Content Storage with Apache Jackrabbit

2009-03-25, Jukka Zitting

Introducing JCR

Content Repository for Java™ Technology API
-> Version 1.0 defined in JSR 170, final in 2006
-> Version 2.0 defined in JSR 283, final (hopefully) in 2009
-> Open source RI and TCK developed in Apache Jackrabbit

Flexible, hierarchically ordered content store with features like full text search, versioning, transactions, observation, etc.

Combines and extends features often found in file systems and databases; a "best of both worlds" approach.

Introducing Apache Jackrabbit

Entered Apache Incubator in 2004, graduated in 2006, now a TLP with 24 committers

Version 1.0 (and JCR RI) in 2006, currently at 1.5.3. Version 2.0 (and JCR 2.0 RI) planned for 2009.

Also: JCR Commons, Sling

Components:
jackrabbit-api
jackrabbit-core
jackrabbit-standalone
jackrabbit-webapp
jackrabbit-jca
jackrabbit-ocm
jackrabbit-jcr-rmi
jackrabbit-webdav
jackrabbit-jcr-server
etc.

Structure of a content repository

Consists of one or more workspaces, one of which is the default workspace.

A workspace consists of a tree of nodes, each of which can have zero or more child nodes. Each workspace has a single root node.

Nodes have properties. Properties are typed (string, integer, date, binary, reference, etc.) and can be either single- or multivalued.

Structure, cont.

Each node or property has a name, and can be accessed using a path. Names are namespaced.

A referenceable node has a UUID, through which it can be accessed or referenced. Referential integrity is guaranteed.

Each node has a primary node type and zero or more mixin types. Node types define the structure of a node. A node can also be unstructured.

Content repository features

- Read/write
- Search (XPath and SQL)
- XML import/export
- Observation
- Versioning
- Node types
- Locking
- Access control
- Atomic changes
- XA transactions

Getting started with Jackrabbit

Download and run the standalone server:

java -jar jackrabbit-standalone-1.5.3.jar

-> Web interface at http://localhost:8080/

-> WebDAV at http://localhost:8080/repository/default

-> JCR access over RMI at http://localhost:8080/rmi

-> Repository data in ./jackrabbit

-> Configuration in ./jackrabbit/repository.xml

If you have a servlet container, use the webapp

If you have a J2EE application server, use jca

Remote access

JCR-RMI layer available since Jackrabbit 0.9.

Good functional coverage, not so good performance.
-> administrative tools

Look at clustering for better performance.

Jackrabbit 1.6: spi2dav

```
o.a.j.rmi.repository:
new URLRemoteRepository(
    "http://.../rmi");
new RMIRemoteRepository(
    "//.../repository");
```

Classpath:
jcr-1.0.jar
jackrabbit-jcr-rmi-1.5.3.jar
jackrabbit-api-1.5.3.jar
(also in rmiregistry!)

Jackrabbit as a shared resource

JCA adapter in an application server
-> accessed through JNDI

Jackrabbit webapp in a servlet container
-> JNDI or servlet context
-> JNDI: complex setup
-> cross-context access

JNDI configuration with jackrabbit-servlet:

```
<servlet>
  <servlet-name>Repository</servlet-name>
  <servlet-class>
  org.apache.jackrabbit.servlet.JNDIRepositoryServlet
  </servlet-class>
  <init-param>
    <param-name>location</param-name>
    <param-value<javax/jcr/Repository</param-value>
  </init-param>
</servlet>
```

Accessing the repository:

```
public class MyServlet extends HttpServlet {
    private final Repository repository =
        new ServletRepository(this);
}
```

Jackrabbit in embedded mode

```
RepositoryConfig config = RepositoryConfig.create(
    "/path/to/repository", "/path/to/repository.xml");
RepositoryImpl repository = RepositoryImpl.create(config);
try {
  // Use the javax.jcr interfaces to access the repository
} finally {
  repository.shutdown();
}
```

Embedded mode, cont.

Only a single instance can be running at a time.

For concurrent access:
- multiple sessions
- RMI for remote access
- clustering

Also: TransientRepository

Maven coordinates

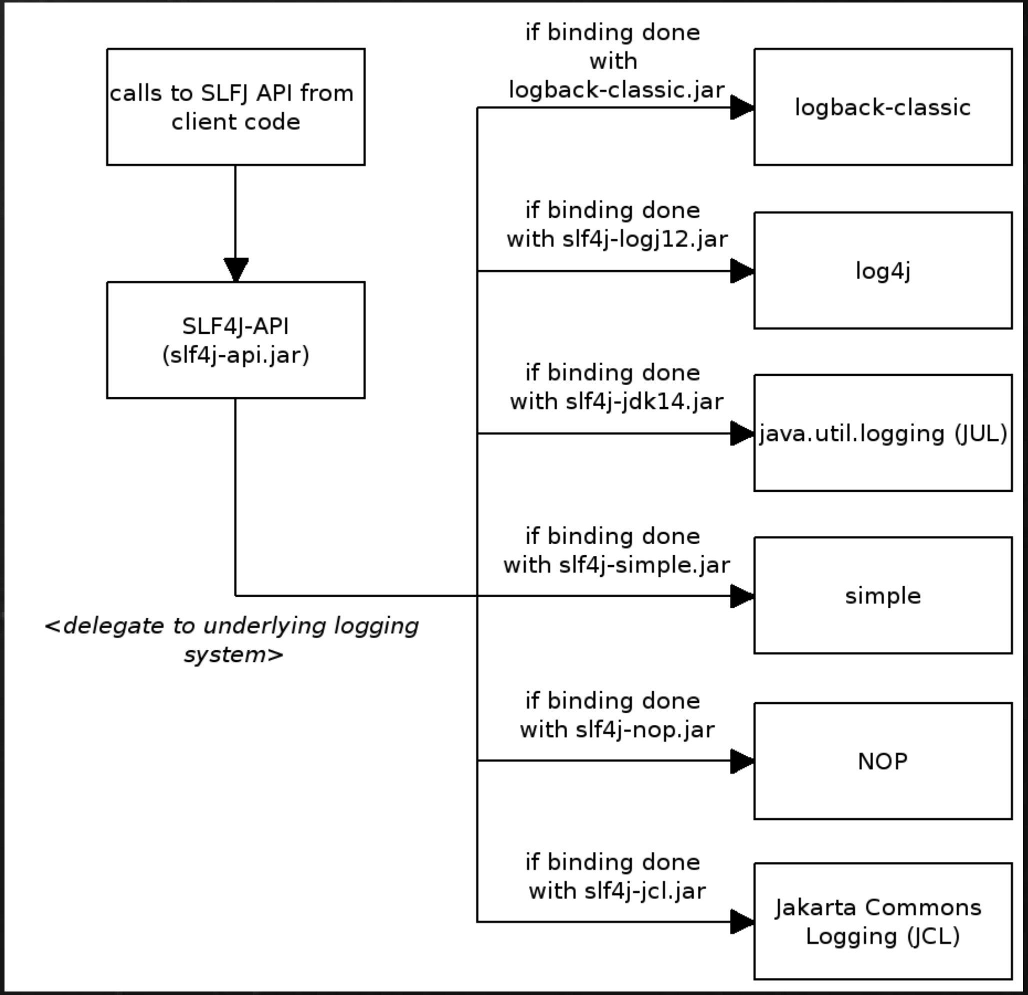
```
<dependency>
  <groupId>org.apache.jackrabbit</groupId>
  <artifactId>jackrabbit-core</artifactId>
  <version>1.5.3</version>
  <exclusions>
    <exclusion>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.5.3</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>jcl-over-slf4j</artifactId>
  <version>1.5.3</version>
</dependency>
```

Logging - what's this SLF4J thing?

Jackrabbit uses the SLF4J logging <u>facade</u> for logging.

Benefits: Great for embedded uses, can adapt to

Drawbacks: What do I put in my classpath?



calls to SLFJ API from client code
SLF4J-API (slf4j-api.jar)
<delegate to underlying logging system>
if binding done with logback-classic.jar
logback-classic
if binding done with slf4j-logj12.jar
log4j
if binding done with slf4j-jdk14.jar
java.util.logging (JUL)
if binding done with slf4j-simple.jar
simple
if binding done with slf4j-nop.jar
NOP
if binding done with slf4j-jcl.jar
Jakarta Commons Logging (JCL)

SLF4J in practice

SLF4J API is automatically included as a dependency of jackrabbit-core.

You need to explicitly add the SLF4J implementation.

Jackrabit webapp, jca and standalone use slf4j-log4j, so you can use normal log4j configuration.

Classpath with log4j:
slf4j-api-1.5.3.jar
slf4j-log4j-1.5.3.jar
log4j-1.2.14.jar
log4j.properties

Classpath with no logging:
slf4j-api-1.5.3.jar
slf4j-nop-1.5.3.jar

Repository configuration

Configuration in a repository.xml file. Default configuration shipped with Jackrabbit. Structure defined in a DTD.

Contains global settings and a workspace configuration <u>template</u>. The template is instantiated to a workspace.xml configuration file for each new workspace.

Main elements: clustering, workspace, versioning, security, persistence, search index, data store, file system

Persistence managers

Use one of the bundle persistence managers.
-> select one for your db

Other persistence managers mostly for backwards compatibility.

Default based on embedded Apache Derby.

Configuration:
driver,url,user,password
schema
schemaObjectPrefix
minBlobSize
bundleCacheSize
consistencyCheck/Fix
blockOnConnectionLoss

Needs CREATE TABLE permissions!

Search index

Per-workspace search indexes based on Apache Lucene.

By default everything is indexed. Use index configuration to customize.

Clustering: Each cluster node maintains local search indexes.

Configuration:
min/maxMergeDocs
mergeFactor
analyzer
textFilterClasses
respectDocumentOrder
resultFetchSize

Performance:
property, type, ft -> fast
path -> slow

Text extraction

Full text indexing of file contents based on various parser libraries (POI, PDFBox, etc.).

Currently only for the jcr:data property with correct jcr:mimeType.

Jackrabbit 2.0: indexing of all binary properties.

textFilterClasses:
PlainTextExtractor
MsWordTextExtractor
MsExcelTextExtractor
MsPowerPointTextExtractor
PdfTextExtractor
OpenOfficeTextExtractor
RTFTextExtractor
HTMLTextExtractor
XMLTextExtractor

Data store - dealing with lots of data

Data store feature available since Jackrabbit 1.4

Content-addressed storage of large binary properties.

Completely transparent to client applications.

Uses garbage collection to remove unused data.

Implementations:
FileDataStore
DbDataStore
sandbox: S3DataStore

Garbage collection:

```
gc = si.createDSGC();
gc.scan();
gc.stopScan();
gc.deleteUnused();
```

Content modeling: David's model

1. Data First. Structure Later. Maybe.
2. Drive the content hierarchy, don't let it happen.
3. Workspaces are for clone(), merge() and update()
4. Beware of Same Name Siblings
5. References considered harmful
6. Files are Files are Files
7. ID's are evil

Content modeling: Example

```
CREATE TABLE author (
   id INTEGER,
   name VARCHAR,
);
CREATE TABLE post (
   id INTEGER,
   author INTEGER,
   posted DATETIME,
   title VARCHAR,
   body TEXT
);
```

```
/blog
  /jukka
    @name = Jukka Zitting
    /2009
      /03
        /25
          /hello
            @author = jukka
            @posted
            @title
            @body
```

Example, cont.

```
CREATE TABLE  comment (
   id INTEGER,
   post INTEGER,
   title VARCHAR,
   body TEXT
);
CREATE TABLE media (
   id INTEGER,
   post INTEGER,
   data BLOB,
   caption VARCHAR
);
```

```
/hello
  /comments
    /salut
      @title = Salut!
      @body
  /media [nt:folder]
    /image.jpg [nt:file]
    /code [nt:folder]
      /Example.java [nt:file]
```

Example, cont.

Versioning:

```
/hello [mix:versionable]
  @jcr:versionHistory
  @jcr:baseVersion

node.checkout();
// ...
node.save();
node.checkin();
```

Locking:

```
/hello [mix:lockable]

node.lock();
// ...
node.unlock();
```

Common issues: Content hierarchy

Jackrabbit doesn't support very flat content hierarchies. You'll start seeing problems when you put more than 10k child nodes under a single parent.

Solution: Add more depth to your hierarchy. Divide entries by date, category, author, etc. If nothing else, use the first letter(s) of a title, a content hash, or even a random number to distribute the nodes.

Note: You can still access the entries as a single flat set through search. The hierarchy is for browsing.

Common issues: Concurrent edits

Three ways to handle concurrent edits:
1. Merge changes
2. Fail conflicting changes
3. Block concurrent changes

Jackrabbit does 1 by default, and falls back to 2 when merge fails. You can explicitly opt for 3 by using the JCR locking feature.

Estimate: How often conflicts would happen? Will the benefits of locking be worth the overhead.

Questions?