# Zookeeper 安装

在安装ZooKeeper之前，请确保你的系统是在以下任一操作系统上运行：

* **任意Linux OS** - 支持开发和部署。适合演示应用程序。
* **Windows OS** - 仅支持开发。
* **Mac OS**- 仅支持开发。

ZooKeeper服务器是用Java创建的，它在JVM上运行。你需要使用JDK 6或更高版本。

现在，按照以下步骤在你的机器上安装ZooKeeper框架。

**步骤1：验证Java安装**

相信你已经在系统上安装了Java环境。现在只需使用以下命令验证它。

$ java -version

如果你在机器上安装了Java，那么可以看到已安装的Java的版本。否则，请按照以下简单步骤安装最新版本的Java。

**步骤1.1：下载JDK**

通过访问链接下载最新版本的JDK，并下载最新版本的[Java](http://www.oracle.com/technetwork/java/javase/downloads/index.html" \t "_blank)。

最新版本（在编写本教程时）是JDK 8u 60，文件是“jdk-8u60-linuxx64.tar.gz"。请在你的机器上下载该文件。

**步骤1.2：提取文件**

通常，文件会下载到**download**文件夹中。验证并使用以下命令提取tar设置。

$ cd /go/to/download/path

$ tar -zxf jdk-8u60-linux-x64.gz

**步骤1.3：移动到opt目录**

要使Java对所有用户可用，请将提取的Java内容移动到“/usr/local/java"文件夹。

$ su

password: (type password of root user)

$ mkdir /opt/jdk

$ mv jdk-1.8.0\_60 /opt/jdk/

**步骤1.4：设置路径**

要设置路径和JAVA\_HOME变量，请将以下命令添加到〜/.bashrc文件中。

export JAVA\_HOME = /usr/jdk/jdk-1.8.0\_60

export PATH=$PATH:$JAVA\_HOME/bin

现在，将所有更改应用到当前运行的系统中。

$ source ~/.bashrc

**步骤1.5：Java替代**

使用以下命令更改Java替代项。

update-alternatives --install /usr/bin/java java /opt/jdk/jdk1.8.0\_60/bin/java 100

**步骤1.6**

使用步骤1中说明的验证命令**(java -version)**验证Java安装。

**步骤2：ZooKeeper框架安装**

**步骤2.1：下载ZooKeeper**

要在你的计算机上安装ZooKeeper框架，请访问以下链接并下载最新版本的ZooKeeper。[http://zookeeper.apache.org/releases.html](http://zookeeper.apache.org/releases.html" \t "_blank)

到目前为止，最新版本的ZooKeeper是3.4.6(ZooKeeper-3.4.6.tar.gz)。

**步骤2.2：提取tar文件**

使用以下命令提取tar文件

$ cd opt/

$ tar -zxf zookeeper-3.4.6.tar.gz

$ cd zookeeper-3.4.6

$ mkdir data

**步骤2.3：创建配置文件**

使用命令**vi conf/zoo.cfg**和所有以下参数设置为起点，打开名为**conf/zoo.cfg**的配置文件。

$ vi conf/zoo.cfg

tickTime = 2000

dataDir = /path/to/zookeeper/data

clientPort = 2181

initLimit = 5

syncLimit = 2

一旦成功保存配置文件，再次返回终端。你现在可以启动zookeeper服务器。

**步骤2.4：启动ZooKeeper服务器**

执行以下命令

$ bin/zkServer.sh start

执行此命令后，你将收到以下响应

$ JMX enabled by default

$ Using config: /Users/../zookeeper-3.4.6/bin/../conf/zoo.cfg

$ Starting zookeeper ... STARTED

**步骤2.5：启动CLI**

键入以下命令

$ bin/zkCli.sh

键入上述命令后，将连接到ZooKeeper服务器，你应该得到以下响应。

Connecting to localhost:2181

................

................

................

Welcome to ZooKeeper!

................

................

WATCHER::

WatchedEvent state:SyncConnected type: None path:null

[zk: localhost:2181(CONNECTED) 0]

**停止ZooKeeper服务器**

连接服务器并执行所有操作后，可以使用以下命令停止zookeeper服务器。

$ bin/zkServer.sh stop

# Zookeeper CLI

ZooKeeper命令行界面（CLI）用于与ZooKeeper集合进行交互以进行开发。它有助于调试和解决不同的选项。

要执行ZooKeeper CLI操作，首先打开ZooKeeper服务器（“bin/zkServer.sh start”），然后打开ZooKeeper客户端（“bin/zkCli.sh”）。一旦客户端启动，你可以执行以下操作：  
1、创建znode  
2、获取数据  
3、监视znode的变化  
4、设置数据  
5、创建znode的子节点  
6、列出znode的子节点  
7、检查状态  
8、移除/删除znode  
现在让我们用一个例子逐个了解上面的命令。

## 创建Znodes

用给定的路径创建一个znode。flag参数指定创建的znode是临时的，持久的还是顺序的。默认情况下，所有znode都是持久的。  
当会话过期或客户端断开连接时，**临时节点**（flag：-e）将被自动删除。  
**顺序节点**保证znode路径将是唯一的。  
ZooKeeper集合将向znode路径填充10位序列号。例如，znode路径 /myapp 将转换为/myapp0000000001，下一个序列号将为/myapp0000000002。如果没有指定flag，则znode被认为是持久的。

### ****语法****

create /path /data

### 示例

create /FirstZnode “Myfirstzookeeper-app"

### 输出

[zk: localhost:2181(CONNECTED) 0] create /FirstZnode “Myfirstzookeeper-app"

Created /FirstZnode

要创建**顺序节点**，请添加flag：**-s**，如下所示。

### 语法

create -s /path /data

### 示例

create -s /FirstZnode second-data

### 输出

[zk: localhost:2181(CONNECTED) 2] create -s /FirstZnode “second-data"

Created /FirstZnode0000000023

要创建**临时节点**，请添加flag：**-e**，如下所示。

### 语法

create -e /path /data

### 示例

create -e /SecondZnode “Ephemeral-data"

### 输出

[zk: localhost:2181(CONNECTED) 2] create -e /SecondZnode “Ephemeral-data"

Created /SecondZnode

记住当客户端断开连接时，临时节点将被删除。你可以通过退出ZooKeeper CLI，然后重新打开CLI来尝试。

## 获取数据

它返回znode的关联数据和指定znode的元数据。你将获得信息，例如上次修改数据的时间，修改的位置以及数据的相关信息。此CLI还用于分配监视器以显示数据相关的通知。

### 语法

get /path

### 示例

get /FirstZnode

### 输出

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

“Myfirstzookeeper-app"

cZxid = 0x7f

ctime = Tue Sep 29 16:15:47 IST 2015

mZxid = 0x7f

mtime = Tue Sep 29 16:15:47 IST 2015

pZxid = 0x7f

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 22

numChildren = 0

要访问顺序节点，必须输入znode的完整路径。

### 示例

get /FirstZnode0000000023

### 输出

[zk: localhost:2181(CONNECTED) 1] get /FirstZnode0000000023

“Second-data"

cZxid = 0x80

ctime = Tue Sep 29 16:25:47 IST 2015

mZxid = 0x80

mtime = Tue Sep 29 16:25:47 IST 2015

pZxid = 0x80

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 13

numChildren = 0

## Watch（监视）

当指定的znode或znode的子数据更改时，监视器会显示通知。你只能在**get**命令中设置**watch**。

### 语法

get /path [watch] 1

### 示例

get /FirstZnode 1

### 输出

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

“Myfirstzookeeper-app"

cZxid = 0x7f

ctime = Tue Sep 29 16:15:47 IST 2015

mZxid = 0x7f

mtime = Tue Sep 29 16:15:47 IST 2015

pZxid = 0x7f

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 22

numChildren = 0

输出类似于普通的**get**命令，但它会等待后台等待znode更改。<从这里开始>

## 设置数据

设置指定znode的数据。完成此设置操作后，你可以使用**get**CLI命令检查数据。

### 语法

set /path /data

### 示例

set /SecondZnode Data-updated

### 输出

[zk: localhost:2181(CONNECTED) 1] get /SecondZnode “Data-updated"

cZxid = 0x82

ctime = Tue Sep 29 16:29:50 IST 2015

mZxid = 0x83

mtime = Tue Sep 29 16:29:50 IST 2015

pZxid = 0x82

cversion = 0

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x15018b47db00000

dataLength = 14

numChildren = 0

如果你在**get**命令中分配了**watch**选项（如上一个命令），则输出将类似如下所示。

### 输出

[zk: localhost:2181(CONNECTED) 1] get /FirstZnode “Mysecondzookeeper-app"

WATCHER: :

WatchedEvent state:SyncConnected type:NodeDataChanged path:/FirstZnode

cZxid = 0x7f

ctime = Tue Sep 29 16:15:47 IST 2015

mZxid = 0x84

mtime = Tue Sep 29 17:14:47 IST 2015

pZxid = 0x7f

cversion = 0

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 23

numChildren = 0

## 创建子项/子节点

创建子节点类似于创建新的znode。唯一的区别是，子znode的路径也将具有父路径。

### 语法

create /parent/path/subnode/path /data

### 示例

create /FirstZnode/Child1 firstchildren

### 输出

[zk: localhost:2181(CONNECTED) 16] create /FirstZnode/Child1 “firstchildren"

created /FirstZnode/Child1

[zk: localhost:2181(CONNECTED) 17] create /FirstZnode/Child2 “secondchildren"

created /FirstZnode/Child2

## 列出子项

此命令用于列出和显示znode的子项。

### 语法

ls /path

### 示例

ls /MyFirstZnode

### 输出

[zk: localhost:2181(CONNECTED) 2] ls /MyFirstZnode

[mysecondsubnode, myfirstsubnode]

## 检查状态

状态描述指定的znode的元数据。它包含时间戳，版本号，ACL，数据长度和子znode等细项。

### 语法

stat /path

### 示例

stat /FirstZnode

### 输出

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

cZxid = 0x7f

ctime = Tue Sep 29 16:15:47 IST 2015

mZxid = 0x7f

mtime = Tue Sep 29 17:14:24 IST 2015

pZxid = 0x7f

cversion = 0

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 23

numChildren = 0

## 移除Znode

移除指定的znode并递归其所有子节点。只有在这样的znode可用的情况下才会发生。

### 语法

rmr /path

### 示例

rmr /FirstZnode

### 输出

[zk: localhost:2181(CONNECTED) 10] rmr /FirstZnode

[zk: localhost:2181(CONNECTED) 11] get /FirstZnode

Node does not exist: /FirstZnode

删除**(delete/path)**命令类似于**remove**命令，除了它只适用于没有子节点的znode。

**Zookeeper API**

ZooKeeper有一个绑定Java和C的官方API。Zookeeper社区为大多数语言（.NET，python等）提供非官方API。使用ZooKeeper API，应用程序可以连接，交互，操作数据，协调，最后断开与ZooKeeper集合的连接。

ZooKeeper API具有丰富的功能，以简单和安全的方式获得ZooKeeper集合的所有功能。ZooKeeper API提供同步和异步方法。

ZooKeeper集合和ZooKeeper API在各个方面都完全相辅相成，对开发人员有很大的帮助。让我们在本章讨论Java绑定。

**ZooKeeper API的基础知识**

与ZooKeeper集合进行交互的应用程序称为**ZooKeeper客户端**或简称**客户端**。

Znode是ZooKeeper集合的核心组件，ZooKeeper API提供了一小组方法使用ZooKeeper集合来操纵znode的所有细节。

客户端应该遵循以步骤，与ZooKeeper集合进行清晰和干净的交互。

* 连接到ZooKeeper集合。ZooKeeper集合为客户端分配会话ID。
* 定期向服务器发送心跳。否则，ZooKeeper集合将过期会话ID，客户端需要重新连接。
* 只要会话ID处于活动状态，就可以获取/设置znode。
* 所有任务完成后，断开与ZooKeeper集合的连接。如果客户端长时间不活动，则ZooKeeper集合将自动断开客户端。

**Java绑定**

让我们来了解本章中最重要的一组ZooKeeper API。ZooKeeper API的核心部分是**ZooKeeper类**。它提供了在其构造函数中连接ZooKeeper集合的选项，并具有以下方法：

* **connect**- 连接到ZooKeeper集合
* **create**- 创建znode
* **exists**- 检查znode是否存在及其信息
* **getData**- 从特定的znode获取数据
* **setData**- 在特定的znode中设置数据
* **getChildren**- 获取特定znode中的所有子节点
* **delete**- 删除特定的znode及其所有子项
* **close**- 关闭连接

**连接到ZooKeeper集合**

ZooKeeper类通过其构造函数提供connect功能。构造函数的签名如下 :

ZooKeeper(String connectionString, int sessionTimeout, Watcher watcher)

* **connectionString**- ZooKeeper集合主机。
* **sessionTimeout**- 会话超时（以毫秒为单位）。
* **watcher**- 实现“监视器”界面的对象。ZooKeeper集合通过监视器对象返回连接状态。

让我们创建一个新的帮助类**ZooKeeperConnection**，并添加一个方法**connect**。 **connect**方法创建一个ZooKeeper对象，连接到ZooKeeper集合，然后返回对象。

这里**CountDownLatch**用于停止（等待）主进程，直到客户端与ZooKeeper集合连接。

ZooKeeper集合通过监视器回调来回复连接状态。一旦客户端与ZooKeeper集合连接，监视器回调就会被调用，并且监视器回调函数调用**CountDownLatch**的**countDown**方法来释放锁，在主进程中**await**。

以下是与ZooKeeper集合连接的完整代码。

**编码：ZooKeeperConnection.java**

// import java classes

import java.io.IOException;

import java.util.concurrent.CountDownLatch;

// import zookeeper classes

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

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

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.AsyncCallback.StatCallback;

import org.apache.zookeeper.KeeperException.Code;

import org.apache.zookeeper.data.Stat;

public class ZooKeeperConnection {

// declare zookeeper instance to access ZooKeeper ensemble

private ZooKeeper zoo;

final CountDownLatch connectedSignal = new CountDownLatch(1);

// Method to connect zookeeper ensemble.

public ZooKeeper connect(String host) throws IOException,InterruptedException {

zoo = new ZooKeeper(host,5000,new Watcher() {

public void process(WatchedEvent we) {

if (we.getState() == KeeperState.SyncConnected) {

connectedSignal.countDown();

}

}

});

connectedSignal.await();

return zoo;

}

// Method to disconnect from zookeeper server

public void close() throws InterruptedException {

zoo.close();

}

}

保存上面的代码，它将在下一节中用于连接ZooKeeper集合。

**创建Znode**

ZooKeeper类提供了在ZooKeeper集合中创建一个新的znode的**create**方法。 **create**方法的签名如下：

create(String path, byte[] data, List<ACL> acl, CreateMode createMode)

* **path** - Znode路径。例如，/myapp1，/myapp2，/myapp1/mydata1，myapp2/mydata1/myanothersubdata
* **data**- 要存储在指定znode路径中的数据
* **acl**- 要创建的节点的访问控制列表。ZooKeeper API提供了一个静态接口**ZooDefs.Ids**来获取一些基本的acl列表。例如，ZooDefs.Ids.OPEN\_ACL\_UNSAFE返回打开znode的acl列表。
* **createMode**- 节点的类型，即临时，顺序或两者。这是一个**枚举**。

让我们创建一个新的Java应用程序来检查ZooKeeper API的**create**功能。创建文件**ZKCreate.java**。在main方法中，创建一个类型为**ZooKeeperConnection**的对象，并调用**connect**方法连接到ZooKeeper集合。

connect方法将返回ZooKeeper对象**zk**。现在，请使用自定义**path**和**data**调用**zk**对象的**create**方法。

创建znode的完整程序代码如下：

**编码：ZKCreate.java**

import java.io.IOException;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

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

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.CreateMode;

import org.apache.zookeeper.ZooDefs;

public class ZKCreate {

// create static instance for zookeeper class.

private static ZooKeeper zk;

// create static instance for ZooKeeperConnection class.

private static ZooKeeperConnection conn;

// Method to create znode in zookeeper ensemble

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

KeeperException,InterruptedException {

zk.create(path, data, ZooDefs.Ids.OPEN\_ACL\_UNSAFE,

CreateMode.PERSISTENT);

}

public static void main(String[] args) {

// znode path

String path = "/MyFirstZnode"; // Assign path to znode

// data in byte array

byte[] data = "My first zookeeper app".getBytes(); // Declare data

try {

conn = new ZooKeeperConnection();

zk = conn.connect("localhost");

create(path, data); // Create the data to the specified path

conn.close();

} catch (Exception e) {

System.out.println(e.getMessage()); //Catch error message

}

}

}

一旦编译和执行应用程序，将在ZooKeeper集合中创建具有指定数据的znode。你可以使用ZooKeeper CLI **zkCli.sh**进行检查。

cd /path/to/zookeeper

bin/zkCli.sh

>>> get /MyFirstZnode

**Exists - 检查Znode的存在**

ZooKeeper类提供了**exists**方法来检查znode的存在。如果指定的znode存在，则返回一个znode的元数据。**exists**方法的签名如下：

exists(String path, boolean watcher)

* **path**- Znode路径
* **watcher**- 布尔值，用于指定是否监视指定的znode

让我们创建一个新的Java应用程序来检查ZooKeeper API的“exists”功能。创建文件“ZKExists.java”。在main方法中，使用“ZooKeeperConnection”对象创建ZooKeeper对象“zk”。然后，使用自定义“path”调用“zk”对象的“exists”方法。完整的列表如下：

**编码：ZKExists.java**

import java.io.IOException;

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

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

import org.apache.zookeeper.data.Stat;

public class ZKExists {

private static ZooKeeper zk;

private static ZooKeeperConnection conn;

// Method to check existence of znode and its status, if znode is available.

public static Stat znode\_exists(String path) throws

KeeperException,InterruptedException {

return zk.exists(path, true);

}

public static void main(String[] args) throws InterruptedException,KeeperException {

String path = "/MyFirstZnode"; // Assign znode to the specified path

try {

conn = new ZooKeeperConnection();

zk = conn.connect("localhost");

Stat stat = znode\_exists(path); // Stat checks the path of the znode

if(stat != null) {

System.out.println("Node exists and the node version is " +

stat.getVersion());

} else {

System.out.println("Node does not exists");

}

} catch(Exception e) {

System.out.println(e.getMessage()); // Catches error messages

}

}

}

一旦编译和执行应用程序，你将获得以下输出。

Node exists and the node version is 1.

**getData方法**

ZooKeeper类提供**getData**方法来获取附加在指定znode中的数据及其状态。 **getData**方法的签名如下：

getData(String path, Watcher watcher, Stat stat)

* **path**- Znode路径。
* **watcher**- 监视器类型的回调函数。当指定的znode的数据改变时，ZooKeeper集合将通过监视器回调进行通知。这是一次性通知。
* **stat**- 返回znode的元数据。

让我们创建一个新的Java应用程序来了解ZooKeeper API的**getData**功能。创建文件**ZKGetData.java**。在main方法中，使用**ZooKeeperConnection**对象创建一个ZooKeeper对象**zk**。然后，使用自定义路径调用zk对象的**getData**方法。

下面是从指定节点获取数据的完整程序代码：

**编码：ZKGetData.java**

import java.io.IOException;

import java.util.concurrent.CountDownLatch;

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

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

import org.apache.zookeeper.data.Stat;

public class ZKGetData {

private static ZooKeeper zk;

private static ZooKeeperConnection conn;

public static Stat znode\_exists(String path) throws

KeeperException,InterruptedException {

return zk.exists(path,true);

}

public static void main(String[] args) throws InterruptedException, KeeperException {

String path = "/MyFirstZnode";

final CountDownLatch connectedSignal = new CountDownLatch(1);

try {

conn = new ZooKeeperConnection();

zk = conn.connect("localhost");

Stat stat = znode\_exists(path);

if(stat != null) {

byte[] b = zk.getData(path, new Watcher() {

public void process(WatchedEvent we) {

if (we.getType() == Event.EventType.None) {

switch(we.getState()) {

case Expired:

connectedSignal.countDown();

break;

}

} else {

String path = "/MyFirstZnode";

try {

byte[] bn = zk.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);

connectedSignal.await();

} else {

System.out.println("Node does not exists");

}

} catch(Exception e) {

System.out.println(e.getMessage());

}

}

}

一旦编译和执行应用程序，你将获得以下输出

My first zookeeper app

应用程序将等待ZooKeeper集合的进一步通知。使用ZooKeeper CLI **zkCli.sh**更改指定znode的数据。

cd /path/to/zookeeper

bin/zkCli.sh

>>> set /MyFirstZnode Hello

现在，应用程序将打印以下输出并退出。

Hello

**setData方法**

ZooKeeper类提供**setData**方法来修改指定znode中附加的数据。 **setData**方法的签名如下：

setData(String path, byte[] data, int version)

* **path**- Znode路径
* **data**- 要存储在指定znode路径中的数据。
* **version**- znode的当前版本。每当数据更改时，ZooKeeper会更新znode的版本号。

现在让我们创建一个新的Java应用程序来了解ZooKeeper API的**setData**功能。创建文件**ZKSetData.java**。在main方法中，使用**ZooKeeperConnection**对象创建一个ZooKeeper对象**zk**。然后，使用指定的路径，新数据和节点版本调用**zk**对象的**setData**方法。

以下是修改附加在指定znode中的数据的完整程序代码。

**编码：ZKSetData.java**

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

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

import java.io.IOException;

public class ZKSetData {

private static ZooKeeper zk;

private static ZooKeeperConnection conn;

// Method to update the data in a znode. Similar to getData but without watcher.

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

KeeperException,InterruptedException {

zk.setData(path, data, zk.exists(path,true).getVersion());

}

public static void main(String[] args) throws InterruptedException,KeeperException {

String path= "/MyFirstZnode";

byte[] data = "Success".getBytes(); //Assign data which is to be updated.

try {

conn = new ZooKeeperConnection();

zk = conn.connect("localhost");

update(path, data); // Update znode data to the specified path

} catch(Exception e) {

System.out.println(e.getMessage());

}

}

}

编译并执行应用程序后，指定的znode的数据将被改变，并且可以使用ZooKeeper CLI**zkCli.sh**进行检查。

cd /path/to/zookeeper

bin/zkCli.sh

>>> get /MyFirstZnode

**getChildren方法**

ZooKeeper类提供**getChildren**方法来获取特定znode的所有子节点。 **getChildren**方法的签名如下：

getChildren(String path, Watcher watcher)

* **path**- Znode路径。
* **watcher**- 监视器类型的回调函数。当指定的znode被删除或znode下的子节点被创建/删除时，ZooKeeper集合将进行通知。这是一次性通知。

**编码：ZKGetChildren.java**

import java.io.IOException;

import java.util.\*;

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.KeeperException;

import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

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

import org.apache.zookeeper.data.Stat;

public class ZKGetChildren {

private static ZooKeeper zk;

private static ZooKeeperConnection conn;

// Method to check existence of znode and its status, if znode is available.

public static Stat znode\_exists(String path) throws

KeeperException,InterruptedException {

return zk.exists(path,true);

}

public static void main(String[] args) throws InterruptedException,KeeperException {

String path = "/MyFirstZnode"; // Assign path to the znode

try {

conn = new ZooKeeperConnection();

zk = conn.connect("localhost");

Stat stat = znode\_exists(path); // Stat checks the path

if(stat!= null) {

//“getChildren" method- get all the children of znode.It has two

args, path and watch

List <String> children = zk.getChildren(path, false);

for(int i = 0; i < children.size(); i++)

System.out.println(children.get(i)); //Print children's

} else {

System.out.println("Node does not exists");

}

} catch(Exception e) {

System.out.println(e.getMessage());

}

}

}

在运行程序之前，让我们使用ZooKeeper CLI **zkCli.sh**为**/MyFirstZnode**创建两个子节点。

cd /path/to/zookeeper

bin/zkCli.sh

>>> create /MyFirstZnode/myfirstsubnode Hi

>>> create /MyFirstZnode/mysecondsubmode Hi

现在，编译和运行程序将输出上面创建的znode。

myfirstsubnode

mysecondsubnode

**删除Znode**

ZooKeeper类提供了**delete**方法来删除指定的znode。 **delete**方法的签名如下：

delete(String path, int version)

* **path**- Znode路径。
* **version**- znode的当前版本。

让我们创建一个新的Java应用程序来了解ZooKeeper API的**delete**功能。创建文件**ZKDelete.java**。在main方法中，使用**ZooKeeperConnection**对象创建一个ZooKeeper对象**zk**。然后，使用指定的路径和版本号调用**zk**对象的**delete**方法。

删除znode的完整程序代码如下：

**编码：ZKDelete.java**

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.KeeperException;

public class ZKDelete {

private static ZooKeeper zk;

private static ZooKeeperConnection conn;

// Method to check existence of znode and its status, if znode is available.

public static void delete(String path) throws KeeperException,InterruptedException {

zk.delete(path,zk.exists(path,true).getVersion());

}

public static void main(String[] args) throws InterruptedException,KeeperException {

String path = "/MyFirstZnode"; //Assign path to the znode

try {

conn = new ZooKeeperConnection();

zk = conn.connect("localhost");

delete(path); //delete the node with the specified path

} catch(Exception e) {

System.out.println(e.getMessage()); // catches error messages

}

}

}