Apache ZooKeeper Essentials

Apache ZooKeeper provides a simple interface to a centralized coordinating service for distributed applications, through a rich set of APIs and primitives, enabling developers to concentrate on the core logic of their applications. With ZooKeeper, it's unnecessary to start from scratch, so the development of distributed applications becomes cleaner and easier.

Starting with how to install, configure, and begin with ZooKeeper, we'll examine the intricacies of ZooKeeper's architecture and internals. Next, you'll learn ZooKeeper's programming model and write code to solve common distributed coordination tasks. We'll also walk you through administering ZooKeeper in a production environment, before finally covering practical examples of real-world projects and services using ZooKeeper to solve complex day-to-day problems.

Who this book is written for

Whether you are a novice to ZooKeeper or already have some experience, you will be able to master the concepts of ZooKeeper and its usage with ease.

This book assumes you to have some prior knowledge of distributed systems and high-level programming knowledge of C, Java, or Python, but no experience with Apache ZooKeeper is required.

What you will learn from this book

- Understand how Apache ZooKeeper solves coordination issues in traditional distributed systems
- Discover how to set up and get started with ZooKeeper in a development environment in addition to production
- Explore ZooKeeper's architecture and internals
- Get to grips with the API model of ZooKeeper to start programming
- Use ZooKeeper to solve common distributed coordination tasks such as leader election, distributed queues, and group memberships
- Administer Apache ZooKeeper for real-world use and production workload
- Investigate the use of ZooKeeper in real-world applications and services

Saurav Haloi

A fast-paced guide to using Apache ZooKeeper to coordinate services in distributed systems

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 “A Crash Course in Apache ZooKeeper”
- A synopsis of the book’s content
- More information on Apache ZooKeeper Essentials

About the Author

Saurav Haloi works as a principal software engineer at EMC in its data protection and availability division. With more than 10 years of experience in software engineering, he has also been associated with prestigious software firms such as Symantec Corporation and Tata Consultancy Services, where he worked in the design and development of complex, large-scale, multiplatform, multi-tier, and enterprise software systems in a storage, networking, and distributed systems domain. He has been using Apache ZooKeeper since 2011 in a variety of different contexts. He graduated from National Institute of Technology, Surathkal, India, with a bachelors degree in computer engineering. An open source enthusiast and a hard rock and heavy metal fanatic, he lives in the city of Pune in India, which is also known as the Oxford of the East.

I would like to thank my family for their support and encouragement throughout the writing of this book.

It was a pleasure to work with Packt Publishing, and I would like to thank everyone associated with this book: the editors, reviewers, and project coordinators, for their valuable comments, suggestions, and assistance during the book development period. Special thanks to Ajinkya Paranjape, my content development editor, who relentlessly helped me while writing this book and patiently answered all my queries relating to the editorial processes.

I would also like to thank the Apache ZooKeeper contributors, committers, and the whole community for developing such a fantastic piece of software and for their continuous effort in getting ZooKeeper to the shape it is in now. Kudos to all of you!
Apache ZooKeeper Essentials

Architecting and building a distributed system is not a trivial job, and implementing coordination systems for the distributed applications is even harder. They are often prone to errors such as race conditions and deadlocks, and such bugs are not easily detectable. Apache ZooKeeper has been developed with this objective in mind, to simplify the task of developing coordination and synchronization systems from scratch. ZooKeeper is an open source service, which enables high performance and provides highly available coordination services for distributed applications.

Apache ZooKeeper is a centralized service, which exposes a simple set of primitives that distributed applications can build on, in order to implement high-level services such as naming, configuration management, synchronization, group services, and so on. ZooKeeper has been designed to be easily programmable with its simple and elegant set of APIs and client bindings for a plethora of languages.

Apache ZooKeeper Essentials takes readers through an enriching practical journey of learning ZooKeeper and understanding its role in developing scalable and robust distributed applications. It starts with a crisp description of why building coordination services for distributed applications is hard, which lays the stepping stone for the need to know and learn ZooKeeper. This book then describes the installation and configuration of a ZooKeeper instance, after which readers will get a firsthand experience of using it.

This book covers the core concepts of ZooKeeper internals, its administration, and the best practices for its usage. The ZooKeeper APIs and the data model are presented in the most comprehensive manner for both beginners and experts, followed by programming with ZooKeeper. Examples of developing client applications have been given in three languages: Java, C, and Python. A full chapter has been dedicated to discuss the various ZooKeeper recipes so that readers get a vivid understanding of how ZooKeeper can be used to carry out common distributed system tasks.

This book also introduces readers to two projects: Curator and Exhibitor, which are used to ease the use of ZooKeeper in client applications and its management in production. Real-world examples of software projects that use ZooKeeper have been cited for readers to understand how ZooKeeper solves real problems. This is followed by examples of organizations that use ZooKeeper in their production platforms and enterprise software systems.

Apache ZooKeeper Essentials will help readers learn everything they need to get a firm grasp of ZooKeeper so that they can start building scalable and high-performant distributed applications with ease and full confidence.
What This Book Covers

Chapter 1, *A Crash Course in Apache ZooKeeper*, introduces you to distributed systems and explains why getting distributed coordination is a hard problem. It then introduces you to Apache ZooKeeper and explains how ZooKeeper solves coordination problems in distributed systems. After this, you will learn how to install and configure ZooKeeper, and get ready to start using it.

Chapter 2, *Understanding the Inner Workings of Apache ZooKeeper*, discusses the architecture of ZooKeeper and introduces you to its data model and the various operations supported by it. This chapter then delves deeper into the internals of ZooKeeper so that you understand how various components of ZooKeeper function in tandem.

Chapter 3, *Programming with Apache ZooKeeper*, introduces you to programming with the ZooKeeper client libraries and explains how to develop client applications for ZooKeeper in Java, C, and Python. This chapter presents ready-to-compile code for you to understand the nitty-gritty of ZooKeeper programming.

Chapter 4, *Performing Common Distributed System Tasks*, discusses the various recipes of distributed system tasks such as locks, queues, leader election, and so on. After going through these recipes, you will understand how ZooKeeper can be used to solve common coordination problems that are often encountered while building distributed systems.

Chapter 5, *Administering Apache ZooKeeper*, provides you with all the information that you need to know about the administration and configuration of ZooKeeper. It also presents the best practices of ZooKeeper usage and the various ways to monitor it.

Chapter 6, *Decorating ZooKeeper with Apache Curator*, cites details about two projects, Curator and Exhibitor, that make ZooKeeper programming and management easier and simpler.

Chapter 7, *ZooKeeper in Action*, discusses examples of real-world software systems, which use ZooKeeper at its core to carry out their functionalities. This chapter also presents examples of how various organizations are using ZooKeeper in their distributed platforms to solve coordination and synchronization problems and to build scalable and highly performant systems.
In the past couple of decades, the Internet has changed the way we live our lives. Services offered over the Internet are often backed up by complex software systems, which span over a large number of servers and are often located geographically apart. Such systems are known as distributed systems in computer science terminology. In order to run these large systems correctly and efficiently, processes within these systems should have some sort of agreement among themselves; this agreement is also known as distributed coordination. An agreement by the components that constitute the distributed system includes the overall goal of the distributed system or an agreement to accomplish some subtasks that ultimately lead to the goal. This is not as simple as it sounds, because the processes must not only agree but also know and be sure about what their peers agree to.

Although coordinating tasks and processes in a large distributed system sounds easy, it is a very tough problem when it comes to implementing them correctly in a fault-tolerant manner. Apache ZooKeeper, a project of the Apache Software Foundation, aims to solve these coordination problems in the design and development of distributed systems by providing a set of reliable primitives through simple APIs.

In this chapter, we will cover the following topics:

- What a distributed system is and its characteristics
- Why coordination in a distributed system is hard
- An introduction to Apache ZooKeeper
- Downloading and installing Apache ZooKeeper
- Connecting to ZooKeeper with the ZooKeeper shell
- Multinode ZooKeeper cluster configuration
Defining a distributed system

A distributed system is defined as a software system that is composed of independent computing entities linked together by a computer network whose components communicate and coordinate with each other to achieve a common goal. An e-mail system such as Gmail or Yahoo! Mail is an example of such a distributed system. A multiplayer online game that has the capability of being played by players located geographically apart is another example of a distributed system.

In order to identify a distributed system, here are the key characteristics that you need to look out for:

- **Resource sharing**: This refers to the possibility of using the resources in the system, such as storage space, computing power, data, and services from anywhere, and so on
- **Extendibility**: This refers to the possibility of extending and improving the system incrementally, both from hardware and software perspectives
- **Concurrency**: This refers to the system's capability to be used by multiple users at the same time to accomplish the same task or different tasks
- **Performance and scalability**: This ensures that the response time of the system doesn't degrade as the overall load increases
- **Fault tolerance**: This ensures that the system is always available even if some of the components fail or operate in a degraded mode
- **Abstraction through APIs**: This ensures that the system's individual components are concealed from the end users, revealing only the end services to them

It is difficult to design a distributed system, and it's even harder when a collection of individual computing entities are programmed to function together. Designers and developers often make some assumptions, which are also known as fallacies of distributed computing. A list of these fallacies was initially coined at Sun Microsystems by engineers while working on the initial design of the **Network File System (NFS)**; you can refer to these in the following table:

<table>
<thead>
<tr>
<th>Assumptions</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>The network is reliable</td>
<td>In reality, the network or the interconnection among the components can fail due to internal errors in the system or due to external factors such as power failure.</td>
</tr>
<tr>
<td>Latency is zero</td>
<td>Users of a distributed system can connect to it from anywhere in the globe, and it takes time to move data from one place to another. The network's quality of service also influences the latency of an application.</td>
</tr>
</tbody>
</table>
Assumptions | Reality
---|---
Bandwidth is infinite | Network bandwidth has improved many folds in the recent past, but this is not uniform across the world. Bandwidth depends on the type of the network (T1, LAN, WAN, mobile network, and so on).
The network is secure | The network is never secure. Often, systems face denial of-service attacks for not taking the security aspects of an application seriously during their design.
Topology doesn't change | In reality, the topology is never constant. Components get removed/added with time, and the system should have the ability to tolerate such changes.
There is one administrator | Distributed systems never function in isolation. They interact with other external systems for their functioning; this can be beyond administrative control.
Transport cost is zero | This is far from being true, as there is cost involved everywhere, from setting up the network to sending network packets from source to destination. The cost can be in the form of CPU cycles spent to actual dollars being paid to network service providers.
The network is homogeneous | A network is composed of a plethora of different entities. Thus, for an application to function correctly, it needs to be interoperable with various components, be it the type of network, operating system, or even the implementation languages.

Distributed system designers have to design the system keeping in mind all the preceding points. Beyond this, the next tricky problem to solve is to make the participating computing entities, or independent programs, coordinate their actions. Often, developers and designers get bogged down while implementing this coordination logic; this results in incorrect and inefficient system design. It is with this motive in mind that Apache ZooKeeper is designed and developed; this enables a highly reliable distributed coordination.

Apache ZooKeeper is an effort to develop a highly scalable, reliable, and robust centralized service to implement coordination in distributed systems that developers can straightaway use in their applications through a very simple interface to a centralized coordination service. It enables application developers to concentrate on the core business logic of their applications and rely entirely on the ZooKeeper service to get the coordination part correct and help them get going with their applications. It simplifies the development process, thus making it more nimble.
With ZooKeeper, developers can implement common distributed coordination tasks, such as the following:

- Configuration management
- Naming service
- Distributed synchronization, such as locks and barriers
- Cluster membership operations, such as detection of node leave/node join

Any distributed application needs these kinds of services one way or another, and implementing them from scratch often leads to bugs that cause the application to behave erratically. Zookeeper mitigates the need to implement coordination and synchronization services in distributed applications from scratch by providing simple and elegant primitives through a rich set of APIs.

**Why coordination in a distributed system is so challenging**

After getting introduced to Apache ZooKeeper and its role in the design and development of a distributed application, let's drill down deeper into why coordination in a distributed system is a hard problem. Let's take the example of doing configuration management for a distributed application that comprises multiple software components running independently and concurrently, spanning across multiple physical servers. Now, having a master node where the cluster configuration is stored and other worker nodes that download it from this master node and auto configure themselves seems to be a simple and elegant solution. However, this solution suffers from a potential problem of the master node being a single point of failure. Even if we assume that the master node is designed to be fault-tolerant, designing a system where change in the configuration is propagated to all worker nodes dynamically is not straightforward.

Another coordination problem in a distributed system is service discovery. Often, to sustain the load and increase the availability of the application, we add more physical servers to the system. However, we can get the client or worker nodes to know about this change in the cluster memberships and availability of newer machines that host different services in the cluster is something. This needs careful design and implementation of logic in the client application itself.
Scalability improves availability, but it complicates coordination. A horizontally scalable distributed system that spans over hundreds and thousands of physical machines is often prone to failures such as hardware faults, system crashes, communication link failures, and so on. These types of failures don't really follow any pattern, and hence, to handle such failures in the application logic and design the system to be fault-tolerant is truly a difficult problem.

Thus, from what has been noted so far, it's apparent that architecting a distributed system is not so simple. Making correct, fast, and scalable cluster coordination is hard and often prone to errors, thus leading to an overall inconsistency in the cluster. This is where Apache ZooKeeper comes to the rescue as a robust coordination service in the design and development of distributed systems.

**Introducing Apache ZooKeeper**

Apache ZooKeeper is a software project of the Apache Software Foundation; it provides an open source solution to the various coordination problems in large distributed systems. ZooKeeper was originally developed at Yahoo!


ZooKeeper, as a centralized coordination service, is distributed and highly reliable, running on a cluster of servers called a ZooKeeper ensemble. Distributed consensus, group management, presence protocols, and leader election are implemented by the service so that the applications do not need to reinvent the wheel by implementing them on their own. On top of these, the primitives exposed by ZooKeeper can be used by applications to build much more powerful abstractions to solve a wide variety of problems. We will dive deeper into these concepts in Chapter 4, *Performing Common Distributed System Tasks*.

Apache ZooKeeper is implemented in Java. It ships with C, Java, Perl, and Python client bindings. Community-contributed client libraries are available for a plethora of languages such as Go, Scala, Erlang, and so on.

A full listing of the client bindings for ZooKeeper can be found at [https://cwiki.apache.org/confluence/display/ZOOKEEPER/ZKClientBindings](https://cwiki.apache.org/confluence/display/ZOOKEEPER/ZKClientBindings).
Apache ZooKeeper is widely used by a large number of organizations, such as Yahoo! Inc., Twitter, Netflix, and Facebook, in their distributed application platforms as a coordination service. We will discuss more about how ZooKeeper is used in the real world in Chapter 7, ZooKeeper in Action.

A detailed listing of organizations and projects using ZooKeeper as a coordination service is available at https://cwiki.apache.org/confluence/display/ZOOKEEPER/PoweredBy.

**Getting hands-on with Apache ZooKeeper**

In this section, we will show you how to download and install Apache ZooKeeper so that we can start using ZooKeeper straightaway. This section is aimed at developers wanting to get their hands dirty using ZooKeeper for their distributed applications' needs by giving detailed installation and usage instructions. We will start with a single node ZooKeeper installation by getting acquainted with the basic configuration, followed by learning the ZooKeeper shell. Finally, you will be taught how to to set up a multinode ZooKeeper cluster.

**Download and installation**

ZooKeeper is supported by a wide variety of platforms. GNU/Linux and Oracle Solaris are supported as development and production platforms for both server and client. Windows and Mac OS X are recommended only as development platforms for both server and client.

In this book, we will assume a GNU-based/Linux-based installation of Apache ZooKeeper for installation and other instructions.

ZooKeeper is implemented in Java and requires Java 6 or later versions to run. While Oracle's version of Java is recommended, OpenJDK should also work fine for the correct functioning of ZooKeeper and many of the code samples in this book.

ZooKeeper runs as a server ensemble known as a ZooKeeper ensemble. In a production cluster, three ZooKeeper servers is the minimum recommended size for an ensemble, and it is recommended that you run them on separate machines. However, you can learn and evaluate ZooKeeper by installing it on a single machine in standalone mode.

A recent stable ZooKeeper distribution can be downloaded from one of the Apache Download Mirrors (http://bit.ly/1xEl8hA). At the time of writing this book, release 3.4.6 was the latest stable version available.

**Downloading**

Let's download the stable version from one of the mirrors, say Georgia Tech's Apache download mirror (http://b.gatech.edu/1xE1xRb) in the following example:

```bash
$ ls -alh zookeeper-3.4.6.tar.gz
-rw-rw-r-- 1 saurav saurav 17M Feb 20  2014 zookeeper-3.4.6.tar.gz
```

**Installing**

Once we have downloaded the ZooKeeper tarball, installing and setting up a standalone ZooKeeper node is pretty simple and straightforward. Let's extract the compressed tar archive into `/usr/share`:

```bash
$ tar -C /usr/share -zxf zookeeper-3.4.6.tar.gz
$ cd /usr/share/zookeeper-3.4.6/
$ ls
```

The location where the ZooKeeper archive is extracted in our case, `/usr/share/zookeeper-3.4.6`, can be exported as `ZK_HOME` as follows:

```bash
$ export ZK_HOME=/usr/share/zookeeper-3.4.6
```
Configuration

Once we have extracted the tarball, the next thing is to configure ZooKeeper. The conf folder holds the configuration files for ZooKeeper. ZooKeeper needs a configuration file called zoo.cfg in the conf folder inside the extracted ZooKeeper folder. There is a sample configuration file that contains some of the configuration parameters for reference.

Let's create our configuration file with the following minimal parameters and save it in the conf directory:

```
$ cat conf/zoo.cfg

    tickTime=2000
    dataDir=/var/lib/zookeeper
    clientPort=2181
```

The configuration parameters' meanings are explained here:

- **tickTime**: This is measured in milliseconds; it is used for session registration and to do regular heartbeats by clients with the ZooKeeper service. The minimum session timeout will be twice the `tickTime` parameter.
- **dataDir**: This is the location to store the in-memory state of ZooKeeper; it includes database snapshots and the transaction log of updates to the database. Extracting the ZooKeeper archive won't create this directory, so if this directory doesn't exist in the system, you will need to create it and set writable permission to it.
- **clientPort**: This is the port that listens for client connections, so it is where the ZooKeeper clients will initiate a connection. The client port can be set to any number, and different servers can be configured to listen on different ports. The default is 2181.

We will learn about the various storage, network, and cluster configuration parameters of ZooKeeper in more detail in Chapter 5, *Administering Apache ZooKeeper*.

As mentioned previously, ZooKeeper needs a Java Runtime Environment for it to work.

> It is assumed that readers already have a working version of Java running in their system where ZooKeeper is being installed and configured.
To see if Java is installed on your system, run the following command:

```
$ java -version
```

If Java is installed and its path is configured properly, then depending on the version and release of Java (Oracle or OpenJDK), the preceding command will show the version of Java and Java Runtime installed on your system. For example, in my system, I have Java 1.7.0.67 installed. So, using the preceding command, this will return the following output in my system:

```
$ java -version
java version "1.7.0_67"
Java(TM) SE Runtime Environment (build 1.7.0_67-b01)
Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)
```

ZooKeeper needs the `JAVA_HOME` environment variable to be set correctly. To see if this is set in your system, run the following command:

```
$ echo $JAVA_HOME
```

On my system, `JAVA_HOME` is set to `/usr/java/latest`, and hence, I got the following output:

```
$ echo $JAVA_HOME
/usr/java/latest
```

**Starting the ZooKeeper server**

Now, considering that Java is installed and working properly, let's go ahead and start the ZooKeeper server. All ZooKeeper administration scripts to start/stop the server and invoke the ZooKeeper command shell are shipped along with the archive in the `bin` folder with the following code:

```
$ pwd
/usr/share/zookeeper-3.4.6/bin

$ ls
README.txt zkCleanup.sh zkCli.cmd zkCli.sh zkEnv.cmd zkEnv.sh
zkServer.cmd zkServer.sh
```

The scripts with the `.sh` extension are for Unix platforms (GNU/Linux, Mac OS X, and so on), and the scripts with the `.cmd` extension are for Microsoft Windows operating systems.
To start the ZooKeeper server in a GNU/Linux system, you need to execute the `zkServer.sh` script as follows. This script gives options to start, stop, restart, and see the status of the ZooKeeper server:

```
$ ./zkServer.sh
```

JMX enabled by default

Using config: /usr/share/zookeeper-3.4.6/bin/../conf/zoo.cfg

Usage: ./zkServer.sh

{start|start-foreground|stop|restart|status|upgrade|print-cmd}

To avoid going to the ZooKeeper install directory to run these scripts, you can include it in your `PATH` variable as follows:

```
export PATH=$PATH:/usr/share/zookeeper-3.4.6/bin
```

Executing `zkServer.sh` with the `start` argument will start the ZooKeeper server. A successful start of the server will show the following output:

```
$ zkServer.sh start
```

JMX enabled by default

Using config: /usr/share/zookeeper-3.4.6/bin/../conf/zoo.cfg

Starting zookeeper ... STARTED

To verify that the ZooKeeper server has started, you can use the following `ps` command:

```
$ ps -ef | grep zookeeper | grep -v grep | awk '{print $2}'
5511
```

If the `jps` command is installed on your system, you can verify the ZooKeeper server's status as follows:

```
$ which jps
jps is /usr/bin/jps
$ jps
5511 QuorumPeerMain
5565 Jps
```

The ZooKeeper process is listed as `QuorumPeerMain`. In this case, as reported by `jps`, the ZooKeeper server is running with the 5511 process ID that matches the one reported by the `ps` command.
The ZooKeeper server's status can be checked with the `zkServer.sh` script as follows:

$ zkServer.sh status
JMX enabled by default
Using config: /usr/share/zookeeper-3.4.6/bin/../conf/zoo.cfg
Mode: standalone

To stop the server process, you can use the same script with the `stop` argument:

$ zkServer.sh stop
JMX enabled by default
Using config: /usr/share/zookeeper-3.4.6/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED

Checking the status of ZooKeeper when it has stopped or is not running will show the following result:

$ zkServer.sh status
JMX enabled by default
Using config: /usr/share/zookeeper-3.4.6/bin/../conf/zoo.cfg
Error contacting service. It is probably not running.

Once our ZooKeeper instance is running, the next thing to do is to connect to it. ZooKeeper ships with a default Java-based command-line shell to connect to a ZooKeeper instance. There is a C client as well, which we will discuss in a later section.

**Connecting to ZooKeeper with a Java-based shell**

To start the Java-based ZooKeeper command-line shell, we simply need to run `zkCli.sh` of the `ZK_HOME/bin` folder with the server IP and port as follows:

`${ZK_HOME}/bin/zkCli.sh -server zk_server:port`

In our case, we are running our ZooKeeper server on the same machine, so the ZooKeeper server will be `localhost`, or the loopback address will be `127.0.0.1`. The default port we configured was `2181`:

$ zkCli.sh -server localhost:2181
As we connect to the running ZooKeeper instance, we will see the output similar to the following one in the terminal (some output is omitted):

```
Connecting to localhost:2181
...............
...............
Welcome to ZooKeeper!
JLine support is enabled
...............
WATCHER::
WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0]
```

To see a listing of the commands supported by the ZooKeeper Java shell, you can run the `help` command in the shell prompt:

```
[zk: localhost:2181(CONNECTED) 0] help
ZooKeeper -server host:port cmd args
    connect host:port
    get path [watch]
    ls path [watch]
    set path data [version]
    rmr path
    delquota [-n|-b] path
    quit
    printwatches on|off
    create [-s] [-e] path data acl
    stat path [watch]
    close
    ls2 path [watch]
    history
    listquota path
    setAcl path acl
    getAcl path
    sync path
    redo cmdno
    addauth scheme auth
    delete path [version]
    setquota -n|-b val path
```
We can execute a few simple commands to get a feel of the command-line interface. Let's start by running the `ls` command, which, as in Unix, is used for listing:

```
[zk: localhost:2181(CONNECTED) 1] ls /
[zookeeper]
```

Now, the `ls` command returned a string called `zookeeper`, which is a znode in the ZooKeeper terminology. Note that we will get introduced to the ZooKeeper data model in the next chapter, Chapter 2, *Understanding the Inner Workings of Apache ZooKeeper*. We can create a znode through the ZooKeeper shell as follows:

To begin with, let's create a `HelloWorld` znode with empty data:

```
[zk: localhost:2181(CONNECTED) 2] create /HelloWorld ""
Created /HelloWorld
[zk: localhost:2181(CONNECTED) 3] ls /
[zookeeper, HelloWorld]
```

We can delete the znode created by issuing the `delete` command as follows:

```
[zk: localhost:2181(CONNECTED) 4] delete /HelloWorld
[zk: localhost:2181(CONNECTED) 5] ls /
[zookeeper]
```

The operations shown here will be clearer as we learn more about the ZooKeeper architecture, its data model, and namespace and internals in the subsequent chapters. We will look at setting up the C language-based command-line shell of the ZooKeeper distribution.

### Connecting to ZooKeeper with a C-based shell

ZooKeeper is shipped with a C language-based command-line shell. However, to use this shell, we need to build the C sources in `${ZK_HOME}/src/c`. A GNU/GCC compiler is required to build the sources. To build them, just run the following three commands in the preceding directory:

```
$ ./configure
$ make
$ make install
```

By default, this installs the C client libraries under `/usr/local/lib`. The C client libraries are built for both single-threaded as well as multithreaded libraries. The single-threaded library is suffixed with `_st`, while the multithreaded library is suffixed with `_mt`.
The C-based ZooKeeper shell uses these libraries for its execution. As such, after the preceding build procedure, two executables called cli_st and cli_mt are also generated in the current folder. These two binaries are the single-threaded and multithreaded command-line shells, respectively. When cli_mt is run, we get the following output:

```
$ cli_mt
USAGE cli_mt zookeeper_host_list
[clientid_file|cmd:(ls|ls2|create|od|...)]
Version: ZooKeeper cli (c client) version 3.4.6
```

To connect to our ZooKeeper server instance with this C-based shell, execute the following command in your terminal:

```
$ cli_mt localhost:2181
Watcher SESSION_EVENT state = CONNECTED_STATE
Got a new session id: 0x148b540cc4d0004
```

The C-based ZooKeeper shell also supports multiple commands, such as the Java version. Let's see the available commands under this shell by executing the help command:

```
help
  create [+[e|s]] <path>
  delete <path>
  set <path> <data>
  get <path>
  ls <path>
  ls2 <path>
  sync <path>
  exists <path>
  wexists <path>
  myid
  verbose
  addauth <id> <scheme>
  quit
prefix the command with the character 'a' to run the command asynchronously.run the 'verbose' command to toggle verbose logging.
i.e. 'aget /foo' to get /foo asynchronously
```
We can issue the same set of commands to list the znodes, create a znode, and finally delete it:

```
ls /
time = 3 msec
/: rc = 0
zookeeper

create /HelloWorld
Creating [/HelloWorld] node
Watcher CHILD_EVENT state = CONNECTED_STATE for path /
[/HelloWorld]: rc = 0
name = /HelloWorld

ls /
time = 3 msec
/: rc = 0
zookeeper
HelloWorld

delete /HelloWorld
Watcher CHILD_EVENT state = CONNECTED_STATE for path /
```

The format of the C-based ZooKeeper shell output displays the amount of time spent during the command execution as well as the return code (rc). A return code equal to zero denotes successful execution of the command.

The C static and shared libraries that we built earlier and installed in /usr/local/lib are required for ZooKeeper programming for distributed applications written in the C programming language. The Perl and Python client bindings shipped with the ZooKeeper distribution are also based on this C-based interface.
Setting up a multinode ZooKeeper cluster

So far, we have set up a ZooKeeper server instance in standalone mode. A standalone instance is a potential single point of failure. If the ZooKeeper server fails, the whole application that was using the instance for its distributed coordination will fail and stop functioning. Hence, running ZooKeeper in standalone mode is not recommended for production, although for development and evaluation purposes, it serves the need.

In a production environment, ZooKeeper should be run on multiple servers in a replicated mode, also called a ZooKeeper ensemble. The minimum recommended number of servers is three, and five is the most common in a production environment. The replicated group of servers in the same application domain is called a quorum. In this mode, the ZooKeeper server instance runs on multiple different machines, and all servers in the quorum have copies of the same configuration file. In a quorum, ZooKeeper instances run in a leader/follower format. One of the instances is elected the leader, and others become followers. If the leader fails, a new leader election happens, and another running instance is made the leader. However, these intricacies are fully hidden from applications using ZooKeeper and from developers.

The ZooKeeper configuration file for a multinode mode is similar to the one we used for a single instance mode, except for a few entries. An example configuration file is shown here:

```ini
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
initLimit=5
syncLimit=2
server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888
```

The two configuration parameters are also explained here:

- **initLimit**: This parameter is the timeout, specified in number of ticks, for a follower to initially connect to a leader
- **syncLimit**: This is the timeout, specified in number of ticks, for a follower to sync with a leader
Both of these timeouts are specified in the unit of time called `tickTime`. Thus, in our example, the timeout for `initLimit` is 5 ticks at 2000 milliseconds a tick, or 10 seconds.

The other three entries in the preceding example in the `server.id=host:port:port` format are the list of servers that constitute the quorum. The `id` identifier is a number that is used for the server with a hostname in the quorum. In our example configuration, the `zoo1` quorum member host is assigned an identifier 1.

The identifier is needed to be specified in a file called `myid` in the data directory of that server. It's important that the `myid` file should consist of a single line that contains only the text (ASCII) of that server's ID. The id must be unique within the ensemble and should have a value between 1 and 255.

Again, we have the two port numbers after each server hostname: 2888 and 3888. They are explained here:

- The first port, 2888, is mostly used for peer-to-peer communication in the quorum, such as to connect followers to leaders. A follower opens a TCP connection to the leader using this port.
- The second port, 3888, is used for leader election, in case a new leader arises in the quorum. As all communication happens over TCP, a second port is required to respond to leader election inside the quorum.

**Starting the server instances**

After setting up the configuration file for each of the servers in the quorum, we need to start the ZooKeeper server instances. The procedure is the same as for standalone mode. We have to connect to each of the machines and execute the following command:

```bash
${ZK_HOME}/bin/zkServer.sh start
```

Once the instances are started successfully, we will execute the following command on each of the machines to check the instance states:

```bash
${ZK_HOME}/bin/zkServer.sh status
```

For example, take a look at the next quorum:

```
[zoo1] # ${ZK_HOME}/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/share/zookeeper-3.4.6/bin/..../conf/zoo.cfg
Mode: follower
```

```
[zoo2] # ${ZK_HOME}/bin/zkServer.sh status
```
JMX enabled by default
Using config: /usr/share/zookeeper-3.4.6/bin/../conf/zoo.cfg
Mode: leader

[zoo3] # ${ZK_HOME}/bin/zkServer.sh status
JMX enabled by default
Using config: /usr/share/zookeeper-3.4.6/bin/../conf/zoo.cfg
Mode: follower

As seen in the preceding example, zoo2 is made the leader of the quorum, while
zoo1 and zoo3 are the followers. Connecting to the ZooKeeper quorum through the
command-line shell is also the same as in standalone mode, except that we should
now specify a connection string in the host1:port2, host2:port2 ... format to the server
argument of ${ZK_HOME}/bin/zkCli.sh:

$ zkCli.sh -server zoo1:2181,zoo2:2181,zoo3:2181
Connecting to zoo1:2181, zoo2:2181, zoo3:2181

Welcome to ZooKeeper!

[zk: zoo1:2181,zoo2:2181,zoo3:2181 (CONNECTED) 0]

Once the ZooKeeper cluster is up and running, there are ways to monitor it using
Java Management Extensions (JMX) and by sending some commands over the client
port, also known as the Four Letter Words. We will discuss ZooKeeper monitoring
in more detail in Chapter 5, Administering Apache ZooKeeper.

Running multiple node modes for ZooKeeper

It is also possible to run ZooKeeper in multiple node modes on a single machine.
This is useful for testing purposes. To run multinode modes on the same machine,
we need to tweak the configuration a bit; for example, we can set the server name as
localhost and specify the unique quorum and leader election ports.

Let's use the following configuration file to set up a multinode ZooKeeper cluster
using a single machine:

```
tickTime=2000
initLimit=5
syncLimit=2
dataDir=/var/lib/zookeeper
clientPort=2181
server.1=localhost:2666:3666
```

[24]
As already explained in the previous section, each entry of the server \( X \) specifies the address and port numbers used by the \( X \) ZooKeeper server. The first field is the hostname or IP address of server \( X \). The second and third fields are the TCP port numbers used for quorum communication and leader election, respectively. As we are starting three ZooKeeper server instances on the same machine, we need to use different port numbers for each of the server entries.

Second, as we are running more than one ZooKeeper server process on the same machine, we need to have different client ports for each of the instances.

Last but not least, we have to customize the `dataDir` parameter as well for each of the instances we are running.

Putting all these together, for a three-instance ZooKeeper cluster, we will create three different configuration files. We will call these `zoo1.cfg`, `zoo2.cfg`, and `zoo3.cfg` and store them in the `conf` folder of `\${ZK_HOME}`. We will create three different data folders for the instances, say `zoo1`, `zoo2`, and `zoo3`, in `/var/lib/zookeeper`. Thus, the three configuration files are shown next.

Here, you will see the configuration file for the first instance:

```plaintext
tickTime=2000
initLimit=5
syncLimit=2
dataDir=/var/lib/zookeeper/zoo1
clientPort=2181
server.1=localhost:2666:3666
server.2=localhost:2667:3667
server.3=localhost:2668:3668
```

The second instance is shown here:

```plaintext
tickTime=2000
initLimit=5
syncLimit=2
dataDir=/var/lib/zookeeper/zoo2
clientPort=2182
server.1=localhost:2666:3666
server.2=localhost:2667:3667
server.3=localhost:2668:3668
```
The third and final instance is then shown here:

```plaintext
tickTime=2000
initLimit=5
syncLimit=2
dataDir=/var/lib/zookeeper/zoo3
clientPort=2183
server.1=localhost:2666:3666
server.2=localhost:2667:3667
server.3=localhost:2668:3668
```

We also need to fix the server ID parameter correctly in the `myid` file for each instance. This can be done using the following three commands:

```plaintext
$ echo 1 > /var/lib/zookeeper/zoo1/myid
$ echo 2 > /var/lib/zookeeper/zoo2/myid
$ echo 3 > /var/lib/zookeeper/zoo3/myid
```

Now, we are all set to start the ZooKeeper instances. Let's start the instances as follows:

```plaintext
$ ${ZK_HOME}/bin/zkServer.sh start ${ZK_HOME}/conf/zoo1.cfg
$ ${ZK_HOME}/bin/zkServer.sh start ${ZK_HOME}/conf/zoo2.cfg
$ ${ZK_HOME}/bin/zkServer.sh start ${ZK_HOME}/conf/zoo3.cfg
```

Once all the instances start, we can use the `zkCli.sh` script to connect to the multinode ZooKeeper cluster, like we did earlier:

```plaintext
$ ${ZK_HOME}/bin/zkCli.sh -server localhost:2181, localhost:2182, localhost:2183
```

Voila! We have a three-node ZooKeeper cluster running on the same machine!
Summary

In this chapter, you learned the general definition of a distributed system and why coordination among entities that constitute a large system is hard and a very important problem to be solved. You learned how Apache ZooKeeper is a great tool for distributed system designer and developers to solve coordination problems. This chapter provided details on installing and configuring a ZooKeeper in various modes, such as standalone, clustered, and also talked about how to connect to a ZooKeeper service from the command line with the ZooKeeper shell.

In the next chapter, you will learn about the internals and architecture of Apache ZooKeeper. You will learn in detail about the ZooKeeper data model and the API interfaces exposed by the ZooKeeper service. The concepts introduced in the next chapter will help you master the design semantics of ZooKeeper and equip readers with confidence in using ZooKeeper in their distributed applications.
Where to buy this book

You can buy Apache ZooKeeper Essentials from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers. Click here for ordering and shipping details.