CacheWriter and CacheListener Best Practices

Geode provides APIs such that a distributed system can capture events, invoking callbacks to process those events either synchronously or asynchronously.

This document covers best practices for the `CacheWriter` and the `CacheListener`.

Event Model

blocked URL

Cache Writers

A `CacheWriter` is an event handler invoked synchronously prior to an event. A cache writer is often used to validate data prior to an update of that data. It may also do a synchronization with external data sources. This provides a write-through capability for regions handling events that can be local, within the same JVM, or remote, in the case of replicated or partitioned region.

Basic rules:

- There can be only **one** `CacheWriter` per region.
- For partitioned regions, the node that hosts the primary bucket of the data will be the one that invokes the cache writer.
- For replicated regions, only the first node to successfully execute the writer will process the event.
- For local regions, only the local cache writer (if defined) will process the event.
- `CacheWriter` can abort operations (fail-fast), and a `CacheWriterException` will propagate back to the caller.
- Being a synchronous callback, it blocks the application's execution until the handler completes.

`CacheWriter` events and callbacks:

```
beforeCreate(EntryEvent event)- Invoked before an entry is created
beforeUpdate(EntryEvent event) - Invoked before an entry is updated
beforeDestroy(EntryEvent event) - Invoked before an entry is destroyed
beforeRegionClear(RegionEvent event) - Invoked before a region is cleared
beforeRegionDestroy(RegionEvent event) - Invoked before a region is destroyed
```

Because CacheWriter handlers are called synchronously, the application does not continue until the handler returns. Therefore, do not do long-running operations inside the handler. If a long-running operation is needed, consider processing the operation asynchronously through an `AsyncEventListener`. Using an `ExecutorService` to delegate the execution to a different thread is possible, but it is an anti-pattern, as it no longer implements the fail-fast property, and the handling of the event is no longer synchronous, so its timing would not be guaranteed relative to the application's completion of the event.

Cache Listeners

A `CacheListener` is an event handler invoked synchronously after modifications to a region occur. The main use cases for a `CacheListener` are synchronous write-behind and notifications. The `CacheListener` can handle cache events related to entries (`EntryEvent`) and regions (`RegionEvent`), but events can be processed in a different order than the order in which they' are applied to the region.

Basic rules:

- You can install **multiple** `CacheListener` handlers in the same region.
- When multiple listeners are installed, the handlers are invoked serially. The invocation ordering is the same as the in which the listeners were registered.
- For partitioned regions, the node that hosts the primary bucket of the data will be the one that invokes the cache listeners.
- For replicated regions, all nodes with the listener installed will process the event.
- For local regions, only local listeners (if defined) will process the event.
- For long running or batch processing, consider using an `AsynchronousEventListener`.
- Being a synchronous callback, the execution of each handler blocks the application's execution until the handler completes.

`CacheListener` events and callbacks:

```
afterCreate(EntryEvent<K,V> event) - Invoked after a new key is added to a region
afterDestroy(EntryEvent<K,V> event) - Invoked after an entry is destroyed
afterInvalidate(EntryEvent<K,V> event) - Invoked after an entry's value is invalidated
afterRegionClear(RegionEvent<K,V> event) - Invoked after a region is cleared
afterRegionCreate(RegionEvent<K,V> event) - Invoked after a region is created
afterRegionDestroy(RegionEvent<K,V> event) - Invoked after a region is destroyed
afterRegionInvalidate(RegionEvent<K,V> event) - Invoked after a region is invalidated
afterRegionLive(RegionEvent<K,V> event) - Invoked after a region becomes live after receiving the marker from
the server
afterUpdate(EntryEvent<K,V> event) - Invoked after an entry's value is modified
```

General recommendations

When dealing with Geode callbacks, there are some operations that should be avoided or used with extra attention. Some general recommendations are:

- Do not perform distributed operations, such as using the *Distributed Lock service.*
- Avoid calling Region methods, particularly on non-colocated, partitioned regions.
- Avoid calling functions through `FunctionService`, since the function's execution can cause distributed deadlock.
- Do not use any Geode APIs inside a `CacheListener` if you have *conserve-sockets* set to true.
- Do not modify region attributes, since those messages will have priority and can cause blocks.
- Avoid configurations in which listeners or writers are deployed in a few nodes of the distributed system. Prefer a cluster-wide installation where every node can process the callback.
- Any exceptions thrown are caught and logged, so users can troubleshoot using Geode logs.
- `EntryEvent.getNewValue()` or `EntryEvent.getOldValue()` can result in deserializations, unless **PDX** and `read-serialized=true` ar e used.
- Operations inside a `CacheListener` or a `CacheWriter` are thread-safe, and entries are locked for the current thread.

When using transactions:

- A `CacheWriter` should not start transactions.
- Both `CacheWriter` and any `CacheListener` will receive all individual operations as part of a transaction, unlike their transactional counterparts `TransactionWriter` and `TransactionListener.`
- When a rollback or commit happens, a `CacheWriter` can only be notified by a `TransactionWriter`, and should handle rollback or failures properly.
- `CacheWriterException` is still propagated to the application, and it should handle the failures in the context of the transaction by continuing or aborting; JTA is the recommended alternative.
- In most cases when dealing with transactions, consider using a `TransactionWriter`, instead of a `CacheWriter.`
- With global transactions, `EntryEvent.getTransactionId()` will return the current internal transaction ID.
- Use the same transactional data source and make sure it is JTA-enabled, so database operations inside a `CacheWriter` can be rolled back and participate in the same global transaction.

When dealing with transactions always consider using `TransactionListener` or `TransactionWriter` for handling transaction events, but do notice that they are cache-wide handlers.

blocked URL