Java Content Repository JSR 170, 283

Rakesh Vidyadharan
rakesh@sptci.com

2009-11-17

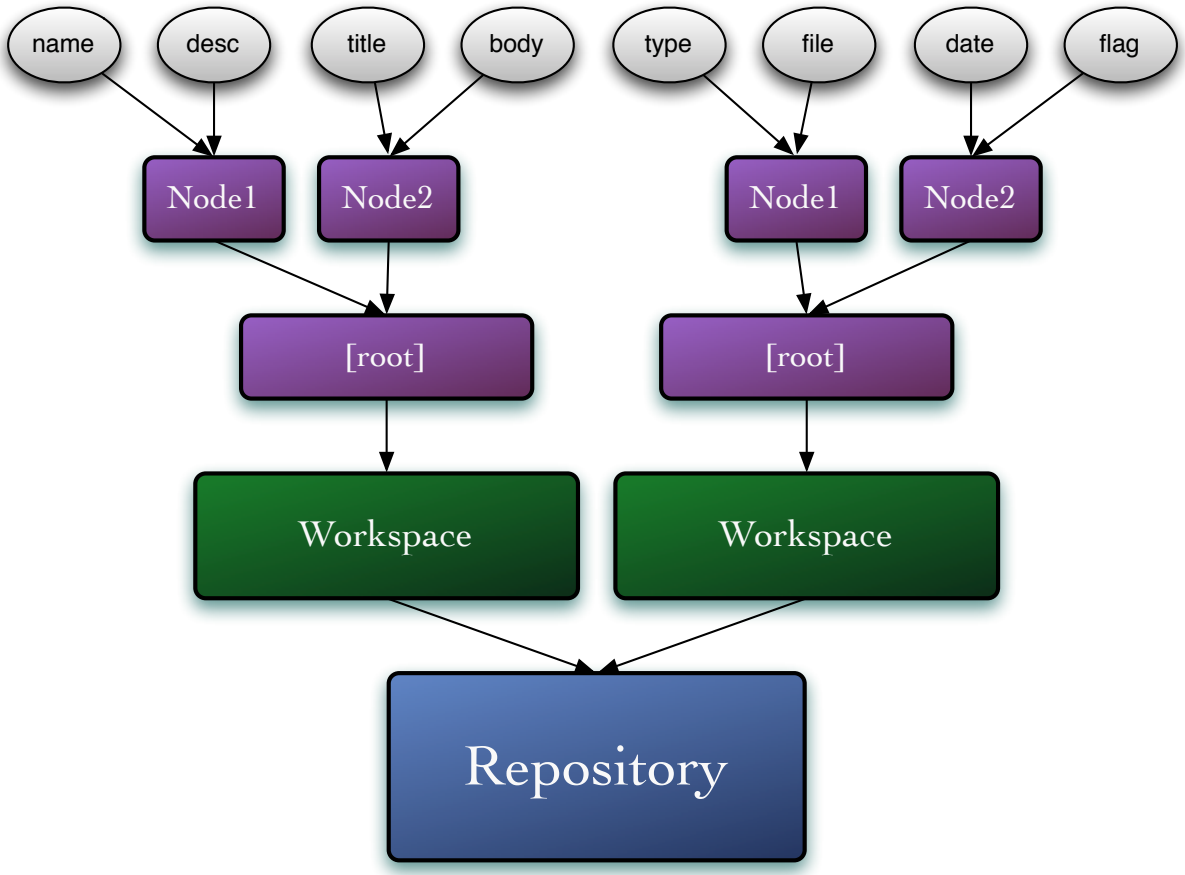
Java Content Repository

- A standard API to unify access to content stored in CMS.
- JSR 170 - JCR version 1, JackRabbit 1.x (currently 1.6)
- JSR 283 - JCR version 2, JackRabbit 2.x (currently 2.0 beta1)
- Content is exposed as Node instances.
  - Nodes have properties.
  - Nodes have child nodes.
- Standard query and full-text search interfaces for content.
- Standard interfaces to version content.
- Implementation levels:

  Level 1 Read-only view of the content via JCR API.

  Level 2 Read-write view of the content via JCR API.

Visual model



Repository Interface

- A repository represents the content storage universe. Equivalent to a database, or a filesystem.
- All access to the content repository starts with a reference to a repository instance.
- Created via constructor or JNDI.
- Usually created/initialised through a configuration file (XML) and a root location. Heavy method that is usually only performed once per application/context.

Repository access

```
import javax.jcr.Repository;
import org.apache.jackrabbit.
    core.TransientRepository;

String conf = "/Users/rakesh/jcr/repo.xml";
String dir = "/var/data/jcr";
Repository rep = new TransientRepository(
      conf, dir );
// Other code
( (TransientRepository) rep ).shutdown();
```

Resource configuration

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
  <Manager pathname='' />
  <Resource
    configFilePath='/var/data/jcr/repository.xml'
    factory='org.apache.jackrabbit.core.
      jndi.BindableRepositoryFactory'
    name='jcr/spt' repHomeDir='/var/data/jcr'
    type='javax.jcr.Repository' auth='Container' />
</Context>
```

Workspace Interface

- Conceptually similar to database schemas, or additional file systems.
- All repositories have a default workspace.
- Additional workspaces may be created as required.
  - No standard way to create/manage workspaces in JSR 170.
  - `create` and `delete` methods added in JSR 283.
  - Standard method to list all accessible workspaces from a login session (see page 15).
- Accessed by specifying the workspace name in call to open a new session (see page 15).
- Workspaces can be cloned completely or partially.

Node Interface

- A node is similar in concept to an XML/JSON element. It can also be thought of in terms of directories and files in a file system.
- A node can have one primary node type (single inheritance).
- A node can optionally expose additional behaviors via Mixin types (implement interfaces).
- Standard methods to list all nodes that reference a node.

Node types

- `nt:unstructured` - The most commonly used node type. Contains no mandatory child nodes or properties.
- `nt:folder` - Represents a directory. Can contain only other `nt:folder` or `nt:file` children.
- `nt:file` - Represents the contents of a file. Used to store binary content such as images, documents, ...
  - `jcr:content` - Mandatory child node of type `nt:resource` that stores the actual binary content.
  - `jcr:mimeType` - Property for MIME type of the binary data.
  - `jcr:encoding` - Property used to represent any content encoding scheme for the binary data.
  - `jcr:data` - Property that contains the binary data.
  - `jcr:lastModified` - Property for the last modification time of the content.

Node mixin types

- `mix:referenceable` - Add if the node is to be referenced from other properties. Adds a UUID property to the node.
- `mix:versionable` - Add if you wish to use revision control features for a node.
- `mix:lockable` - Add if you wish to lock/unlock a node to prevent concurrent modifications.

Node access

```
import javax.jcr.Session;
import javax.jcr.Node;
import javax.jcr.NodeType;
import static org.apache.jackrabbit.JcrConstants.*;

Node root = session.getRootNode();
Node node1 = root.getNode( "parent" );
node1.addMixin( MIX_VERSIONABLE );
boolean exists = root.hasNode( "another" );
Node node2 = node1.addNode( "child" );
NodeType nt = node2.getPrimaryNodeType();
NodeType[] mixin = node1.getMixinNodeTypes();
```

Property Interface

- A property represents data associated with a node.
- Conceptually similar to attributes or simple child elements. Also similar to fields, database columns etc.
- A propery can be single or multi-valued.
- Properties can reference other nodes/properties. Referential integrity checks apply.

Property types

- Constants defined in `javax.jcr.PropertyType`.
- `LONG`
- `DOUBLE`
- `BINARY` - Represents binary data.
- `BOOLEAN`
- `DATE` - Stored as a calendar object.
- `STRING`
- `PATH` - Stored as a string and represents the path to another node. Does not enforce referential integrity.
- `NODE` - Only mix:referenceable nodes may be referenced.

Property API

```
import javax.jcr.Property;
import static javax.jcr.PropertyType.DATE;

boolean exists = node.hasProperty( "title" );
Property p = node.getProperty( "name" );
boolean array = p.getDefinition().isMultiple();
boolean date = p.getType() == DATE;
System.out.format( "value: %s%n", p.getString() );
```

Session Interface

- A session to access a repository.
- A session normally exposes the default workspace within the repository.
- User can specify the name of the workspace they wish to access when opening a session.
- Light-weight object. Can be used per request or execution thread.

Session access

```
import javax.jcr.Session;
import javax.jcr.Credentials;
import javax.jcr.SimpleCredentials;

Credentials cred = new SimpleCredentials( "user",
  "password".toCharArray() );
Session session1 = repository.login( cred );
Session session2 = repository.login(
    cred, workspaceName );
// other code
session1.logout();
session2.logout();
```

Query Interface

JSR 170

- The mandatory language is XPATH. Only a subset of features are supported.
- Repositories may optionally support SQL.

JSR 283

- SQL2 based on SQL-92 standard.
- JQOM Java Query Object Model
- XPath and SQL have been deprecated.

Query access

```
import javax.jcr.NodeIterator;
import javax.jcr.Workspace;
import javax.jcr.query.Query;
import javax.jcr.query.QueryManager;

Workspace ws = session.getWorkspace();
QueryManager qm = ws.getQueryManager();
String query = String.format(
  "//*[jcr:contains(., '%s') order by jcr:score()",
  term );
Query q = qm.createQuery( query, Query.XPATH );
QueryResult result = q.execute();
NodeIterator iter = result.getNodes();
```

Revision Control

- Nodes that adopt the `mix:versionable` type can be versioned.
- Uses familiar `checkin` and `checkout` semantics.
  - `Node.checkin` and `Node.checkout` in JSR 170.
  - `VersionManager.checkin` and `VersionManager.checkout` in JSR 283.
- The VersionHistory and VersionIterator interfaces in javax.jcr.version are commonly used.

Node versioning

```
import javax.jcr.Node;
import javax.jcr.version.VersionHistory;
import javax.jcr.version.VersionIterator;

Node node = session.getRootNode().getNode("test")
node.checkout();
node.setProperty("title", "New title");
session.save();
node.checkin();
VersionHistory history = node.getVersionHistory();
VersionIterator iter = history.getAllVersions();
```

Content Import/Export

- Content can be imported or exported using a standard XML representation.
- Export can be restricted to a node tree, or to just a node without any child nodes.
- Node UUID clashes during import can be handled:
  - Replace the node with the incoming UUID.
  - Remove the node with the incoming UUID.
  - Create a new UUID for incoming nodes. Safest method, but may not be what the application wants.
  - Throw an exception on UUID clash.
- Version history not preserved on import.
- Single value multi-value properties are converted into single value properties on import.

Import/Export API

```
import java.io.FileOutputStream;
import javax.jcr.Session;
import static javax.jcr.ImportUUIDBehavior.
  IMPORT_UUID_COLLISION_REPLACE_EXISTING;

session.exportSystemView( "/",
    new FileOutputStream( file ),
    false, false );
session.getWorkspace().importXML( node.getPath(),
      new FileInputStream( file ),
      IMPORT_UUID_COLLISION_REPLACE_EXISTING );
```

Iterators

- Iterators are provided to iterate over nodes, properties, versions . . . .
- The base iterator interface is `javax.jcr.RangeIterator`.
  - Iterators provide a `getSize()` method.
  - Iterators provide a `skip()` method.
  - Iterator provide a `getPosition()` method.

Registration

- You can register custom node types and namespaces for each workspace.
- Node types may be specified in XML or CND.
  - JSR 170 does not expose any register methods.
  - JSR 283 exposes a couple of register methods. However, these methods require implementations of interfaces for which no default implementations are provided.

Registration API

```
import javax.jcr.NamespaceRegistry;
import org.apache.jackrabbit.
    api.JackrabbitNodeTypeManager;

Workspace ws = session.getWorkspace();
NamespaceRegistry reg = ws.getNamespaceRegistry();
reg.registerNamespace( "prefix", "uri" );
JackRabbitNodeTypeManager nm =
    (JackRabbitNodeTypeManager)
    ws.getNodeTypeManager();
manager.registerNodeTypes(
        new FileInputStream( file ),
        JackrabbitNodeTypeManager.TEXT_XML );
```

Data Modelling

- Model your data as a tree of `nt:unstructured` nodes.
- Keep the tree deep, with minimal child nodes per parent node. This is especially significant for JackRabbit.
- Avoid fixed structures for nodes.
- Try not to use OCM frameworks.
- Use PATH property types instead of REFERENCE.
  - JSR 283 has a WEAKREFERENCE type that does not enforce referential integrity.

Performance

- Try and keep the number of child nodes to below 1000.
- Try and avoid query filters on node path. This is specific to JackRabbit.
- Query performance is not related to database. Queries are executed against the search indices and trees in memory.
- Limit the number of versions you store.
- Store large blobs outside the database.

Miscellaneous Notes

- Access control for nodes and workspaces via JAAS.
- Impersonate as another user after logging in to repository.
- Observation system to register for events in workspace.
- JSR 283 introduces shareable nodes.
- All JCR operations throw a checked RepositoryException.

Resources

Specifications

- JSR 170 Specifications
- JSR 283 Specifications

Reference

- Apache JackRabbit
- Introduction to JCR
- JCR Deep Dive
- JackRabbit Query Performance
- David's Model
- Day, eXo, Hippo, Magnolia, Nuxeo, Alfresco