链滴

【ZooKeeper 系列】2. 用 Java 实现 ZooKeeper API 的调用

作者：MiracleHe

原文链接：https://ld246.com/article/1577775977290

来源网站：链滴

温馨提示：在这里我再次提个小要求，希望大家能习惯看**官方文档**，文档虽然是英文但用词都比较简，基本都能看懂文档表达的意思。**授之以鱼不如授之以渔**的道理相信大家都明白，也希望通过猿人谷这个ZooKeeper系列，让大家入门、到熟悉，举一反三后能精通ZooKeeper。

在前一篇我们介绍了ZooKeeper单机版、伪集群和集群环境搭建，通过命令行的方式做了节点的创建删除、更新、获取节点信息的测试。Zookeeper 的目的是为客户端构建复杂的协调功能提供简单、效的核心 API，这一篇我们用Java通过ZooKeeper提供的API接口来实现这些增删改查的功能。

1 简介

org.apache.zookeeper.Zookeeper是ZooKeeper客户端的主类，在官方文档（**该系列文章以v3.5.5主**，v3.6.6的API Docs还没有）中已明确说明（This is the main class of ZooKeeper client library.。

This is the main class of ZooKeeper client library. To use a ZooKeeper service, an application ust first instantiate an object of ZooKeeper class. All the iterations will be done by calling the methods of ZooKeeper class. The methods of this class are thread-safe unless otherwise note .

Once a connection to a server is established, a session ID is assigned to the client. The client w ll send heart beats to the server periodically to keep the session valid.

创建一个ZooKeeper的实例来使用org.apache.zookeeper.Zookeeper里的方法，官方文档已经指出有特别声明的话，ZooKeeper类里的方法是线程安全的。客户端连接到ZooKeeper服务的时候，会客户端分配一个会话ID（session ID），客户端与服务端会通过心跳来保持会话有效。

org.apache.zookeeper.Zookeeper里的方法非常多，就不一一列举了，只列几个增删改查的。

Method	Description
create(String path, byte[] data, List acl, CreateMode createMode)	Create a node with the given path.（创建指定路径的节点）
create(String path, byte[] data, List acl, CreateMode createMode, AsyncCallback.Create2Callback cb, Object ctx)	The asynchronous version of create.（异步形式创建）
create(String path, byte[] data, List acl, CreateMode createMode, Stat stat)	Create a node with the given path and returns the Stat of that node.（按指定路径创建节点并返回节点状态信息）
delete(String path, int version)	Delete the node with the given path.（删除指定路径的节点）
delete(String path, int version, AsyncCallback.VoidCallback cb, Object ctx)	The asynchronous version of delete.（异步删除指定路径的节点）
exists(String path, boolean watch)	Return the stat of the node of the given path.（返回指定路径的节点状态信息）
getChildren(String path, boolean watch)	Return the list of the children of the node of the given path.（返回指定路径的所有子节点状态信息）
getData(String path, boolean watch, Stat stat)	Return the data and the stat of the node of the given path.（返回指定路径的节点数据和状态信息）
setData(String path, byte[] data, int version)	Set the data for the node of the given path if such a node exists and the given version matches the version of the node (if the given version is -1, it matches any node's versions).（给指定路径和版本的节点设置新值，如版本为-1，即给所有版本设置值）

2 测试环境搭建

这里新建一个**Spring Boot**的项目来进行测试，新建Spring Boot项目的过程很简单，也不是这里的点，就不做介绍了。

项目里会需要额外引入两个包来进行测试：

```
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
            <version>3.5.5</version>
        </dependency>

        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.5.2</version>
        </dependency>
```

3 API测试

完整测试代码如下：

```
/**
 *  简单测试示例
 * @author 猿人谷
 * @date 2019/12/16
 */
public class ZooKeeperDemo {

    private static final Logger LOGGER = LoggerFactory.getLogger(ZooKeeperDemo.class);

    private static final int SESSION_TIME_OUT = 10000;
    // ZooKeeper服务的地址，如为集群，多个地址用逗号分隔
    private static final String CONNECT_STRING = "127.0.0.1:2181";
    private static final String ZNODE_PATH = "/zk_demo";
    private static final String ZNODE_PATH_PARENT = "/app1";
    private static final String ZNODE_PATH_CHILDREN = "/app1/app1_1";

    private ZooKeeper zk = null;

    @Before
    public void init() throws IOException {
        zk = new ZooKeeper(CONNECT_STRING, SESSION_TIME_OUT, new Watcher(){
            @Override
            public void process(WatchedEvent event) {
                System.out.println("已经触发了" + event.getType() + "事件！");
            }
        });

    }

    @Test
    public void testCreate() throws KeeperException, InterruptedException {
        zk.create(ZNODE_PATH, "anna2019".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE, Create
ode.PERSISTENT);
```

```
    }

    @Test
    public void testCreateParentZnode() throws KeeperException, InterruptedException {
        zk.create(ZNODE_PATH_PARENT, "anna2019".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE
 CreateMode.PERSISTENT);
    }

    @Test
    public void testCreateChildrenZnode() throws KeeperException, InterruptedException {
        zk.create(ZNODE_PATH_CHILDREN, "anna2020".getBytes(), ZooDefs.Ids.OPEN_ACL_UNS
FE, CreateMode.PERSISTENT);
    }

    @Test
    public void testGet() throws KeeperException, InterruptedException {
        byte[] data1 = zk.getData(ZNODE_PATH, false, null);
        byte[] data2 = zk.getData(ZNODE_PATH_PARENT, false, null);
        byte[] data3 = zk.getData(ZNODE_PATH_CHILDREN, false, null);
        LOGGER.info("{}的信息：{}", ZNODE_PATH, new String(data1) );
        LOGGER.info("{}的信息：{}", ZNODE_PATH_PARENT, new String(data2) );
        LOGGER.info("{}的信息：{}", ZNODE_PATH_CHILDREN, new String(data3) );
    }

    /**
     * 删除
     * @throws KeeperException
     * @throws InterruptedException
     */
    @Test
    public void testDelete() throws KeeperException, InterruptedException {
        // 指定要删除的版本，-1表示删除所有版本
        zk.delete(ZNODE_PATH, -1);
    }

    /**
     * 删除含有子节点
     * @throws KeeperException
     * @throws InterruptedException
     */
    @Test
    public void testDeleteHasChildrenZnode() throws KeeperException, InterruptedException {
        // 指定要删除的版本，-1表示删除所有版本
        zk.delete(ZNODE_PATH_PARENT, -1);
    }

    @Test
    public void testSet() throws KeeperException, InterruptedException {
        Stat stat = zk.setData(ZNODE_PATH, "yuanrengu".getBytes(), -1);
        LOGGER.info(stat.toString());
    }

}
```

上面有用到@Before，简单说明下：

- @BeforeClass – 表示在类中的任意public static void方法执行之前执行
- @AfterClass – 表示在类中的任意public static void方法执行之后执行
- @Before – 表示在任意使用@Test注解标注的public void方法执行之前执行
- @After – 表示在任意使用@Test注解标注的public void方法执行之后执行
- @Test – 使用该注解标注的public void方法会表示为一个测试方法

如果将**SESSION_TIME_OUT**设置的时间太短，会报API客户端异常：org.apache.zookeeper.KeepeException$ConnectionLossException: KeeperErrorCode = ConnectionLoss for /zk_demo 。完的报错信息如下：

```
09:33:52.139 [main-SendThread(106.12.111.172:2181)] DEBUG org.apache.zookeeper.ClientCnnSocketNIO - Ignoring exception during shutdown input
java.net.SocketException: Socket is not connected
    at sun.nio.ch.Net.translateToSocketException(Net.java:123)
    at sun.nio.ch.Net.translateException(Net.java:157)
    at sun.nio.ch.Net.translateException(Net.java:163)
    at sun.nio.ch.SocketAdaptor.shutdownInput(SocketAdaptor.java:401)
    at org.apache.zookeeper.ClientCnxnSocketNIO.cleanup(ClientCnxnSocketNIO.java:198)
    at org.apache.zookeeper.ClientCnxn$SendThread.cleanup(ClientCnxn.java:1338)
    at org.apache.zookeeper.ClientCnxn$SendThread.cleanAndNotifyState(ClientCnxn.java:1276

    at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1254)
Caused by: java.nio.channels.NotYetConnectedException: null
    at sun.nio.ch.SocketChannelImpl.shutdownInput(SocketChannelImpl.java:782)
    at sun.nio.ch.SocketAdaptor.shutdownInput(SocketAdaptor.java:399)
    ... 4 common frames omitted
09:33:52.140 [main-SendThread(106.12.111.172:2181)] DEBUG org.apache.zookeeper.ClientCnnSocketNIO - Ignoring exception during shutdown output
java.net.SocketException: Socket is not connected
    at sun.nio.ch.Net.translateToSocketException(Net.java:123)
    at sun.nio.ch.Net.translateException(Net.java:157)
    at sun.nio.ch.Net.translateException(Net.java:163)
    at sun.nio.ch.SocketAdaptor.shutdownOutput(SocketAdaptor.java:409)
    at org.apache.zookeeper.ClientCnxnSocketNIO.cleanup(ClientCnxnSocketNIO.java:205)
    at org.apache.zookeeper.ClientCnxn$SendThread.cleanup(ClientCnxn.java:1338)
    at org.apache.zookeeper.ClientCnxn$SendThread.cleanAndNotifyState(ClientCnxn.java:1276

    at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1254)
Caused by: java.nio.channels.NotYetConnectedException: null
    at sun.nio.ch.SocketChannelImpl.shutdownOutput(SocketChannelImpl.java:799)
    at sun.nio.ch.SocketAdaptor.shutdownOutput(SocketAdaptor.java:407)
    ... 4 common frames omitted

org.apache.zookeeper.KeeperException$ConnectionLossException: KeeperErrorCode = ConnectionLoss for /zk_demo

    at org.apache.zookeeper.KeeperException.create(KeeperException.java:102)
    at org.apache.zookeeper.KeeperException.create(KeeperException.java:54)
    at org.apache.zookeeper.ZooKeeper.getData(ZooKeeper.java:2131)
    at org.apache.zookeeper.ZooKeeper.getData(ZooKeeper.java:2160)
```

```
    at com.yuanrengu.demo.ZooKeeperDemo.testGet(ZooKeeperDemo.java:48)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
3)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:5
)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:56)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
    at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
    at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
    at org.junit.runners.BlockJUnit4ClassRunner$1.evaluate(BlockJUnit4ClassRunner.java:100)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:366)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:103)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:63)
    at org.junit.runners.ParentRunner$4.run(ParentRunner.java:331)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:79)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:329)
    at org.junit.runners.ParentRunner.access$100(ParentRunner.java:66)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:293)
    at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:413)
    at org.junit.runner.JUnitCore.run(JUnitCore.java:137)
    at com.intellij.junit4.JUnit4IdeaTestRunner.startRunnerWithArgs(JUnit4IdeaTestRunner.java:
8)
    at com.intellij.rt.execution.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRu
ner.java:47)
    at com.intellij.rt.execution.junit.JUnitStarter.prepareStreamsAndStart(JUnitStarter.java:242)
    at com.intellij.rt.execution.junit.JUnitStarter.main(JUnitStarter.java:70)

Disconnected from the target VM, address: '127.0.0.1:60454', transport: 'socket'

Process finished with exit code -1
```

起初以为是ZooKeeper服务部署有问题或服务没启动，经检查确认无误后，debug调试发现，是SESSION_TIME_OUT = 2000;设置的值太小，改为10000后，不再报错。

SESSION_TIME_OUT 是会话超时时间，也就是当一个zookeeper超过该时间没有心跳，则认为该节故障。所以，如果此值小于zookeeper的创建时间，则当zookeeper还未来得及创建连接，会话时间到，因此抛出异常认为该节点故障。

3.1 创建会话

通过创建一个ZooKeeper实例来连接ZooKeeper服务器（详见ZooKeeper单机版、伪集群和集群环搭建）。

官方提供了10种ZooKeeper构造方法和描述如下：

Constructor	Description
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, boolean canBeReadOnly)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, boolean canBeReadOnly, HostProvider aHostProvider)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, boolean canBeReadOnly, HostProvider aHostProvider, ZKClientConfig clientConfig)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, boolean canBeReadOnly, ZKClientConfig conf)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd, boolean canBeReadOnly)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd, boolean canBeReadOnly, HostProvider aHostProvider)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd, boolean canBeReadOnly, HostProvider aHostProvider, ZKClientConfig clientConfig)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.
ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, ZKClientConfig conf)	To create a ZooKeeper client object, the application needs to pass a connection string containing a comma separated list of host:port pairs, each corresponding to a ZooKeeper server.

关于每种构造方法的英文描述用词都很简单，基本都能看的很明白，根据实际的应用场景选取相应的造方法。

有传入参数中包括sessionId和sessionPasswd的构造方法，分别代表会话ID和会话密钥。这两个参能够唯一确定一个会话，同时客户端使用这两个参数可以实现客户端会话复用，从而达到恢复会话的果。具体使用方法是第一次连接上ZooKeeper服务器时，通过调用ZooKeeper对象实例的以下两个口，即可获取当前会话的ID和密钥：long getSessionId(); byte[] getSessionPasswd();获取到这两参数值之后，就可以在下次创建ZooKeeper对象实例的时候传入构造方法了。

选取几个典型的构造方法来带领大家解读下文档。

3.1.1 ZooKeeper(String connectString, int sessionTimeout, Watcher watcher)

```
public ZooKeeper(String connectString,
                 int sessionTimeout,
                 Watcher watcher)
        throws IOException
```

To create a ZooKeeper client object, the application needs to pass a connection string containg a comma separated list of host:port pairs, each corresponding to a ZooKeeper server. Session establishment is asynchronous. This constructor will initiate connection to the server nd return immediately - potentially (usually) before the session is fully established. The watcher argument specifies the watcher that will be notified of any changes in state. This notification can come at any point before or after the constructor call has returned.

The instantiated ZooKeeper client object will pick an arbitrary server from the connectString nd attempt to connect to it. If establishment of the connection fails, another server in the connect string will be tried (the order is non-deterministic, as we random shuffle the list), until a cnnection is established. The client will continue attempts until the session is explicitly closed.

Added in 3.2.0: An optional "chroot" suffix may also be appended to the connection string. Ths will run the client commands while interpreting all paths relative to this root (similar to the nix chroot command).

Parameters:
connectString - comma separated host:port pairs, each corresponding to a zk server. e.g. "1270.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002" If the optional chroot suffix is used the example would look like: "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002/app/a" where the client would be roted at "/app/a" and all paths would be relative to this root - ie getting/setting/etc... "/foo/bar" would result in operations being run on "/app/a/foo/bar" (from the server perspective).
sessionTimeout - session timeout in milliseconds
watcher - a watcher object which will be notified of state changes, may also be notified for nde events
Throws:
IOException - in cases of network failure
IllegalArgumentException - if an invalid chroot path is specified

有一点需要特别说明下，文档说客户端和服务端建立会话是异步的。构造方法会在处理完客户端初始工作后立即返回，在通常情况下，此时并没有真正建立好一个可用的会话，此时在会话的生命周期中于“CONNECTING”的状态。当该会话真正创建完毕后，ZooKeeper服务端会向会话对应的客户端送一个事件通知，以告知客户端，客户端只有在获取这个通知后，才算真正建立了会话。

实例化的ZooKeeper客户端对象将从connectString列举的服务器中随机选择一个服务器，并尝试连到该服务器。如果建立连接失败，将尝试连接另一个服务器（顺序是不确定的，因为列举的服务器是机洗牌的），直到建立连接。即客户端连接一个服务器失败，将继续尝试，直到会话显式关闭。

从3.2.0版本开始添加了可选的"chroot"后缀，意思就是可将“chroot”加在connectString中列举的务器后面，即客户端连上ZooKeeper服务器后，所有对ZooKeeper的操作都会基于这个根目录。

对参数做下简要说明：

参数	描述
connectString	指定ZooKeeper服务器列表，有文逗号分隔的host：port字符串组成，如"127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002"。可以指客户端连上connectString中服务器后的根目录，如 "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002/pp/a"，对ZooKeeper的操作都会基于/app/a这个根目录，即创建路径为"/foo/bar"的节点，实际节点的路径为"/app/a/foo/bar" 。
sessionTimeout	会话的超时事件，以“毫秒”单位的整型值。在一个会话周期内，ZooKeeper客户端和服务器之间会通过心跳检测机制来维持会话有效性，一旦在sessionTimeout时间内没有进行有效的心跳检测，会话就会失效。
watcher	ZooKeeper允许客户端在构造方法中传

一个接口Watcher（org.apache.zookeeper.Watcher）的实现类对象来作为默认的Watch事件通知。**该参数也可以设置为null，表明不需要设置默认的Watch处理器。**

3.1.2 ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, boolean canBeReadOnly)

```
public ZooKeeper(String connectString,
                 int sessionTimeout,
                 Watcher watcher,
                 boolean canBeReadOnly)
          throws IOException
```

To create a ZooKeeper client object, the application needs to pass a connection string containng a comma separated list of host:port pairs, each corresponding to a ZooKeeper server. Session establishment is asynchronous. This constructor will initiate connection to the server nd return immediately - potentially (usually) before the session is fully established. The watchr argument specifies the watcher that will be notified of any changes in state. This notification can come at any point before or after the constructor call has returned.

The instantiated ZooKeeper client object will pick an arbitrary server from the connectString nd attempt to connect to it. If establishment of the connection fails, another server in the conect string will be tried (the order is non-deterministic, as we random shuffle the list), until a cnnection is established. The client will continue attempts until the session is explicitly closed.

Added in 3.2.0: An optional "chroot" suffix may also be appended to the connection string. Ths will run the client commands while interpreting all paths relative to this root (similar to the nix chroot command).

Parameters:
connectString - comma separated host:port pairs, each corresponding to a zk server. e.g. "1270.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002" If the optional chroot suffix is used the example would look like: "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002/app/a" where the client would be roted at "/app/a" and all paths would be relative to this root - ie getting/setting/etc... "/foo/bar" would result in operations being run on "/app/a/foo/bar" (from the server perspective).
sessionTimeout - session timeout in milliseconds
watcher - a watcher object which will be notified of state changes, may also be notified for nde events
canBeReadOnly - (added in 3.4) whether the created client is allowed to go to read-only mod in case of partitioning. Read-only mode basically means that if the client can't find any majoriy servers but there's partitioned server it could reach, it connects to one in read-only mode, i.e read requests are allowed while write requests are not. It continues seeking for majority in th background.
Throws:
IOException - in cases of network failure
IllegalArgumentException - if an invalid chroot path is specified

这个构造方法跟上个方法非常相似，只是从3.4版本开始多了一个canBeReadOnly参数，用于标识当会话是否支持“read-only”模式（只读模式）。默认情况下，在ZooKeeper集群中，一个节点如果集群中过半及以上节点失去网络连接（建立不了连接），那么这个机器将不再处理客户端请求（包括写请求）。但是在某些使用场景下，当ZooKeeper服务器发生此类故障的时候，还是希望ZooKeepe服务器能够提供读服务（写服务肯定无法提供），这就是ZooKeeper的“read-only”模式。但客户可以连接某一分区的服务器，它将以**只读模式**连接到其中一个服务器，允许读取请求，而不允许写入求，然后继续在后台寻找更多数的服务器(**这一句我描述的不够简练精准**)。

3.2 新增

```
public String create(String path,
                     byte[] data,
                     List<ACL> acl,
                     CreateMode createMode)
              throws KeeperException,
                     InterruptedException
```

Create a node with the given path. The node data will be the given data, and node acl will be he given acl.
The flags argument specifies whether the created node will be ephemeral or not.

An ephemeral node will be removed by the ZooKeeper automatically when the session associ ted with the creation of the node expires.

The flags argument can also specify to create a sequential node. The actual path name of a s quential node will be the given path plus a suffix "i" where i is the current sequential number f the node. The sequence number is always fixed length of 10 digits, 0 padded. Once such a n de is created, the sequential number will be incremented by one.

If a node with the same actual path already exists in the ZooKeeper, a KeeperException with e ror code KeeperException.NodeExists will be thrown. Note that since a different actual path is used for each invocation of creating sequential node with the same path argument, the call wi l never throw "file exists" KeeperException.

If the parent node does not exist in the ZooKeeper, a KeeperException with error code Keepe Exception.NoNode will be thrown.

An ephemeral node cannot have children. If the parent node of the given path is ephemeral, a KeeperException with error code KeeperException.NoChildrenForEphemerals will be thrown.

This operation, if successful, will trigger all the watches left on the node of the given path by xists and getData API calls, and the watches left on the parent node by getChildren API calls.

If a node is created successfully, the ZooKeeper server will trigger the watches on the path left by exists calls, and the watches on the parent of the node by getChildren calls.

The maximum allowable size of the data array is 1 MB (1,048,576 bytes). Arrays larger than thi will cause a KeeperExecption to be thrown.

Parameters:
path - the path for the node
data - the initial data for the node
acl - the acl for the node
createMode - specifying whether the node to be created is ephemeral and/or sequential
Returns:
the actual path of the created node
Throws:
KeeperException - if the server returns a non-zero error code
KeeperException.InvalidACLException - if the ACL is invalid, null, or empty
InterruptedException - if the transaction is interrupted
IllegalArgumentException - if an invalid path is specified

Talk is cheap. Show me the code.这里我们不瞎BB，直接上官方文档。官方文档是不是很容易看懂而且解释的非常清楚（而且稍显啰嗦的感觉）？

这里简单列下文档中的几个关键点：

1. 按指定路径和节点形式创建，可指定节点为持久节点、临时节点等。

这里要说下CreateMode,大家可能都说ZooKeeper只有4种形式的节点（持久、临时、持久顺序、临顺序），看文档的话，其实是有7种形式的。

```
public enum CreateMode {
    PERSISTENT(0, false, false, false, false),
    PERSISTENT_SEQUENTIAL(2, false, true, false, false),
    EPHEMERAL(1, true, false, false, false),
    EPHEMERAL_SEQUENTIAL(3, true, true, false, false),
    CONTAINER(4, false, false, true, false),
    PERSISTENT_WITH_TTL(5, false, false, false, true),
    PERSISTENT_SEQUENTIAL_WITH_TTL(6, false, true, false, true);
}
```

- PERSISTENT：持久节点（也有叫永久节点的），不会随着会话的结束而自动删除。
- PERSISTENT_SEQUENTIAL：带单调递增序号的持久节点，不会随着会话的结束而自动删除。
- EPHEMERAL：临时节点，会随着会话的结束而自动删除。
- EPHEMERAL_SEQUENTIAL：带单调递增序号的临时节点，会随着会话的结束而自动删除。
- CONTAINER：容器节点，用于Leader、Lock等特殊用途，当容器节点不存在任何子节点时，容器成为服务器在将来某个时候删除的候选节点。
- PERSISTENT_WITH_TTL：带TTL（time-to-live，存活时间）的持久节点，节点在TTL时间之内没得到更新并且没有子节点，就会被自动删除。
- PERSISTENT_SEQUENTIAL_WITH_TTL：带TTL（time-to-live，存活时间）和单调递增序号的持节点，节点在TTL时间之内没有得到更新并且没有子节点，就会被自动删除。

2. 如果指令路径和版本的节点已经存在，则会抛出一个KeeperException异常。

3. 临时节点不能有子节点。如果给临时节点创建子节点会抛KeeperException异常。

4. 临时节点的生命周期与客户端会话绑定。 **一旦客户端会话失效（客户端与 Zookeeper连接断开一定会话失效），那么这个客户端创建的所有临时节点都会被移除。**

5. byte[] data允许的最大数据量为1MB（1,048,576 bytes）。如果超过，会抛KeeperExecption。

运行创建节点的代码：

```
    @Test
    public void testCreate() throws KeeperException, InterruptedException {
        zk.create(ZNODE_PATH, "anna2019".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE, Create
ode.PERSISTENT);
    }
```

可以通过日志信息得到节点创建成功：

```
DEBUG org.apache.zookeeper.ClientCnxn - Reading reply sessionid:0x101402626bb000b, pac
et:: clientPath:null serverPath:null finished:false header:: 1,1  replyHeader:: 1,12884901937,0  r
quest:: '/zk_demo,#616e6e6132303139,v{s{31,s{'world,'anyone}}},0  response:: '/zk_demo
```

在服务端查看,/zk_demo节点创建成功:

```
[zk: 127.0.0.1:2181(CONNECTED) 21] ls /
[zookeeper, zk_demo]
[zk: 127.0.0.1:2181(CONNECTED) 22] stat /zk_demo
cZxid = 0x300000031
ctime = Tue Dec 17 12:52:50 CST 2019
mZxid = 0x300000031
mtime = Tue Dec 17 12:52:50 CST 2019
pZxid = 0x300000031
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 8
numChildren = 0
```

3.3 获取

```
public byte[] getData(String path,
                      boolean watch,
                      Stat stat)
               throws KeeperException,
                      InterruptedException
```

Return the data and the stat of the node of the given path.
If the watch is true and the call is successful (no exception is thrown), a watch will be left on t e node with the given path. The watch will be triggered by a successful operation that sets da a on the node, or deletes the node.

A KeeperException with error code KeeperException.NoNode will be thrown if no node with t e given path exists.

Parameters:
path - the given path
watch - whether need to watch this node
stat - the stat of the node
Returns:
the data of the node
Throws:
KeeperException - If the server signals an error with a non-zero error code
InterruptedException - If the server transaction is interrupted.

指定路径的节点不存在时就抛KeeperException.NoNode异常。

运行:

```
    @Test
    public void testGet() throws KeeperException, InterruptedException {
        byte[] data1 = zk.getData(ZNODE_PATH, false, null);
        byte[] data2 = zk.getData(ZNODE_PATH_PARENT, false, null);
        byte[] data3 = zk.getData(ZNODE_PATH_CHILDREN, false, null);
        LOGGER.info("{}的信息:  {}", ZNODE_PATH, new String(data1) );
        LOGGER.info("{}的信息:  {}", ZNODE_PATH_PARENT, new String(data2) );
```

```
        LOGGER.info("{}的信息：{}", ZNODE_PATH_CHILDREN, new String(data3) );
    }
```

结果：

```
13:51:00.288 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - /zk_demo的信息：anna20
9
13:51:00.288 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - /app1的信息：anna2019
13:51:00.289 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - /app1/app1_1的信息：ann
2020
```

3.4 更新

```
public Stat setData(String path,
                    byte[] data,
                    int version)
             throws KeeperException,
                    InterruptedException

Set the data for the node of the given path if such a node exists and the given version match
s the version of the node (if the given version is -1, it matches any node's versions). Return th
 stat of the node.
This operation, if successful, will trigger all the watches on the node of the given path left by
etData calls.

A KeeperException with error code KeeperException.NoNode will be thrown if no node with t
e given path exists.

A KeeperException with error code KeeperException.BadVersion will be thrown if the given ve
sion does not match the node's version.

The maximum allowable size of the data array is 1 MB (1,048,576 bytes). Arrays larger than thi
 will cause a KeeperException to be thrown.

Parameters:
path - the path of the node
data - the data to set
version - the expected matching version
Returns:
the state of the node
Throws:
InterruptedException - If the server transaction is interrupted.
KeeperException - If the server signals an error with a non-zero error code.
IllegalArgumentException - if an invalid path is specified
```

主要注意以下几点：

1. 版本为-1时，即代表适配指定路径节点的所有版本。

2. 如果指定路径的节点不存在会抛 **KeeperException.NoNode**异常，该节点没有传入的版本，会抛**eeperException.BadVersion**异常。

3. byte[] data允许的最大数据量为1MB（1,048,576 bytes）。如果超过，会抛KeeperExecption。

运行：

```
    @Test
    public void testSet() throws KeeperException, InterruptedException {
        Stat stat = zk.setData(ZNODE_PATH, "yuanrengu".getBytes(), -1);
        byte[] data = zk.getData(ZNODE_PATH, false, null);
        LOGGER.info(new String(data));
    }
```

可以看到数据已经更新：

```
15:46:16.472 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - yuanrengu
```

更新的接口提到了版本的概念，上面提到版本为-1时，即代表适配指定路径节点的所有版本。节点每setData时版本会加1，更新时指定的版本不存在会报KeeperException.BadVersion异常。我们做个试：

```
    @Test
    public void testSetForVersion() throws KeeperException, InterruptedException {
        String pathVersion = "/versionDemo";
        zk.create(pathVersion, "yuanrengu2019".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE, Cre
teMode.PERSISTENT);

        Stat stat = zk.setData(pathVersion, "yuanrengu2020".getBytes(), -1);
        LOGGER.info("============1111111  start======================");
        LOGGER.info(String.valueOf(stat));
        LOGGER.info("version:{}", stat.getVersion());
        byte[] data1 = zk.getData(pathVersion, false, null);
        LOGGER.info("data1:{}", new String(data1));
        LOGGER.info("============1111111  end======================");

        Stat stat2 = zk.setData(pathVersion, "yuanrengu2021".getBytes(), stat.getVersion());
        LOGGER.info("============222222  start======================");
        LOGGER.info(String.valueOf(stat2));
        LOGGER.info("version2:{}", stat2.getVersion());
        byte[] data2 = zk.getData(pathVersion, false, null);
        LOGGER.info("data2:{}", new String(data2));
        LOGGER.info("============222222  end======================");

        Stat stat3 = zk.setData(pathVersion, "yuanrengu2022".getBytes(), stat.getVersion());
        LOGGER.info("============3333333  start======================");
        LOGGER.info(String.valueOf(stat3));
        LOGGER.info("version3:{}", stat3.getVersion());
        byte[] data3 = zk.getData(pathVersion, false, null);
        LOGGER.info("data3:{}", new String(data3));
        LOGGER.info("============3333333  end======================");
    }
```

运行结果如下：

```
09:56:00.931 [main-SendThread(106.12.111.172:2181)] DEBUG org.apache.zookeeper.ClientCn
n - Reading reply sessionid:0x1014dce26220008, packet:: clientPath:null serverPath:null finish
d:false header:: 1,5  replyHeader:: 1,12884902005,0  request:: '/versionDemo,#7975616e72656
677532303230,-1  response:: s{12884901996,12884902005,1576720362715,1576720560918,1,
,0,0,13,0,12884901996}
09:56:00.940 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - ============111111
  start======================
```

```
09:56:00.941 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - 12884901996,128849020
5,1576720362715,1576720560918,1,0,0,0,13,0,12884901996

09:56:00.942 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - version:1
09:56:00.971 [main-SendThread(106.12.111.172:2181)] DEBUG org.apache.zookeeper.ClientCn
n - Reading reply sessionid:0x1014dce26220008, packet:: clientPath:null serverPath:null finish
d:false header:: 2,4  replyHeader:: 2,12884902005,0  request:: '/versionDemo,F  response:: #79
5616e72656e677532303230,s{12884901996,12884902005,1576720362715,1576720560918,1,0
0,0,13,0,12884901996}
09:56:00.971 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - data1:yuanrengu2020
09:56:00.971 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - ============111111
  end======================
09:56:00.988 [main-SendThread(106.12.111.172:2181)] DEBUG org.apache.zookeeper.ClientCn
n - Reading reply sessionid:0x1014dce26220008, packet:: clientPath:null serverPath:null finish
d:false header:: 3,5  replyHeader:: 3,12884902006,0  request:: '/versionDemo,#7975616e72656
677532303231,1  response:: s{12884901996,12884902006,1576720362715,1576720561002,2,0
0,0,13,0,12884901996}
09:56:00.988 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - ============222222
start======================
09:56:00.988 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - 12884901996,128849020
6,1576720362715,1576720561002,2,0,0,0,13,0,12884901996

09:56:00.990 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - version2:2
09:56:01.017 [main-SendThread(106.12.111.172:2181)] DEBUG org.apache.zookeeper.ClientCn
n - Reading reply sessionid:0x1014dce26220008, packet:: clientPath:null serverPath:null finish
d:false header:: 4,4  replyHeader:: 4,12884902006,0  request:: '/versionDemo,F  response:: #79
5616e72656e677532303231,s{12884901996,12884902006,1576720362715,1576720561002,2,0
0,0,13,0,12884901996}
09:56:01.017 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - data2:yuanrengu2021
09:56:01.017 [main] INFO com.yuanrengu.demo.ZooKeeperDemo - ============222222
end======================
09:56:01.037 [main-SendThread(106.12.111.172:2181)] DEBUG org.apache.zookeeper.ClientCn
n - Reading reply sessionid:0x1014dce26220008, packet:: clientPath:null serverPath:null finish
d:false header:: 5,5  replyHeader:: 5,12884902007,-103  request:: '/versionDemo,#7975616e72
56e677532303232,1  response::

org.apache.zookeeper.KeeperException$BadVersionException: KeeperErrorCode = BadVersion
for /versionDemo

    at org.apache.zookeeper.KeeperException.create(KeeperException.java:122)
    at org.apache.zookeeper.KeeperException.create(KeeperException.java:54)
    at org.apache.zookeeper.ZooKeeper.setData(ZooKeeper.java:2384)
```

测试代码进行了3次setData操作，第一次setData时传入的版本为-1，成功后version变为1；第二次setData时传入的version为1，成功后version变为2；第三次setData时传入的版本为1，此时就抛了KeeperException.BadVersion异常。如果第三次setData传入的版本为-1，能更新成功。

3.5 删除

```
public void delete(String path,
                   int version)
            throws InterruptedException,
                   KeeperException
```

Delete the node with the given path. The call will succeed if such a node exists, and the given ersion matches the node's version (if the given version is -1, it matches any node's versions). A KeeperException with error code KeeperException.NoNode will be thrown if the nodes does not exist.

A KeeperException with error code KeeperException.BadVersion will be thrown if the given ve sion does not match the node's version.

A KeeperException with error code KeeperException.NotEmpty will be thrown if the node has hildren.

This operation, if successful, will trigger all the watches on the node of the given path left by xists API calls, and the watches on the parent node left by getChildren API calls.

Parameters:
path - the path of the node to be deleted.
version - the expected node version.
Throws:
InterruptedException - IF the server transaction is interrupted
KeeperException - If the server signals an error with a non-zero return code.
IllegalArgumentException - if an invalid path is specified

节点可能含有子节点，删除节点的操作有几点需要特别注意：

1. 版本为-1时，即代表适配指定路径节点的所有版本。

2. 如果指定路径的节点不存在会抛KeeperException.NoNode异常，该节点没有传入的版本，会抛KeperException.BadVersion异常。

3. 如果节点含有子节点，删除父节点(parent node)时会抛KeeperException.NotEmpty异常。

在ZooKeeper中，只允许删子节点。如果一个节点存在一个或多个子节点，该节点就无法被直接删，必须先删除所有子节点。

/app1有子节点，我们做下删除操作：

```
/**
 *  删除含有子节点的父节点
 * @throws KeeperException
 * @throws InterruptedException
 */
@Test
public void testDeleteHasChildrenZnode() throws KeeperException, InterruptedException {
    // 指定要删除的版本，-1表示删除所有版本
    zk.delete(ZNODE_PATH_PARENT, -1);
}
```

可以看到日志：

```
org.apache.zookeeper.KeeperException$NotEmptyException: KeeperErrorCode = Directory no
 empty for /app1

    at org.apache.zookeeper.KeeperException.create(KeeperException.java:132)
    at org.apache.zookeeper.KeeperException.create(KeeperException.java:54)
    at org.apache.zookeeper.ZooKeeper.delete(ZooKeeper.java:1793)
    at com.yuanrengu.demo.ZooKeeperDemo.testDeleteHasChildrenZnode(ZooKeeperDemo.ja
```

a:89)

4 总结

上面我们实现了节点的增、删、改、查的测试，后面的篇章会有更多好玩的用法，如实现分布式锁、置中心等。

基于上面的分析，总结几个注意的点：

1. 节点有　7种形式：

- PERSISTENT：持久节点（也有叫永久节点的），不会随着会话的结束而自动删除。
- PERSISTENT_SEQUENTIAL：带单调递增序号的持久节点，不会随着会话的结束而自动删除。
- EPHEMERAL：临时节点，会随着会话的结束而自动删除。
- EPHEMERAL_SEQUENTIAL：带单调递增序号的临时节点，会随着会话的结束而自动删除。
- CONTAINER：容器节点，用于Leader、Lock等特殊用途，当容器节点不存在任何子节点时，容器成为服务器在将来某个时候删除的候选节点。
- PERSISTENT_WITH_TTL：带TTL（time-to-live，存活时间）的持久节点，节点在TTL时间之内没得到更新并且没有子节点，就会被自动删除。
- PERSISTENT_SEQUENTIAL_WITH_TTL：带TTL（time-to-live，存活时间）和单调递增序号的持节点，节点在TTL时间之内没有得到更新并且没有子节点，就会被自动删除。

2. **临时节点不能有子节点。**如果给临时节点创建子节点会抛KeeperException异常。

3. 临时节点的生命周期与客户端会话绑定。　**一旦客户端会话失效（客户端与 Zookeeper连接断开一定会话失效），那么这个客户端创建的所有临时节点都会被移除。**

4. byte[] data允许的最大数据量为1MB（1,048,576 bytes）。