementación interna de esta clase.

**Ejemplo**

FwkCache wFwkCache = new FwkCache("Ventas");

ClienteBE wCli = new ClienteBE();

wCli.IdCliente = 50999;

wCli.Apellido = "Aguirre";

wCli.Edad = 69;

wFwkCache.SaveItemInCache(wCli.IdCliente.ToString(), wCli);

**Código 1.0**

Métodos

|  |  |
| --- | --- |
| **Nombre de método** | **Descripción** |
| CheckIfCachingExists | Determino si el Item Existe en Caché |
| SaveItemInCache(String pCahcheId, Object pObject) | Guarda Ítem en Caché según una key especificada.  ***pCahcheId***: Claye del Ítem a Guardar  ***pObject***: Ítem a Guardar |
|  |  |
| SaveItemInCache(String pCahcheId, Object pObject,Boolean pReplaceIfExist) | Guarda Ítem en Caché según una key especificada  **pReplaceIfExist**  Si es = True y si existe algun item con el mismo Id lo reemplaza |
| SaveItemInCache(Object pObject, CacheItemPriority pPriority, Double pDaysFromExpiration) | Guarda Ítem en Caché según una key especificada.  Permite establecer la prioridad y los días deseados para la expiración del ítem en la cache  Este método genera el GUID y lo retorna como un string |
| SaveItemInCache(String pCahcheId,Object pObject, CacheItemPriority pPriority, Double pDaysFromExpiration) | Similar al anterior pero recibe como parámetro el Guid. |
| SaveItemInCache(Object pObject,CacheItemPriority pPriority,DateTime pDateExpiration) | Guarda Ítem en Caché según una key especificada.  Permite establecer la prioridad y la fecha de expiración deseada del ítem en la cache  Este método genera el GUID y lo retorna como un string. |
| SaveItemInCache(String pCahcheId, Object pObject, CacheItemPriority pPriority, DateTime pDateExpiration) | Similar al anterior pero recibe como parámetro el Guid |
| ClearAll() | Borra todos los ítems de Caché |
| RemoveItem(String pCahcheId) | Eliminar un Ítem de Caché  pCahcheId = Clave con el que se guardó el objeto |
| GetItemFromCache(String pCahcheId) | Recupera un Ítem de Cache dependiendo del Identificador del mismo.  pCahcheId = Clave con el que se guardó el objeto |

**Tabla 1**

**FwkCacheCollectionMannager**

Debido que pueden convivir diferentes contextos de cacheo en una aplicación es conveniente tener más de una clase FwkCache

Por tal motivo para alejar al desarrollador de la instanciación continua y verificaciones de configuración de cada una de las cache. se utiliza esta clase que es la encargada de mantener una colección de FwkCache y los mantiene en memoria mediante el patrón singlenton (instanciación única vez)

**CacheSettings**

(*Se explica con mas detalle en Implementación de Caching en la Arquitectura)*

**Implementación de Caching en la Arquitectura**

Viéndolo desde esta perspectiva integrada en la arquitectura tenemos dos puntos principales para aplicar caching.

Por un lado todos los servicios disponen de una sección llamada **CacheSettings** donde se establecen todos los atributos necesarios para almacenar o leer de la Cache.

**Atributos**

|  |  |
| --- | --- |
| **Nombre de atributo** | **Descripción** |
| CacheOnServerSide | Bandera que indica si los resultados de la ejecución del servicio serán primero intentados obtener desde la cache del lado del servidor |
| CacheOnClientSide | Bandera que indica si los resultados de la ejecucion del servicio seran primero intentados  otener desde la cache del lado del cliente |
| ResponseCacheId | Identificador de la cache para el caso de que el servicio este cacheado  tanto en el lado del cliente como en el servidor.  Puede proporcionarle cualquier identificador de cache. Por ejemplo:  1- El mismo nombre del servicio  2 - Nombre de servicio mas fecha  2 - Nombre de servicio mas Dominio/ Area donde corra el cliente o servidor  Si este valor es = Empty() y alguna de las CacheOnClientSide o CacheOnServerSide están establecidas en true se asume el Id de la cache del servicio con el nombre del servicio.-  EJ:  BuscarPaisesClienteRequest req = new BuscarPaisesRequest();  req.ResponseCacheId = req.ServiceName + "RRHH";  req.CacheOnServerSide = true; |
| CacheItemPriority | Enumeración que especifica la prioridad relativa de los elementos almacenados en el objeto FwkCache.  Valores   |  |  | | --- | --- | | **AboveNormal** | Los elementos de la memoria caché con este nivel de prioridad tienen menos posibilidades de ser eliminados cuando el servidor libera la memoria del sistema que aquéllos que tengan asignada una prioridad Normal. | | **BelowNormal** | Los elementos de la memoria caché con este nivel de prioridad tienen más posibilidad de ser eliminados cuando el servidor libera la memoria del sistema que aquéllos que tengan asignada una prioridad Normal. | | **Default** | El valor predeterminado para la prioridad de un elemento de la memoria caché es Normal. | | **High** | High Los elementos de la memoria caché con este nivel de prioridad son los que menos posibilidades tienen de ser eliminados de la memoria caché cuando el servidor libera la memoria del sistema. | | **Low** | Los elementos de la memoria caché con este nivel de prioridad son los que más posibilidades tienen de ser eliminados de la memoria caché cuando el servidor libera la memoria del sistema. | | **Normal** | Los elementos de la memoria caché con este nivel de prioridad podrán ser eliminados de la memoria caché cuando el servidor libere la memoria del sistema sólo después de eliminarse los elementos con la prioridad Low o BelowNormal. Éste es el valor predeterminado. | | **NotRemovable** | Los elementos de la memoria caché con este nivel de prioridad no se eliminarán de la memoria caché cuando el servidor libere la memoria del sistema. Sin embargo, los elementos con este nivel de prioridad se quitan junto con otros elementos en función de la fecha de caducidad absoluta o variable del elemento. | |
| DaysForExpiration | Determina si existe una marca de bloqueo por su GUID o BlockingId  Retorna un true o false |
| CacheManagerName | Es similar a la GetByParam pero no retorna las marcas sino el/los usuarios que tienen tomada esa marca de bloqueo.- |

Tabla 2

Los componentes de un servicio que disponen de objetos relacionados a caching integrado son los Request.

**Porque esto?**

Esto es porque un request es el punto de entrada para la solicitud de un servicio por lo tanto es este quien le informa al servicio si los datos se obtienen de un repositorio de cache o de un servidor de aplicaciones que ejecuta comandos de base de datos u otro origen diferente al cache.

También decide si la cache se va a aplicar del lado del servidor de aplicaciones o del lado del cliente.

**Ejemplo:**

Supongamos un servicio de búsqueda de localidades que es de muy poca actualización y se decide dejar almacenada en memoria o disco sin necesidad de ir por cada peticion al servidor de aplicaciones.

El código se vería como sigue:

BuscarProvinciasTodasRequest req = new BuscarProvinciasTodasRequest();

req.CacheSettings.CacheManagerName = "localidades";

req.CacheSettings.DaysForExpiration = 60;

req.CacheSettings.CacheOnClientSide = true;

req.CacheSettings.ResponseCacheId = "localidades cba";

**Código 2.0**

Aquí se decidió para este caso cachear las localidades de córdoba en el lado de cliente con un tiempo de expiración de 60 días.

Aclaraciones:

* La configuración de donde se almacena la cache se describe de manera configurativa y es explicado en la sección “***configuración de la cache****”*
* ResponseCacheId determina el nombre que identifica el ítem almacenado en la cache.

Si no se especifica nada se tomara el nombre del servicio.

En este caso es recomendable establecer un nombre para identificar las busquedas de

Localidades de diferentes provincias.

**Configuración de la cache para servicios:**

1. Para configurar el sistema de cacheo es necesario editar el archivo .config de la aplicación.
2. Agregar la sección de configuración de caching, en **configSections** del archivo .config la siguiente sección

<section name="cachingConfiguration" type="Microsoft.Practices.EnterpriseLibrary.Caching.Configuration.CacheManagerSettings, Microsoft.Practices.EnterpriseLibrary.Caching, Version=4.0.0.0, Culture=neutral" />

1. Agregar la configuración en el .config los diferentes contextos de caching bajo los que se quiera trabajar.

El siguiente ejemplo muestra la configuración de un contexto de ventas y otro de localidades donde se cachean en niveles de aislamiento diferentes y con diferentes criterios.

<cachingConfiguration defaultCacheManager="ConfigurationIsolatedManager">

<cacheManagers>

<add expirationPollFrequencyInSeconds="60"

maximumElementsInCacheBeforeScavenging="1000"

numberToRemoveWhenScavenging="60"

backingStoreName="NullStorage"

name="localidades" />

<add expirationPollFrequencyInSeconds="60"

maximumElementsInCacheBeforeScavenging="1000"

numberToRemoveWhenScavenging="10"

backingStoreName="IsolatedStorage"

name="ventas" />

</cacheManagers>

<cachingConfiguration

1. El nodo raíz indica la sección de Caching por defecto en caso que se use caching sin establecer el nombre.

<cachingConfiguration defaultCacheManager="ConfigurationIsolatedManager">

1. Lo que continua es una colección de cacheManagers que definen los diferentes contextos de cacheo.

Por ejemplo el primero, “localidades”:

* Cantidad de elementos en cache antes de que se produzca la recolección, 1000 elementos
* Cantidad de elementos a destruir cuando se produce la recolección 10.
* Almacena en memoria ***backingStoreName***=" NullStorage"
* Frecuencia de de barrido de elementos 60 segundos

**CacheManagers**

Atributos de configuración de **cacheManagers**

|  |  |
| --- | --- |
| **Nombre del atributo** | **Descripción** |
| expirationPollFrequencyInSeconds | Lo utiliza la clase ***ExpirationPollTimer***  Interna del modelo de caching y determina los la frecuencia con la que se lee los tiempos de expiración de la cache. La expiración se determina por medio de la clase FwkCache directamente o sección cacheSetting de un servicio. El atributo a modificar es DaysForExpirationexplicadoen la tabla 01 |
| maximumElementsInCacheBeforeScavenging | Máximo número de elementos que pueden estar en la cache antes de que la búsqueda de elementos para su expiración comience. Por defecto es 1000 elementos.  Es decir si ya se alcanzó el número máximo de 1000 elementos en cache y se insertan nuevos elementos primero se eliminaran  numberToRemoveWhenScavengingelementos por orden de prioridad y luego se insertaran los nuevos. |
| backingStoreName | Identificador dl almacén de la cache  IsolatedStorage = Disco  NullStorage = Memoria |
| numberToRemoveWhenScavenging | Número de elementos que se eliminaran de la cache después de que la búsqueda de elementos. Por defecto son 10. |
| name | Nombre de la administradora de cache |

**Tabla 3**

**Escenarios de uso**

Ejemplo de buenos escenarios de uso son comúnmente cuando las aplicaciones necesitan obtener listados de Categorías, Clasificaciones de tipos, Países, Localidades etc.

Lo que tienen en común estas entidades es que con muy baja frecuencia son modificadas de su origen de datos.

Con el fin de evitar los round-trips innecesarios para al obtener siempre la misma información se decide persistirlas en un medio de almacenamiento más veloz y que no tenga tanto costo de uso de servidores.

También es posible persistir otro tipo de información con más taza de modificaciones, pero que lo mismo sigue siendo necesario obtenerlos rápidamente. Ejemplo de estos pueden ser: permisos de usuarios, listado de proveedores, etc.

Si bien estos tipos de entidades pueden ser alterados día a día, las aplicaciones generalmente necesitan consultarlas varias veces en el transcurso del mismo y es muy poco probable que una modificación u agregado de nuevos registros para estas entidades altere la tarea diaria que tiene un usuario en su jornada.

Para estos diferentes casos de persistir información es que existen diferentes configuraciones de cacheo y también con diferentes tiempos de expiración.

También es posible desde cualquier punto de la aplicación limpiar intencionalmente la cache de modo que el sistema vuelva a consultar los datos desde su origen real.

**Ejemplo:**

FwkCache wFwkCache = \_FwkCacheCollectionMannager.GetFwkCache(“Ventas”);

wFwkCache.ClearAll();

**Código 4.0**

**FwkSimpleStorageBase**

Este componente permite mantener la persistencia de objetos de una manera muy sencilla y sin ningún tipo de configuración.

*FwkSimpleStorageBase* no utiliza las Enterprise Library para realizar sus tareas de cache.

No se configura ningún app.config

Es una clase genérica, por lo tanto permite almacenar la información de cualquier tipo de objeto.

Esta clase es comúnmente utilizada en entornos Win32 cuando se desea mantener la persistencia de inputs del usuario.

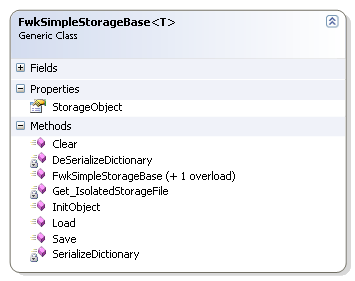
**(SO) Windows**

Isolation Storage

**FwkSimpleStorageBase<T>**

**Fwk.Caching Block**

Propiedades y métodos:



|  |  |
| --- | --- |
|  |  |
| **StorageObject 🡪 T** | Clase generica que reprecenta el objeto serializable que se almacena en la cache del sistema |
| **InitObject** | Si el objeto necesita ser inicializado con algunos valores se debe sobrescribir esta clase donde se inicializa el  StorageObject |
| **Load** | Permite cargar el almacenammiento del objeto.. Este metodo llama al metodo virtual InitObject.  Generalmente se usa desde el Load del formulario |
| **Save**. | Almacena en cache el objeto |

**Como se utiliza?**

Este objeto se puede utilizar de dos maneras. La primera es simplemente declarando un objeto de tipo FwkSimpleStorageBase<T> donde T será el tipo de objeto serializable a almacenar en cache. La segunda es utilizar FwkSimpleStorageBase<T> como clase base y así de esta forma poder sobrescribir el método virtual InitObject

Vamos a un ejemplo para ambas situaciones con el supuesto de que nuestra clase a inicializar y cachear representa ciertos valores de campos de texto en un formulario:

* LastFile (nombre de ultimo archivo accesadoarchivo)
* LastAccess (Ultimo acceso del usuario)

1. **Primera forma**

Si tenemos un formulario y deseamos almacenar en cache una clase llamada FormInit, primero debemos declarar:

FwkSimpleStorageBase<FormInit> \_Storage = new FwkSimpleStorageBase<FormInit>();

* 1. En el Load del Formulario podemos autocargar el storage

\_Storage.Load();

*En este momento el objeto ya esta inicializado y cargado desde la chache si es que en algun momemto se lo almaceno. Si no se encuentra el objeto en cache, el componente automaticamente lo instancia utilizando Fwk.Reflection Components.-*

* 1. Simplemente utilizar:

txtLastFile.Text = \_Storage.StorageObject.LastFile;

* 1. Recuerde siempre ejecutar el método **Save** para informarle al componente **FwkSimpleStorageBase** que debe almacenar en cache el objeto. De lo contrario la proxima vez que levante la aplicación los valores que apareceran en el objeto no representaran los ultimos datos que el usuario ingreso en la aplicación.

Generalmente el método **Save** se llama en el método **Closing** del formulario

1. **Segunda forma**

Existen situaciones donde los valores del objeto (ej: formInit) deben ser pasados como parámetros o inicializados e manera especial. En tal caso podemos hacer una clase Custom que herede de FwkSimpleStorageBase

1)

internal class MyStorageSetting : FwkSimpleStorageBase<**FormInit**>

{

public override void InitObject()

{

base.InitObject();

if (!File.Exists(\_Object.LastFile))

\_**Object**.LastFile = String.Empty;

}

}

T

Aquí fue necesaria la clase MyStorageSetting para intersectar la propiedad LastFile ya que no se desea pintar el campo de texto txtLastFile con un archivo y aInixistente

2)

internal class MyStorageSetting : FwkSimpleStorageBase<**FormInit**>

{

public override void InitObject()

{

~~base.InitObject();~~

\_Object = new Object(new DateTime())

}

}

Aquí el desarrollador lo que necesita es utilizar una sobrecarga del constructor para construir el objeto a cachear. Por lo tanto quita la inicialización automática del componente **FwkSimpleStorageBase**