Explaining Apache Zookeeper

Apache Zookeeper is a highly consistent, scalable and reliable cluster co-ordination service. ZooKeeper itself is a distributed service that is ideal for Configuration management, Naming service, providing distributed synchronization, leader election and group services. It is a open source service that reliably coordinates distributed processes.
Your distributed clustered applications can make use of Zookeeper to store,manage and propagate updates to required cluster configuration and management data.

Table of Contents

Coordination in Distributed Applications

Cluster coordination in distributed application is a complex process. Large scale distributed systems needs to manage and maintain a number of vital attributes to keep the distributed system available and healthy.

The Problem Detection in Distributed Applications

Detecting node level events is very important to make any distributed system reliable and available. Node crash/failure detectors should identify and notify other nodes of the cluster about the event. Also the availability detectors needs to maintains information about the availability history of nodes in the system.
A master node (in a master-slave) or any node (in master-master) in a distributed environment needs to know about the approximate number of non-faulty nodes present as a part of its group.

Configuration Management

Managing the configuration common to all nodes in the distributed group is another critical task. Each node having its own copy of configuration makes the system less easy to maintain.

Leader Election and Consensus

Leader Election among a group of nodes based on any protocol requires consensus and this is a difficult to manage a in dynamic distributed system as members may leave or fail or join any time.

Explaining Apache Zookeeper

As mentioned in the beginning, Zookeeper is a distributed system coordination service. While considering CAP Theorem, Zookeeper is a CP system where Consistency and Partition tolerance are guaranteed.

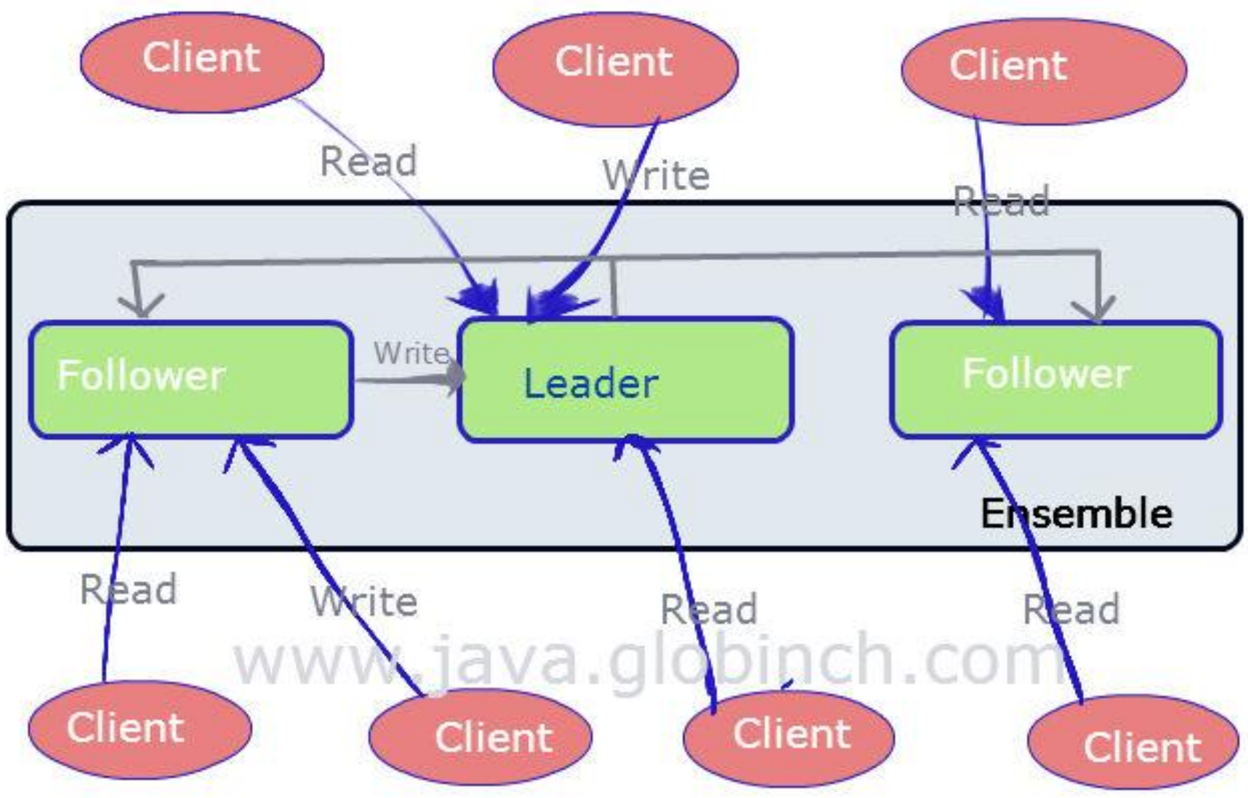
CAP Theorem

```
CAP Theorem (Brewer’s theorem/ By Eric Brewer) states that, it is nearly impossible to achieve all three of Consistency, Availability and Partition tolerance. Any distributed system can choose only two of the following:

1. Consistency : All nodes is a distributed environment see the same data at the same time.

2. Availability: Guarantee that every request receives a response.

3. Partition Tolerance: Even if the connections between nodes are down, the other nodes works as guaranteed. (Tolerance to network partitions/broken communication)
```

Zookeeper relaxes availability, but guarantees consistency and partition tolerance. Zookeeper implements an API that manipulates simple waitfree data objects organized hierarchically as in file systems.Pipelined architecture of ZooKeeper enables the execution of operations from a single client in FIFO order.

Zookeeper Architecture

The Zookeeper service comprise of an Ensemble of servers. Ensemble uses replication to achieve high availability and extreme performance. A Zookeeper cluster, Ensemble, consists of a Leader node and followers. A leader node is chosen by consensus within the ensemble. If the leader fails another node



Zookeeper Data Model :ZNode

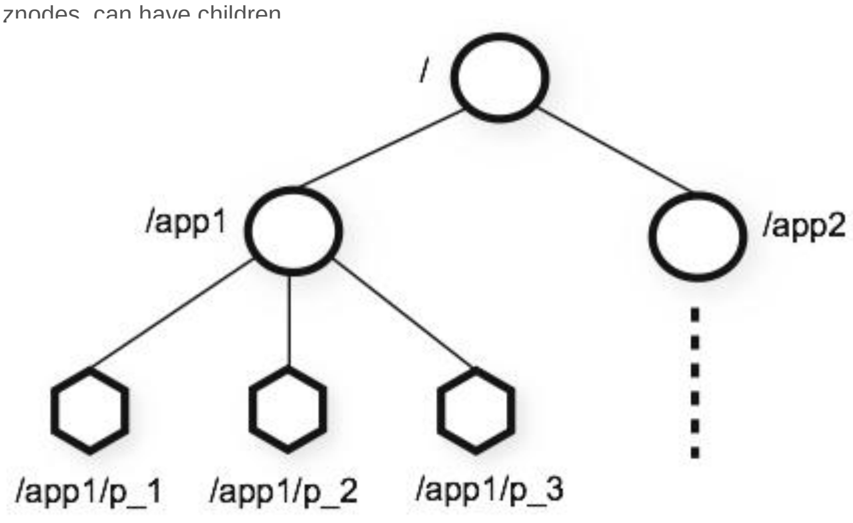
ZooKeeper provides to its clients the abstraction of a set of data nodes (znodes), organized according to a hierarchical name space. The znodes in this hierarchy are data objects that clients manipulate through the ZooKeeper API.

Types of ZNodes

There are three types of ZNodes:

```
Regular: Clients manipulate regular znodes by creating and deleting them explicitly;

Ephemeral: Clients create such znodes, and they either delete them explicitly, or let the system remove them
automatically when the session that creates them terminates.

Sequential: These Znodes when created, gets a unique number (sequence) suffixed to its name.
```

Each ZNode is using UNIX style notation. For example, /A/B/C to denote the path to znode C, where C has B as its parent and B has A as its parent. All znodes store data, and all znodes,except for ephemeral znodes, can have children



ZAB : Zookeeper Atomic Protocol

All requests that update ZooKeeper state are forwarded to the leader. The leader executes the request and broadcasts the change to the ZooKeeper state using an Zookeeper Atomic Broadcast (ZAB) protocol. The server that
receives the client request responds to the client when it delivers the corresponding state change. Zab uses by default
simple majority quorums to decide on a proposal, so Zab and thus ZooKeeper can only work if a majority of servers are correct.
A Message is a sequence of bytes to be atomically broadcast to all ZooKeeper servers. A message put into a Proposal and agreed upon before it is delivered. Total ordering is achieved using a ZooKeeper transaction id (zxid). All
proposals will be stamped with a zxid when it is proposed and exactly reflects the total ordering.
ZooKeeper messaging operates the similar way a classic two phase commit works. Proposals are sent to all ZooKeeper servers and committed when a quorum of them acknowledge the proposal.

Zookeeper Leader Election

Zookeeper leader activation includes leader election. A leader becomes active only when a quorum of followers has synced up with the leader. The quorum counts leader as a follower as well. All these servers in the ensemble will have the same state. This state consists of all of the proposals that the leader believes have been committed and the proposal to follow the leader, the NEW_LEADER proposal. Leader election process results in a single server designated as a leader. Leader start waiting for followers to connect. Once the follower connection is completed the Zookeeper state synchronization process begins. The leader will sync up with followers by sending any proposals followers are missing. If a follower is missing too many proposals, leader will send a full snapshot of the state to the follower.

Zookeeper Sessions

Zookeeper sessions are attached to the connected clients. Zookeeper session is initiated when a client connects to ZooKeeper with an associated timeout. Session ends when clients explicitly close a session handle or ZooKeeper detects that a clients is faulty.
Zookeeper considers a client faulty if it does not receive anything from its session for more than the configured timeout.

Zookeeper Read,Write, Events and Watch

All requests that update ZooKeeper state (Writes)are forwarded to the leader. The leader executes the request and
broadcasts the change to the ZooKeeper state through Zab. Zab by default uses a simple majority quorums to decide on a proposal. All read requests are serviced from the local replica of each server data tree.
ZooKeeper supports the concept of watches. Clients can set a watch on a znodes. A watch event will be triggered and removed when the znode changes. When a watch is triggered the client receives a packet saying that the znode has changed.

ZooKeeper High Availability

ZooKeeper provides high availability by replicating the ZooKeeper state (data) on each server that composes the service. The replicated database is an in-memory database containing the entire data tree (snapshot). Each znode in the tree stores a maximum of 1MB of data by default. Another way ZooKeeper achieves high throughput is by keeping the request processing pipeline full.

How to Setup Zookeeper

It is easy to get the Zookeeper download and setup in your workstation. Setting up a ZooKeeper server in standalone mode is straightforward and the server is contained in a single JAR file.

Zookeeper Download and Setup

You can download a stable release from Apache Zookeeper site.
http://zookeeper.apache.org/releases.html
The configuration file is usually named as “zoo.cfg” but you can name it the way you want.
To start the Zookeeper

Running Zookeeper standalone mode

Open the configuration fle and update the following minimum properties.

```
tickTime=2000 //Tick time is the used for heart beats.Minimum session timeout will be twice the tickTime.

dataDir=/var/somdir/zookeeper/data  //Location to store the in-memory database snapshots and transaction log

clientPort=2181  // Port to listen for client connections
```

To run the zookeeper in standalone mode, navigate to /bin/ folder and use the following command
bin/zkServer.sh start
This will bring up the Zookeeper in standalone mode.

Running Replicated ZooKeeper

A replicated group of servers in the same application is called a quorum, and in replicated mode, all servers in the quorum have copies of the same configuration file.
For replicated mode, a minimum of three servers are required, and it is strongly recommended that you have an odd number of servers. This is because when you use even number of servers, for example 2 servers, you are in a situation where if one of them fails, there are not enough machines to form a majority quorum. Two servers is inherently less stable than a single server, because there are two single points of failure.
Here is an example of zoo.cfg entries of replicated ZooKeeper servers.

```
tickTime=2000

dataDir=/var/lib/zookeeper

clientPort=2181

initLimit=5   //Length of time the ZooKeeper servers in quorum have to connect to a leader

syncLimit=2          //How far out of date a server can be from a leader.

server.1=zoo1:2888:3888                      //Server Id,  Peers communication port and Port for leader
election.

server.2=zoo2:2888:3888

server.3=zoo3:2888:3888
```

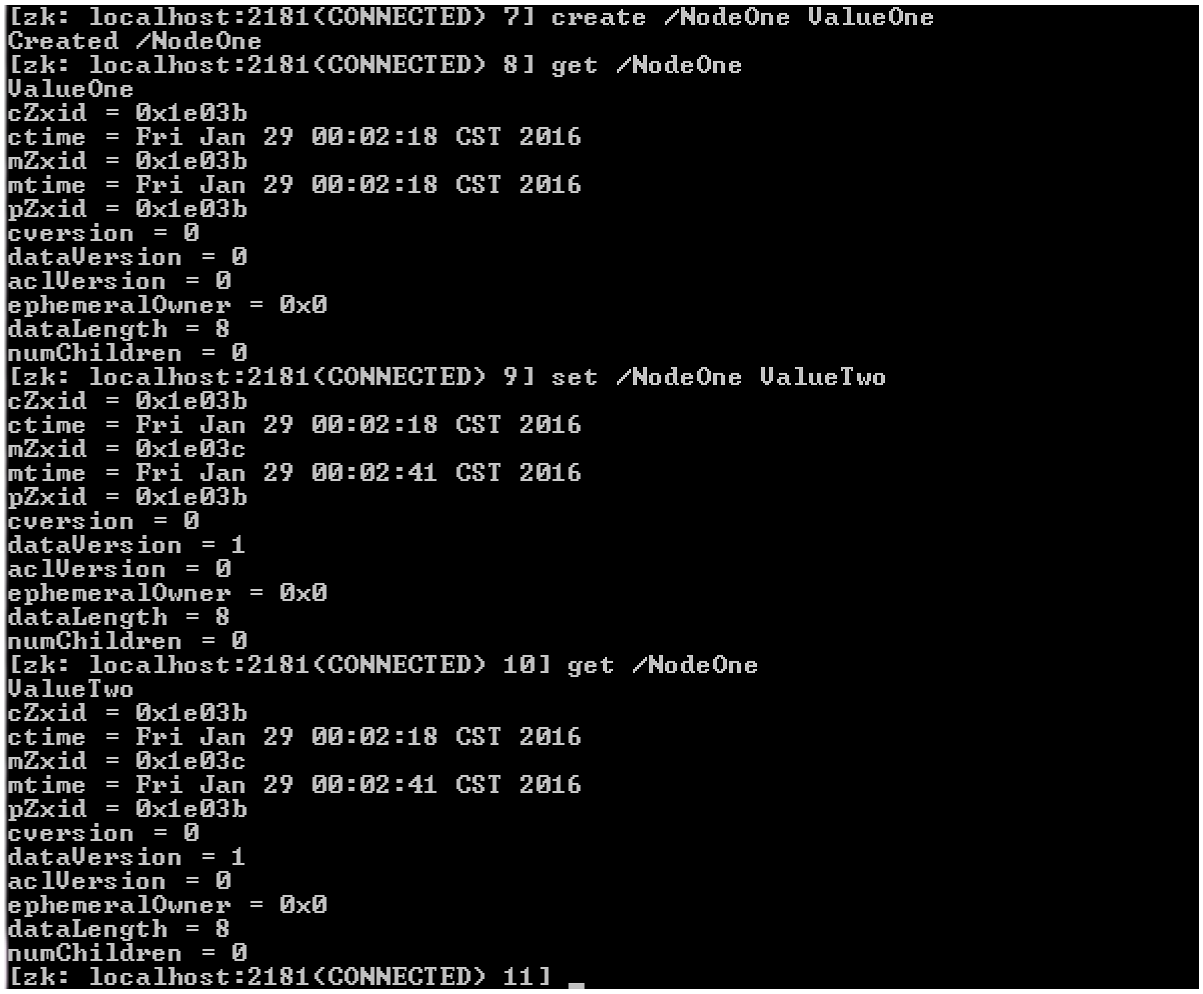
Note: If you are trying a replicated mode in single server, be sure to use different ports for each Zookeeper instances.

Zookeeper CLI examples

Assuming that you are running Zookeeper in local mode. Go to bin folder and issue the command
bin/zkCli.sh -server 127.0.0.1:2181
This will connect to the Zookeeper server.
To know all the available commands, type help.

```
[zk: localhost:2181(CONNECTED) 0] help

ZooKeeper -server host:port cmd args

        stat path [watch]

        set path data [version]

        ls path [watch]

        delquota [-n|-b] path
        ls2 path [watch]

        setAcl path acl

        setquota -n|-b val path

        history

        redo cmdno

        printwatches on|off

        delete path [version]

        sync path

        listquota path

        rmr path

        get path [watch]

        create [-s] [-e] path data acl

        addauth scheme auth

        quit

        getAcl path

        close

        connect host:port

[zk: localhost:2181(CONNECTED) 1]
```

Let us create a znode ‘NodeOne’ and write ‘ValueOne’ using the ‘create’ command. After that issue the ‘get’ command to get the details of the node. You can then update the ZNode using ‘set’ command. See the commands and their results below.



```
[zk: localhost:2181(CONNECTED) 7] create /NodeOne ValueOne
Created /NodeOne
[zk: localhost:2181(CONNECTED) 8] get /NodeOne
ValueOne
cZxid = 0x1e03b
ctime = Fri Jan 29 00:02:18 CST 2016
mZxid = 0x1e03b
mtime = Fri Jan 29 00:02:18 CST 2016
pZxid = 0x1e03b
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 8
numChildren = 0
[zk: localhost:2181(CONNECTED) 9] set /NodeOne ValueTwo
cZxid = 0x1e03b
ctime = Fri Jan 29 00:02:18 CST 2016
mZxid = 0x1e03c
mtime = Fri Jan 29 00:02:41 CST 2016
pZxid = 0x1e03b
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 8
numChildren = 0
[zk: localhost:2181(CONNECTED) 10] get /NodeOne
ValueTwo
cZxid = 0x1e03b
ctime = Fri Jan 29 00:02:18 CST 2016
mZxid = 0x1e03c
mtime = Fri Jan 29 00:02:41 CST 2016
pZxid = 0x1e03b
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 8
numChildren = 0
[zk: localhost:2181(CONNECTED) 11] _
```

Zookeeper Java API Example

ZooKeeper has an official API binding for Java and C. ZooKeeper API provides both synchronous and asynchronous methods and can be used to create an application that can connect, interact, manipulate data, coordinate, and finally disconnect from a ZooKeeper ensemble to achieve variety of co-ordination services.
Here is a simple example that shows how to use the Zookeeper API to create a node, update the node, retrive the data and remove it.

ZooKeeper Connection Class:ZKConnection.java

```java
/*

 * @(#)ZKConnection.java

 * @author Binu George

 * Globinch.com

 * copyright http://www.java.globinch.com. All rights reserved.

 */

package com.globinch.zoo.client;
```

```java
import java.io.IOException;
import java.util.concurrent.CountDownLatch;

import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.Watcher.Event.KeeperState;
import org.apache.zookeeper.ZooKeeper;

/**
 * ZKConnection Class. Class that initialize connection to Ensemble
 *
 * @author Binu George
 * @since 2016
 * @version 1.0
 * http://www.java.globinch.com. All rights reserved
 */
public class ZKConnection {

	// Local Zookeeper object to access ZooKeeper ensemble
	private ZooKeeper zoo;
	final CountDownLatch connectionLatch = new CountDownLatch(1);

	/**
	 *
	 */
	public ZKConnection() {
		// TODO Auto-generated constructor stub
	}

	/**
	 * Initialize the Zookeeper connection
	 * @param host
	 * @return
	 * @throws IOException
```

```
             * @throws InterruptedException

             */

            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;

            }


            // Method to disconnect from zookeeper server

            public void close() throws InterruptedException {

                        zoo.close();

            }


}
```

Zookeeper Client Operation Interface: ZKManager.java

```
/*

 * @(#)ZKManager.java

 * @author Binu George

 * Globinch.com

 * copyright http://www.java.globinch.com. All rights reserved.

 */

package com.globinch.zoo.client;


import java.util.List;
import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.data.Stat;


/**

 * ZKManager Interface. Defines the methods to manipulate znodes

 *

 * @author Binu George

 * @since 2016

 * @version 1.0

 * http://www.java.globinch.com. All rights reserved

 */

public interface ZKManager {

           /**

            * Create a Znode and save some data

            *

            * @param path

            * @param data

            * @throws KeeperException

            * @throws InterruptedException

            */

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

                                 InterruptedException;


           /**

            * Get the ZNode Stats

            *

            * @param path

            * @return Stat

            * @throws KeeperException

            * @throws InterruptedException

            */

           public Stat getZNodeStats(String path) throws KeeperException,

                                 InterruptedException;
```

```
/**

 * Get ZNode Data

 *

 * @param path

 * @param boolean watchFlag

 * @throws KeeperException

 * @throws InterruptedException

 */

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

                       InterruptedException;


/**

 * Update the ZNode Data

 *

 * @param path

 * @param data

 * @throws KeeperException

 * @throws InterruptedException

 */

public void update(String path, byte[] data) throws KeeperException,

                       InterruptedException;


/**

 * Get ZNode children

 *

 * @param path

 * @throws KeeperException

 * @throws InterruptedException

 * return List<String>

 */

public List<String> getZNodeChildren(String path) throws KeeperException,

                       InterruptedException;


/**

 * Delete the znode
```

```
            *
            * @param path
            * @throws KeeperException
            * @throws InterruptedException
            */
           public void delete(String path) throws KeeperException,
                                      InterruptedException;
}
```

Zookeeper Client Operation class: ZKClientManagerImpl.java

```
/*
 * @(#)ZKClientManagerImpl.java
 * @author Binu George
 * Globinch.com
 * copyright http://www.java.globinch.com. All rights reserved.
 */
package com.globinch.zoo.client;

import java.util.List;

import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.ZooDefs;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.data.Stat;

/**
 * ZKClientManagerImpl class. Implements the methods to manipulate znodes.
 *
 * @author Binu George
 * @since 2016
 * @version 1.0 http://www.java.globinch.com. All rights reserved
 */
public class ZKClientManagerImpl implements ZKManager {

           private static ZooKeeper zkeeper;
	private static ZKConnection zkConnection;

	/**
	 *
	 */
	public ZKClientManagerImpl() {
		initialize();
	}

	/**
	 * Initialize connection
	 */
	private void initialize() {
		try {
			zkConnection = new ZKConnection();
			zkeeper = zkConnection.connect("localhost");

		} catch (Exception e) {
			System.out.println(e.getMessage());
		}
	}

	/**
	 * Close the zookeeper connection
	 */
	public void closeConnection() {
		try {
			zkConnection.close();
		} catch (InterruptedException e) {
			System.out.println(e.getMessage());
		}
	}

	@Override
```

```
        public void create(String path, byte[] data) throws KeeperException,
                        InterruptedException {
                zkeeper.create(path, data, ZooDefs.Ids.OPEN_ACL_UNSAFE,
                                CreateMode.PERSISTENT);

        }

        @Override
        public Stat getZNodeStats(String path) throws KeeperException,
                        InterruptedException {
                Stat stat = zkeeper.exists(path, true);
                if (stat != null) {
                        System.out.println("Node exists and the node version is "
                                        + stat.getVersion());
                } else {
                        System.out.println("Node does not exists");
                }
                return stat;
        }

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



                try {


                        Stat stat = getZNodeStats(path);
                        byte[] b = null;
                        if (stat != null) {
                                if(watchFlag){
                                        ZKWatcher watch = new ZKWatcher();
                                         b = zkeeper.getData(path, watch,null);
                                         watch.await();
                                }else{
```

```
                                                    b = zkeeper.getData(path, null,null);
                                            }
                                            /*byte[] b = zkeeper.getData(path, new Watcher() {

                                                        public void process(WatchedEvent we) {

                                                                    if (we.getType() ==
Event.EventType.None) {
                                                                                switch (we.getState()) {
                                                                                case Expired:

            connectedSignal.countDown();
                                                                                            break;
                                                                                }

                                                                    } else {

                                                                                try {
                                                                                            byte[] bn =
zkeeper.getData(path, false, null);
                                                                                            String data =
new String(bn, "UTF-8");

            System.out.println(data);

            connectedSignal.countDown();

                                                                                } catch (Exception ex) {

            System.out.println(ex.getMessage());
                                                                                }
                                                                    }
                                                        }
                                            }, null);*/

                                            String data = new String(b, "UTF-8");
                                            System.out.println(data);
```

```
                return data;
            } else {
                System.out.println("Node does not exists");
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

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

    }

    @Override
    public List<String> getZNodeChildren(String path) throws KeeperException,
            InterruptedException {
        Stat stat = getZNodeStats(path);
        List<String> children  = null;

        if (stat != null) {
            children = zkeeper.getChildren(path, false);
            for (int i = 0; i < children.size(); i++)
                System.out.println(children.get(i));

        } else {
            System.out.println("Node does not exists");
        }
        return children;
    }
```

```

            @Override
            public void delete(String path) throws KeeperException,
                                   InterruptedException {
                        int version = zkeeper.exists(path, true).getVersion();
                        zkeeper.delete(path, version);

            }

            /**
             * @param args
             */
            public static void main(String[] args) {
                        // TODO Auto-generated method stub

            }

}
```

Zookeeper Watcher: ZKWatcher.java

```
/**
 *
 */
package com.globinch.zoo.client;

import java.util.concurrent.CountDownLatch;

import org.apache.zookeeper.AsyncCallback.StatCallback;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.data.Stat;

/**
 * @author bpaikay
 *
 */
```

```
public class ZKWatcher implements Watcher, StatCallback {

           CountDownLatch latch;


           /**

            *

            */

           public ZKWatcher() {

                       latch = new CountDownLatch(1);

           }


           @Override

           public void processResult(int rc, String path, Object ctx, Stat stat) {

                       // TODO Auto-generated method stub


           }


           @Override

           public void process(WatchedEvent event) {

                        System.out.println("Watcher fired on path: " + event.getPath() + " state: " +

                 event.getState() + " type " + event.getType());

         latch.countDown();

           }


           public void await() throws InterruptedException {

         latch.await();

     }


}
```

Test Out Zookeeper API example Client

```
/*

 * @(#)ZKClientTest.java

 * @author Binu George

 * Globinch.com

 * copyright http://www.java.globinch.com. All rights reserved.

 */
```

```
package com.globinch.zoo.client.test;


import static org.junit.Assert.*;


import java.util.List;


import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.data.Stat;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;


import com.globinch.zoo.client.ZKClientManagerImpl;


/**
 * ZKClientTest Test Class
 *
 * @author Binu George
 * @since 2016
 * @version 1.0
 * http://www.java.globinch.com. All rights reserved
 */
public class ZKClientTest {


	private static ZKClientManagerImpl zkmanager = new ZKClientManagerImpl();
	// ZNode Path
	private String path = "/QN-GBZnode";
	byte[] data = "www.java.globinch.com ZK Client Data".getBytes();


	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
	}
```

```
/**
 * @throws java.lang.Exception
 */
@After
public void tearDown() throws Exception {
}

/**
 * Test method for
 * {@link com.globinch.zoo.client.ZKClientManagerImpl#create(java.lang.String, byte[])}
 * .
 *
 * @throws InterruptedException
 * @throws KeeperException
 */
@Test
public void testCreate() throws KeeperException, InterruptedException {
           // data in byte array


           zkmanager.create(path, data);
           Stat stat = zkmanager.getZNodeStats(path);
           assertNotNull(stat);
           zkmanager.delete(path);
}

/**
 * Test method for
 * {@link com.globinch.zoo.client.ZKClientManagerImpl#getZNodeStats(java.lang.String)}
 * .
 *
 * @throws InterruptedException
 * @throws KeeperException
 */
@Test
```

```java
public void testGetZNodeStats() throws KeeperException,
                InterruptedException {
        zkmanager.create(path, data);
        Stat stat = zkmanager.getZNodeStats(path);
        assertNotNull(stat);
        assertNotNull(stat.getVersion());
        zkmanager.delete(path);

}

/**
 * Test method for
 * {@link com.globinch.zoo.client.ZKClientManagerImpl#getZNodeData(java.lang.String)}
 * .
 * @throws InterruptedException
 * @throws KeeperException
 */
@Test
public void testGetZNodeData() throws KeeperException, InterruptedException {
        zkmanager.create(path, data);
        String data = (String)zkmanager.getZNodeData(path,false);
        assertNotNull(data);
        zkmanager.delete(path);
}

/**
 * Test method for
 * {@link com.globinch.zoo.client.ZKClientManagerImpl#update(java.lang.String, byte[])}
 * .
 * @throws InterruptedException
 * @throws KeeperException
 */
@Test
public void testUpdate() throws KeeperException, InterruptedException {
        zkmanager.create(path, data);
```

```
            String data = "www.java.globinch.com Updated Data";
            byte[] dataBytes = data.getBytes();
            zkmanager.update(path, dataBytes);
            String retrivedData = (String)zkmanager.getZNodeData(path,false);
            assertNotNull(retrivedData);
            zkmanager.delete(path);
    }

    /**
     * Test method for
     * {@link com.globinch.zoo.client.ZKClientManagerImpl#getZNodeChildren(java.lang.String)}
     * .
     * @throws InterruptedException
     * @throws KeeperException
     */
    @Test
    public void testGetZNodeChildren() throws KeeperException, InterruptedException {
            zkmanager.create(path, data);
            List<String> children= zkmanager.getZNodeChildren(path);
            assertNotNull(children);
            zkmanager.delete(path);
    }

    /**
     * Test method for
     * {@link com.globinch.zoo.client.ZKClientManagerImpl#delete(java.lang.String)}
     * .
     * @throws InterruptedException
     * @throws KeeperException
     */
    @Test
    public void testDelete() throws KeeperException, InterruptedException {
            zkmanager.create(path, data);
            zkmanager.delete(path);
            Stat stat = zkmanager.getZNodeStats(path);
```

```
                    assertNull(stat);

          }



}
```

Apache Zookeeper Use Cases :Where and how to use it

ZooKeeper offers the library to create and manage synchronization primitives.Since it is a distributed service,ZooKeeper avoids the single-point-of-failure. Typical use cases includes Leader Election implementation, Distributed Locks implementation, Barrier implementation etc. Any system that needs a centralized reliable service to manage its configuration across distributed environment can make use of ZooKeeper.
Mainly, you will find ZooKeeper ideal choice for the following typical use cases.

1. Naming service
2. Configuration management
3. Synchronization
4. Leader election
5. Message Queue
6. Notification system

Here are a list of most popular examples where Apache ZooKeeper is used.

1. Apache Hadoop relies on ZooKeeper for automatic fail-over of Hadoop HDFS Namenode and for the high availability of YARN ResourceManager.
2. Apache HBase, a distributed database built on Hadoop, uses ZooKeeper for master election, lease management etc.
3. Apache Solr uses ZooKeeper for leader election and centralized configuration

Apache Zookeeper Alternatives

ZooKeeper is used by extensively and is definitly the most popular cluster co-ordiation solution. There are other services with more or less same features.

1. etcd
2. Eureka – Netflix
3. Consul

History of Zookeeper

Zookeeper is originally developed by Yahoo research Team and became a top level Apache project.ZooKeeper is used by many enterprises including Rackspace, Yahoo! and eBay.

References

1. Apache ZooKeeper
2. Apache Zookeeper Examples
3. ZooKeeper Programmer’s Guide
4. Eric Brewer CAP Twelve Years Later: How the “Rules” Have Changed

Incoming search terms:

- zookeeper processResult examples
- oadd org apache drill exec rpc RpcException: Failure setting up ZK for client
- apache zookeeper use cases
- apache zoo
- apache zookeeper examples
- how to update data in znode in zookeeper api
- zookeeper apache
- zookeeper data directory is missing at /var/lib/zookeeper
- zookeeper example
- zookeeper java client example