WebAgenda API Version 0.1

Daniel Kettle, Mark Hazlett, Daniel Wehr

January 12, 2010

Chapter 1

Introduction to the API

Chapter 2

Private Method Calls

2.1 Introduction to Private Method Calls

The following method calls are hidden from the programmer and are called automatically from other usable method calls. Generally they consist of security functions such as permission checking and omitting unnecessary data from database requests.

These method calls with contain less information compared to the higher level method calls as they are straightfoward and interact less with other objects. The pre- and post-conditions for accessing these private methods are not covered as they are out of scope. When a private method is invoked, it is safe to assume that the proper conditions have been met and the result has been properly delivered.

2.2 Cache(Object)

Description

Places an employee's permission information into memory so that it can be retrieved quickly when performing actions that require access to multiple and/or consecutive retrieval calls.

Usage

N/A

Parameters

- Object - Object containing the information identifying the object

Examples

N/A

Notes

Multiple Cache Access

Only one cache per object can exist when dealing with permissions. Multiple caches would not only introduce inconsistencies between permissions if edited simultaneously, but would also be more difficult to control. When dealing with cache control, the employee with the highest permission level will gain cache access: no user can override the higher permission employee‘s changes. A user with cache control overrides users from making conflicting changes while cache changes have not been saved back to the database. Other users will be notified that another employee is making changes to applicable values and will disable their ability to edit those values.

Cache Flushing and Write Persistence

Whenever an employee makes changes to permissions and saves them, the cache is written back to the database through the Broker. If multiple users have access to the cache, all changes that do not conflict with the cache control user‘s decisions will be written as well. Cache control will be passed on to the user with the next-highest permission level. If a user chooses to discard changes made, or does not save changes, the Broker will automatically check periodically for unflushed caches and write them provided (1) changes have been made, and (2) the cache has been marked as ‘old‘. A cache, once retrieved or edited, will stamp itself with an ‘old‘ date, approximately 20 minutes ahead of the current time. The Broker checks this date and if past, will flush data. Once a cache has been flushed by the last remaining user, it is destroyed and only another request for access will regenerate it. If a user is in the middle of editing a cache and it was flushed because another user saves their work, the old date stamp is reset, control is given to the user who has the next highest permission level and previously disabled fields become enabled with newest data. If a higher permissioned user starts editing a field being edited by another user, the higher permissioned user is reminded

if a field is being accessed by another user. If they choose to overwrite currently edited data, the lower permissioned user will have their fields disabled.

Monitoring Cache Access

Caches are monitored by a temporary object in memory:

- Boolean value that represents whether the cache has been modified. Any changes that occur via methods that access the cache will set that value to true.
- String or Date value that represents when the cache should be deleted and the next request get from the database.
- String or Date value that represents the least-recent existing cache (the 'old' cache date) closest to the current time, which sets the timer for the next FlushPermissionCache() call.

Pre-Conditions

1. The Object being requested to load up the cache with must exist. If the Object does not exist, the cache request will fail. This is checked by the object's Broker.
2. There must be enough room in the server's memory to retrieve and store the returned data. This event is out of scope and not covered.

Post-Conditions

1. Cache is saved in memory and any access to the broker's object will be redirected to it.
2. 'Old' date stamp is assigned

Errors

N/A

Related Methods

- Cache(Integer) : uses an Object ID number instead of the entire Object's data.
- FlushCache() : writes data to the database if a change has been made, otherwise will do nothing and exit
- - Broker Methods -

2.3 FlushCache()

Description

Checks for the modification date and a boolean value if cache has been saved; if there has been any modifications done to the cache, changes will be written, otherwise the date is looked at. If the date is past, cache is deleted.

Usage

N/A

Parameters

N/A

Examples

N/A

Notes

Use of this Method

Broker classes start monitoring caches once they are used; They will routinely call this method periodically until the number of caches being monitored drop to zero. The calls are dynamic, the first-initialized cache date is saved in the temporary table and that date minus the current date is when the next call is made. After the call is made, if the cache is deleted (if it hasn't been reset by another user), the next cache is assigned to the next call date. There must be a minimum wait time, such as a one minute period, for the timer to wait before another call is made.

Reason for Using a Cache

Broker classes control how caches are initialized and maintained. A database may also include caches as a performance enabler, but it does not take authority (user permissions) into account for the purpose of this software.

How Caches are Referenced and Called

As a broker controls access to the database, it can (and does) effectively control user access to the data. When a broker is given a request, the broker will load up a table of id‘s for the particular broker‘s content. For an Employee Broker, a table of employee id‘s will be placed into memory alongside the temporary statistics required (See CachePermission(Employee) method). When a request is sent to to the broker, it searches for the requested id in its id cache table, returning those values instead of making a database call. As the id‘s can be in sorted order, an algorithm to find the exact placement of an id would greatly speed up access to the cache table, negating overall access time. If no id is found, a regular database call is made and that id is saved to the cache table, as are the values.

Pre-Conditions

1. The Object-based cache must exist. If the Object does not exist, the cache flush will do nothing.
2. Broker must be able to modify the database if a change must be written. If the database does not exist, a harddrive write of modified data may be utilized for restoration later.

Post-Conditions

1. Temporary data is written back to the database
2. Temporary data is removed from memory
3. Reference to cache data is removed from the employee id cache list

Errors

N/A

Related Methods

- Cache(Object)
- Cache(Integer)
- ClearCache(Object)
- ClearCache(Integer)

2.4 ClearCache(Object)

Description

Removes all data from temporary memory from a specific cache based on the employee specified.

Usage

N/A

Parameters

- Object - Object containing an Object‘s information identifying an existing Object

Examples

N/A

Notes

Use of this Method

Broker classes start monitoring caches once they are used; They will routinely call this method periodically until the number of caches being monitored drop to zero. The calls are dynamic, the first-initialized cache date is saved in the temporary table and that date minus the current date is when the next call is made. After the call is made, if the cache is deleted (if it hasn’t been reset by another user), the next cache is assigned to the next call date. There must be a minimum wait time, such as a one minute period, for the timer to wait before another call is made.

Reason for Using a Cache

Broker classes control how caches are initialized and maintained. A database may also include caches as a performance enabler, but it does not take authority (user permissions) into account for the purpose of this software.

How Caches are Referenced and Called

As a broker controls access to the database, it can (and does) effectively control user access to the data. When a broker is given a request, the broker will load up a table of id‘s for the particular broker‘s content. For an Employee Broker, a table of employee id‘s will be placed into memory alongside the temporary statistics required (See CachePermission(Object) method). When a request is sent to to the broker, it searches for the requested id in its id cache table, returning those values instead of making a database call. As the id‘s can be in sorted order, an algorithm to find the exact placement of an id would greatly speed up access to the cache table, negating overall access time. If no id is found, a regular database call is made and that id is saved to the cache table, as are the values.