Exploring the Jackrabbit Java Content Repository

RAPID RABBIT

Speed up your web development with Jackrabbit, an open source implementation of the Java Content Repository standard.

BY CARSTEN ZIEGELER

sad, Fotolia

Databases and conventional data structures are not always a perfect solution for web development. A web application, such as a staff portal on an intranet, for example, has specific requirements for how and when users can access the data. Users have different needs with respect to the data as well. For example, some are interested in targeted content searches, whereas others want to be notified when significant events occur, and all users need access privileges to modify content.

To put it in more general terms: Simply providing content is not enough. Today's users – and today's web developers – expect that various services will accompany the content. For example, web applications often rely on access controls, search functions, and versioning, and, although the developer could build these functions into the application from scratch, the economies of the programming profession cry out for a more efficient approach.

The idea behind the Content Repository API for Java Technology (JCR) is to abstract data-related services from the underlying application and use a standard API to access these service. A content repository avoids the need to continually re-implement data services with each application. Instead, the application simply calls a function through the repository API.

A content repository combines some of the advantages of a filesystem and a database. As a filesystem, it supports hierarchical storage of unstructured files and permissions for access control. As a database, it supports storage of structured data, queries, transactions, and integrity checks. Content repositories also support features such as versioning and change management (Figure 1).

The Content Repository

The full specification for the Content Repository standard is an excellent starting point if you want to get to know the Java Content Repository API [1]. The idea is that the definition of a repository is independent of the underlying data sources, protocols, and architecture. The API is split into two levels. Level 1 provides basic functionality for read access, and level 2 addresses issues related to modifying stored data.

The JCR reference implementation was created by Day Software and then handed over to the Apache Software Foundation. This implementation has since become a successful open source project that goes by the name of Apache Jackrabbit [2]. An active community has grown around the project and continues to push its development. The Jackrabbit repository is a full-fledged implementation of the standard, with a full set of level 1 and level 2 functions. Jackrabbit also adds several extra features, such as the ability to set up a repository cluster.

A web application included with Jackrabbit supports the first few steps of defining a content repository. This web app provides an interface that gives users the ability to set up new repositories.

Apache Jackrabbit supports access to the repository via WebDAV, which makes it easy to mount the repository, copy any kind of files to it, create directories, and manage the repository contents.

Figure 2 shows an overview of the repository model: It has a simple, hierarchical structure as a tree with *n* levels. The central instance is the repository, which can contain one or multiple workspaces. In turn, each workspace contains a tree of items, wherein each item is either a node

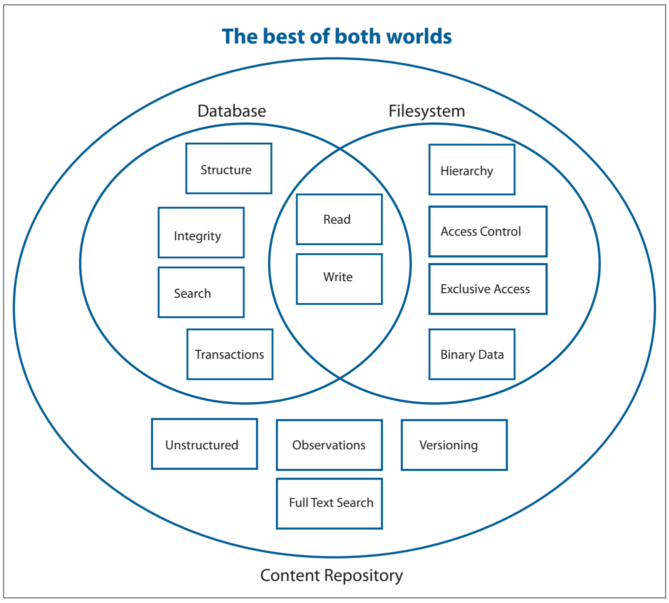

Figure 1: A content repository combines the advantages of databases and filesystems.

or a property. A node can have child nodes, and 0 to *n* properties that store the data (see the "Types" box). A property is typed and contains a data type (string, number, binary string, and so on).

Nodes allow hierarchical data storage of, say, digital photos below a *photos* node. Other nodes in the repository represent photo albums, which can also contain sub-albums. For instance, all photos taken in the year 2008 in Amsterdam might reside below */photos/2008/Amsterdam*. Each item, whether a node or a property, can be uniquely accessed via a path starting at the repository root. Below the *Amsterdam* node are photos, and each photo has its own node. But a content repository is more than an ordinary collection of files and directories. The properties belonging to the individual nodes can include parameters useful to a web application, such as the binary stream for the image, as well as parameters such as the photo date and location. It is up to the developer to structure the data in the repository in the best way for the application. The Jackrabbit documentation and the wiki [3] offer tips and tricks for content modeling.

Listing 1: Accessing the Repository

```
InitialContext jndiContext = ...
Repository repository = (Repository)jndiContext.
lookup("MyRepository");

// Create Credentials
Credentials credentials = new SimpleCredentials(
    "carsten", "secretpassword".toCharArray());

// Open a session
Session session = repository.login(credentials, "Workspace A");
```

Listing 2: Reading and Writing in the Repository

```
// Retrieve album "/photos/2008"
Node knoten = (Node)session.getRootNode().getNode("photos/2008");
// Create new album
Node album = node.addNode("Amsterdam");
// Add property
album.setProperty("Description", "Photos from Amsterdam");
album.setProperty("public_readable", true);

// and finally save
session.save();
```

The Java API

Interacting with the repository requires a couple of initial steps: Application programmers first have to set up a connection to the repository and then create a session for an individual user. All further actions use the session. The first step for accessing the repository is not defined by the standard. Depending on how the infrastructure is implemented, Apache Jackrabbit offers various approaches. One approach uses the Java Naming and Directory Interface (JNDI). The code in Listing 1 creates a session for a fictional user.

Developers can use the session to query and modify nodes. In case of changes, the repository does not save them itself; instead, a message is sent to the session, which means that multiple changes can be saved at the same time. On top of this are explicit transactions. The code in Listing 2 queries a node (i.e., */photos/2008*) in a repository. Below this node, it creates a new *Amsterdam* node, assigns properties, and saves the changes.

The API supports a search function via SQL or XPath to support complex queries. To track changes to the repository, reference the *EventListeners* registered with the session. Users can specify which parts of the repository to monitor during the registration process and restrict notification to specific types and changes. With the last feature, it is easy to launch a specific workflow for a specific type of content in the repository. Different applications or their components can thus react to each other flexibly while remaining loosely connected. For example, many application could store images in the same photo album.

INFO

[1] Java Specification Request 170: *http://www.jcp.org/jsr/detail/170.jsp*

[2] Apache Jackrabbit: *http://jackrabbit.apache.org*

[3] Apache Jackrabbit wiki: *http://wiki.apache.org/jackrabbit/FrontPage*

[4] Apache Sling project: *http://incubator.apache.org/sling*

[5] Java Specification Request 283 *http://jcp.org/en/jsr/detail?id=283*

[6] REST explained: *http://www.xfront.com/REST-Web-Services.html*

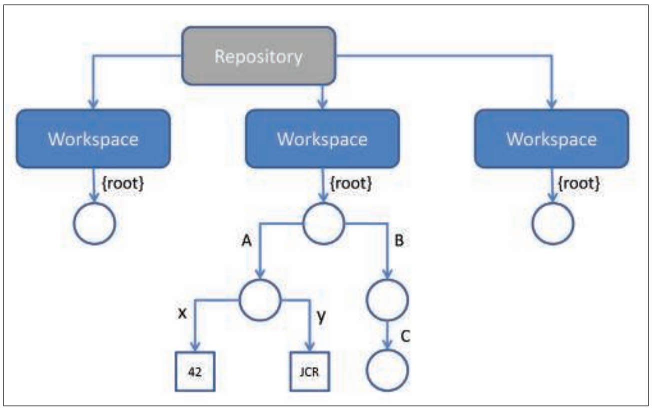

Figure 2: The repository model: The workspace is filled with nodes A, B, and C. Node A has the properties x and y.

The new Apache Sling framework [4] is based on a content repository for REST-based applications. Each browser query is matched with content in the repository, and a script for displaying the content is selected in a second step.

Development: JSR 283

Version 2 of the standard is currently under construction as JSR 283. The new standard, which is planned for a 2008 release date, will include a number of extensions. The major focus is on improving repository management and administrative functionality. For example, the current standard does not concern itself with access control and the type management of nodes. JSR 283 will close these gaps [5].

Future

The JCR project has made the name JCR synonymous with content repository. A content repository is deployed as an infrastructure with defined services. Users only need to concern themselves with the application, and a single application can easily access different content repositories.

The flexibility of JCR is an enormous boon for developers, in that programmers can learn a single API suited to a variety of application cases. In addition, Jackrabbit removes dependencies on vendors and reduces the cost of migration. ■

Types

Each node has exactly one primary type. This type defines the node structure, for example, specifying which properties or children the node can possess. Besides the primary type, a node can have any number of mixins. A mixin is also a type definition that can add properties to any node. Each application can define its own types. The combination of multiple inheritance and mixins supports extremely flexible and precise type definition. The standard defines a number of types, such as *nt:unstructured*, with which arbitrary trees made of nodes and properties are permitted.

It is generally a good idea to use existing types and then add your own requirements. For a photo album, this would mean using an existing type such as *nt:folder*, which describes a directory, with your own mixin that contains additional information for the album. At the same time, images could use *nt:file* and a mixin for special photo data. The use of standard types also offers the advantage of helping third-party applications handle the data more easily.