(/)

Getting Started with Java and Zookeeper

Last modified: November 5, 2018

by baeldung (/author/baeldung/)

DevOps (/category/devops/) **Library (/category/library/)**

1. Overview

Apache ZooKeeper (https://zookeeper.apache.org/) **is a distributed coordination service** which eases the development of distributed applications. It's used by projects like Apache Hadoop, HBase and others (https://cwiki.apache.org/confluence/display/ZOOKEEPER/PoweredBy) for different use cases like leader election, configuration management, node coordination, server lease management, etc.

Nodes within ZooKeeper cluster store their data in a shared hierarchal namespace which is similar to a standard file system or a tree data structure.

In this article, we'll explore how to use Java API of Apache Zookeeper to store, update and delete information stored within ZooKeeper.

2. Setup

The latest version of the Apache ZooKeeper Java library can be found here (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.apache.zookeeper%22%20AND%20a%3A%22zookeeper%22):

```
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.4.11</version>
</dependency>
```

3. ZooKeeper Data Model – ZNode

ZooKeeper has a hierarchal namespace, much like a distributed file system where it stores coordination data like status information, coordination information, location information, etc. This information is stored on different nodes.

Every node in a ZooKeeper tree is referred to as ZNode.

Each ZNode maintains version numbers and timestamps for any data or ACL changes. Also, this allows ZooKeeper to validate the cache and to coordinate updates.

4. Installation

4.1. Installation

Latest ZooKeeper release can be downloaded from here (https://www.apache.org/dyn/closer.cgi/zookeeper/). Before doing that we need to make sure we meet the system requirements described here (https://zookeeper.apache.org/doc/r3.3.5/zookeeperAdmin.html#sc_systemReq).

4.2. Standalone Mode

For this article, we'll be running ZooKeeper in a standalone mode as it requires minimal configuration. Follow the steps described in the documentation here (https://zookeeper.apache.org/doc/r3.3.5/zookeeperStarted.html#sc_InstallingSingleMode).

Note: In standalone mode, there's no replication so if ZooKeeper process fails, the service will go down.

5. ZooKeeper CLI Examples

We'll now use the ZooKeeper Command Line Interface (CLI) to interact with ZooKeeper:

```
1 bin/zkCli.sh -server 127.0.0.1:2181
```

Above command starts a standalone instance locally. Let's now look at how to create a ZNode and store information within ZooKeeper:

```
[zk: localhost:2181(CONNECTED) 0] create /MyFirstZNode
ZNodeVal
Created /FirstZnode
```

We just created a ZNode *'MyFirstZNode'* at the root of ZooKeeper hierarchical namespace and written *'ZNodeVal'* to it.

Since we've not passed any flag, a created ZNode will be persistent.

Let's now issue a *'get'* command to fetch the data as well as metadata associated with a ZNode:

```
[zk: localhost:2181(CONNECTED) 1] get /FirstZnode

“Myfirstzookeeper-app”
cZxid = 0x7f
ctime = Sun Feb 18 16:15:47 IST 2018
mZxid = 0x7f
mtime = Sun Feb 18 16:15:47 IST 2018
pZxid = 0x7f
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 22
numChildren = 0
```

We can update the data of an existing *ZNode* using the *set* operation.

For example:

```
1 set /MyFirstZNode ZNodeValUpdated
```

This will update the data at *MyFirstZNode* from *ZNodeVal* to *ZNodeValUpdated.*

6. ZooKeeper Java API Example

Let's now look at Zookeeper Java API and create a node, update the node and retrieve some data.

6.1. Java Packages

The ZooKeeper Java bindings are composed mainly of two Java packages:

1. *org.apache.zookeeper*: which defines the main class of the ZooKeeper client library along with many static definitions of the ZooKeeper event types and states
2. *org.apache.zookeeper.data*: that defines the characteristics associated with ZNodes, such as Access Control Lists (ACL), IDs, stats, and so on

There's also ZooKeeper Java APIs are used in server implementation such as *org.apache.zookeeper.server*, *org.apache.zookeeper.server.quorum*, and *org.apache.zookeeper.server.upgrade*.

However, they're beyond the scope of this article.

6.2. Connecting to a ZooKeeper Instance

Let's now create *ZKConnection* class which will be used to connect and disconnect from an already running ZooKeeper:

```
public class ZKConnection {
    private ZooKeeper zoo;
    CountDownLatch connectionLatch = new CountDownLatch(1);

    // ...

    public ZooKeeper connect(String host)
      throws IOException,
      InterruptedException {
        zoo = new ZooKeeper(host, 2000, new Watcher() {
            public void process(WatchedEvent we) {
                if (we.getState() == KeeperState.SyncConnected) {
                    connectionLatch.countDown();
                }
            }
        });

        connectionLatch.await();
        return zoo;
    }

    public void close() throws InterruptedException {
        zoo.close();
    }
}
```

To use a ZooKeeper service, an application must first instantiate an object of ***ZooKeeper* class, which is the main class of ZooKeeper client library**.

In *connect* method, we're instantiating an instance of *ZooKeeper* class. Also, we've registered a callback method to process the *WatchedEvent* from ZooKeeper for connection acceptance and accordingly finish the *connect* method using *countdown* method of *CountDownLatch*.

Once a connection to a server is established, a session ID gets assigned to the client. To keep the session valid, the client should periodically send heartbeats to the server.

The client application can call ZooKeeper APIs as long as its session ID remains valid.

6.3. Client Operations

We'll now create a *ZKManager* interface which exposes different operations like creating a ZNode and saving some data, fetching and updating the ZNode Data:

```
public interface ZKManager {
    public void create(String path, byte[] data)
      throws KeeperException, InterruptedException;
    public Object getZNodeData(String path, boolean watchFlag);
    public void update(String path, byte[] data)
      throws KeeperException, InterruptedException;
}
```

Let's now look at the implementation of the above interface:

```
public class ZKManagerImpl implements ZKManager {
    private static ZooKeeper zkeeper;
    private static ZKConnection zkConnection;

    public ZKManagerImpl() {
        initialize();
    }

    private void initialize() {
        zkConnection = new ZKConnection();
        zkeeper = zkConnection.connect("localhost");
    }

    public void closeConnection() {
        zkConnection.close();
    }

    public void create(String path, byte[] data)
      throws KeeperException,
      InterruptedException {

        zkeeper.create(
          path,
          data,
          ZooDefs.Ids.OPEN_ACL_UNSAFE,
          CreateMode.PERSISTENT);
    }

    public Object getZNodeData(String path, boolean watchFlag)
      throws KeeperException,
      InterruptedException {

        byte[] b = null;
        b = zkeeper.getData(path, null, null);
        return new String(b, "UTF-8");
    }

    public void update(String path, byte[] data) throws KeeperExcepti
      InterruptedException {
        int version = zkeeper.exists(path, true).getVersion();
        zkeeper.setData(path, data, version);
    }
}
```

In the above code, *connect* and *disconnect* calls are delegated to the earlier created *ZKConnection* class. Our *create* method is used to create a ZNode at given path from the byte array data. For demonstration purpose only, we've kept ACL completely open.

Once created, the ZNode is persistent and doesn't get deleted when the client disconnects.

The logic to fetch ZNode data from ZooKeeper in our *getZNodeData* method is quite straightforward. Finally, with the *update* method, we're checking the presence of ZNode on given path and fetching it if it exists.

Beyond that, for updating the data, we first check for ZNode existence and get the current version. Then, we invoke the *setData* method with the path of ZNode, data and current version as parameters. ZooKeeper will update the data only if the passed version matches with the latest version.

7. Conclusion

When developing distributed applications, Apache ZooKeeper plays a critical role as a distributed coordination service. Specifically for use cases like storing shared configuration, electing the master node, and so on.

ZooKeeper also provides an elegant Java-based API's to be used in client-side application code for seamless communication with ZooKeeper ZNodes.

And as always, all sources for this tutorial can be found over on Github (https://github.com/eugenp/tutorials/tree/master/apache-zookeeper).

Learning to build your API

with Spring?

Enter your email address

>> Get the eBook

CATEGORIES

SPRING (/CATEGORY/SPRING/)
REST (/CATEGORY/REST/)
JAVA (/CATEGORY/JAVA/)
SECURITY (/CATEGORY/SECURITY-2/)
PERSISTENCE (/CATEGORY/PERSISTENCE/)
JACKSON (/CATEGORY/JACKSON/)
HTTP CLIENT (/CATEGORY/HTTP/)
KOTLIN (/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)

SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)

SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)

THE COURSES (HTTPS://COURSES.BAELDUNG.COM)

CONSULTING WORK (/CONSULTING)

META BAELDUNG (HTTP://META.BAELDUNG.COM/)

THE FULL ARCHIVE (/FULL_ARCHIVE)

WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)

CONTACT (/CONTACT)

EDITORS (/EDITORS)

ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)

PRIVACY POLICY (/PRIVACY-POLICY)

COMPANY INFO (/BAELDUNG-COMPANY-INFO)