Observation usage patterns

Overview

Analyzing how we use observation in our Sling-based apps shows a number of recurring patterns, described on this page.

Cached Content

Scenario
: In-memory data structures, or "compiled" versions of some content, are created when content changes.

Typical uses
: Configurations, CSS/javascript processing, Sling installer, etc.

Trigger
: Fine-grained detection of changes in a tree of content, based on paths, path regexps, node types or any other meaningful property.

Action
: Clear an internal cache that is rebuilt the next time someone needs it.

Frequency
: Content changes are usually not very frequent, for the above typical uses.

Performance requirements
: Some latency between content changes and processing is usually not a problem.

Potential issues
: N requests coming just after clearing the cache should cause just one cache rebuild, not N.

  Code using this pattern for configurations might rather take advantage of OSGi configurations managed by the Sling installer.

Security considerations
: Loading content in memory with an admin session will make it available to all users. Loading just the paths of the corresponding content items, and letting users retrieve the content themselves, avoids this problem.

Content Export, Replication to Remote Systems

Scenario
: Content is exported as a file or pushed to a remote system when it changes.

This is similar to the Cached Content pattern.

Content Ingestion

Scenario
: Files that are dropped into the repository are parsed or processed, resulting in content changes and/or workflow events.

Typical uses
: Ingesting digital assets, parsing incoming email or other structured files.

Trigger
: A file appears in a watched folder.

Action
: Process the file, create the corresponding content, move the file to a "processed" or "rejected" folder.

Frequency
: Depends on the application.

Performance requirements
: Some latency is usually not a problem, but some applications need to process large number of files quickly.

Potential issues
: Processing partially saved files too early can be a problem, depending on how files are added to the repository.

Security considerations
: Ingestion folders must be properly secured, and incoming content quarantined unless it can be proven safe.

Content Tree Replication

Scenario
: Two or more content trees are kept in sync, usually with customizable mappings and transformations.

Typical uses
: Management of federations of websites, which have some common parts and some specific parts.

Trigger
: Fine-grained detection of changes in a tree of content, based on paths, path regexps, node types or any other meaningful property.

Action
: Replicate the source tree to the target tree(s), optionally applying customizable content transformations.

Frequency
: Source events might be quite frequent depending on authoring activity.

Performance requirements
: Tree transformations might be costly and usually need to run as background jobs.

Potential issues
: An explosion in the number of application-level and repository-level operations is possible depending on the shape of the content tree federation and on the frequency of source content changes.

Aggregation of changes

Scenario
: Collect a number of change events over time and/or for a content subtree, and provide an aggregated view.

Typical uses
: Detect and act on changes to digital assets, without reacting to each and every small change.

Trigger
: Fine-grained detection of changes in a tree of content, based on paths, path regexps, node types or any other meaningful property.

Action
: Store and aggregate events and deliver the results on demand.

Frequency
: Might be quite frequent on a busy content tree.

Performance requirements
: Aggregating events efficiently, as well as storing them until they're not needed anymore, can impact performance.

Potential issues
: Might create a noticeable load on the eventing/observation system, if listening to many detailed events.

Security considerations
: Careless aggregation might expose privileged data.

Consistency Checks and Fixes

Scenario
: Watch specific content subtrees for specific changes (nodes moved etc.) and react to them to avoid inconsistencies in the content.

Typical uses
: Adapt paths that point to other pieces of content when content moves around.

Trigger
: Fine-grained detection of changes in a tree of content, based on paths, path regexps, node types or any other meaningful property.

Action
: Modify content to keep it consistent

Frequency
: Usually not very frequent as that's mostly meant to handle edge cases.

Performance requirements
: A time window during which content can be seen as inconsistent is often unavoidable, keeping that window small is useful.

Potential issues
: Content must not be modified by JCR listeners, that should happen asynchronously if using JCR observation.

Workflow/Job Trigger

Scenario
: Trigger workflows and jobs when content changes.

Typical uses
: The unix print queue system is a good example, with folders named *incoming*, *printing*, *done*, *rejected* that store print job definitions.

Trigger
: Detection of new content items in the *incoming* folder(s).

Action
: Execute the corresponding tasks and move the job definition nodes according to the results.

Frequency
: Depends on the application.

Performance requirements

Depends on the application.

Potential issues

Locking must be used if several job processors are competing for *incoming* jobs.

Distributed processing of those jobs introduces cluster management requirements.

Message Queue

Scenario

Implement a simple message queue backed by a content repository subtree.

This is very similar to the Workflow/Job trigger use case: messages are exchanged between producers and consumers, in the workflow/job trigger case the consumers are job processors.