Solution for locking across plugins in a distributed system

Hewlett Packard Enterprise: Brian Zimpfer

December 2, 2015

Abstract

This publication describes a method to achieve locking across multiple plugins on a distributed system. Distributed systems may leverage the Observer Patterns Subject and Observer interaction to implement locking at coarse or fine granularity. The granularity implemented in this solution is determined by the implementation of an individual component and decoupled from other components within the same system. This enables decentralized locking structures that may be leveraged independently from a centralized locking architecture. It further decouples code dependencies from a consolidated locking solution and allows components to manage locking independently of other components in the system, while still enabling a coordinated, system-wide locking solution in a distributed manner.

Keywords.
locking – distributed system – observer pattern

Problem

While developing multiple software plugins for a management layer solution, the need arose to communicate locking behavior both between plugins and between nodes on a distributed system. The situation was caused when registry values for protocols needed to be updated. Underlying code that the plugin leveraged was also used by five other plugins. This caused problems during registry updates because the underlying code needed to be restarted, and restarting this code would cause other plugins to fail unexpectedly.

Moreover, because the underlying code would not be in a running state, locking could not be enforced in the underlying code and therefore needed to be enforced in the plugin layer. The need to enforce locking in the plugin layer presented the problem of enforcing locking between independent plugins on a distributed system because of the need to communicate locking and unlocking not only within a single plugin but coordinate locking across multiple plugins in a decentralized manner.

Solution

Our solution involved leveraging existing architecture for other features to create a distributed locking solution between plugins.

This solution leveraged the inter plugin notification system already present in the architecture which implemented a modified observer pattern. This standard design pattern was used as the way for individual plugins to communicate between each other and separate nodes on a distributed system.

The pattern was modified such that there were multiple methods a subject could leverage to notify registered Observers. Two such methods had the ability to send a notification before and send a notification after operations. These notifications were leveraged as lock and unlock indications to system plugins. They allowed individual plugins to lock and unlock themselves, and therefore the distributed system, without necessitating a centralized locking structure.

The send a notification before method was modified from a traditional observer implementation to allow plugins to veto an operation. This is how the equivalent of lock functionality was achieved.

The send a notification after method was modified to signal to plugins that locks should be removed. In the event that the send notification after method was not called the lock would remove locks after five minutes to prevent deadlocking.

The solutions flow path is as follows:

A plugins' API is called to execute defined functionality. That plugin checks its internal collection of locks it maintains to see if that functionality is currently allowed.

If the desired functionality is currently prevented then an exception is returned noting that a lock is held. This is referred to as a ‘veto’.

If no lock is held, then the plugin uses the Observer pattern to send a notification before call to each other plugin in the system.

Each additional plugin will receive the notification after call which may be handled in three different ways:

1. The notification could be ignored. If a plugin has no use for another plugin’s notification then it does not register itself as an Observer of the notifications from the plugin to be ignored. In this scenario the plugin has no effect on locking.
2. The notification may be used to lock the plugin to prevent functionality from occurring under the desired conditions on this plugin. This puts the plugin in a locked state.
3. The plugin may ‘veto’ the functionality indicated by the notification sent from the other plugin. This will return an indication back to the Subject that the originating plugins functionality may not be performed.

If no plugin in the system returns a ‘veto’ then all plugins are in a state of 1 or 2 listed above. In this case the original plugin may perform its functionality. After this functionality has completed the plugin will send a notification after call which each additional plugin will receive and perform operations upon itself as needed. These operations will typically include removing any locks it has placed upon itself.

If any plugin returns a ‘veto’ then the Subject will not perform the API’s intended functionality, but instead send a notification after call to the other plugins, and then will return an exception to indicate that the functionality was prevented.

The send a notification before call is sent to each registered plugin on the system. That plugin is responsible for determining if it has any desire to check and/or add locking requirements relevant to itself and functionality it is responsible for. The send a notification before method allowed a Concrete Observer to return a ‘veto’ that would indicate any functionality a Subject could not perform. This behavior was used to indicate to a plugin that it could not perform its intended functionality because another plugin was in some state such that the desired functionality should be prevented. This was how plugins indicated to other plugins that the system was locked.

For the plugins implementation of locking a collection of Action objects was created that were stored in an array to prevent desired functionality from occurring.

When a notification was received from another plugin that indicated a locking scenario a new Action object was created that corresponded to the action a plugin was notifying it desired to perform. An attempt was made to add the Action to the collection. If any Action currently in the collection prevented the new action from being added then a 'veto' notice was returned back to the Subject. If there was no conflicting action in the collection then the Action was added and it could prevent future Actions from being added to the collection. This created a locking mechanism.

In the instance that a notification after call was not received by a plugin then each plugin was responsible for how it removed its own locking. Our implementation removed Actions that were improperly in its collection by having a time stamp on when an Action was sent. During health checks that ran every five minutes a check was performed to see if an Action had been in the collection for greater than three minutes. If such an Action existed the Action was removed from the collection because it had either failed to send the notification after call or the plugin that sent the action was in a state such that none of its intended functionality could be performed so it should not lock out any other plugins.

References

Observer Pattern: http://en.wikipedia.org/wiki/Observer_pattern
Distributed systems: http://en.wikipedia.org/wiki/Distributed_computing
Locking: http://en.wikipedia.org/wiki/Lock_(computer_science)
Plugin: http://www.webopedia.com/TERM/P/plug_in.html