# 从Paxos到Zookeeper分布式一致性原理与实践

# Zookeeper与Paxos

## 4.1 初始Zookeeper

### 4.1.1 Zookeeper介绍

集群角色：Leader、Follower、Observer

Leader：集群中的所有机器通过一个Leader选举过程来选定一台被称为“Leader”的机器。Leader服务器为客户端提供读和写服务。

Follower和Observer都能够提供读服务，唯一的区别在于，Observer不参与Leader选举过程，也不参与写操作的“过半写成功”策略。

会话（Session）：在Zookeeper中，一个客户端连接是指客户端和服务器之间的一个TCP长连接。当由于服务器眼里太大、网络故障或是客户端主动断开连接等各种原因导致客户端连接断开时，只要在sessionTimeout规定的时间内能够重新连接上集群中任意一台服务器，那么之前创建的会话仍然有效。

节点：第一类构成集群的机器（机器节点），第二类是指数据模型中的数据单元（数据节点）。

ZNode可以分为持久节点和临时节点。

持久节点：一旦ZNode被创建了，除非主动进行ZNode的移除操作，否则这个ZNode将一直保存在Zookeeper上。

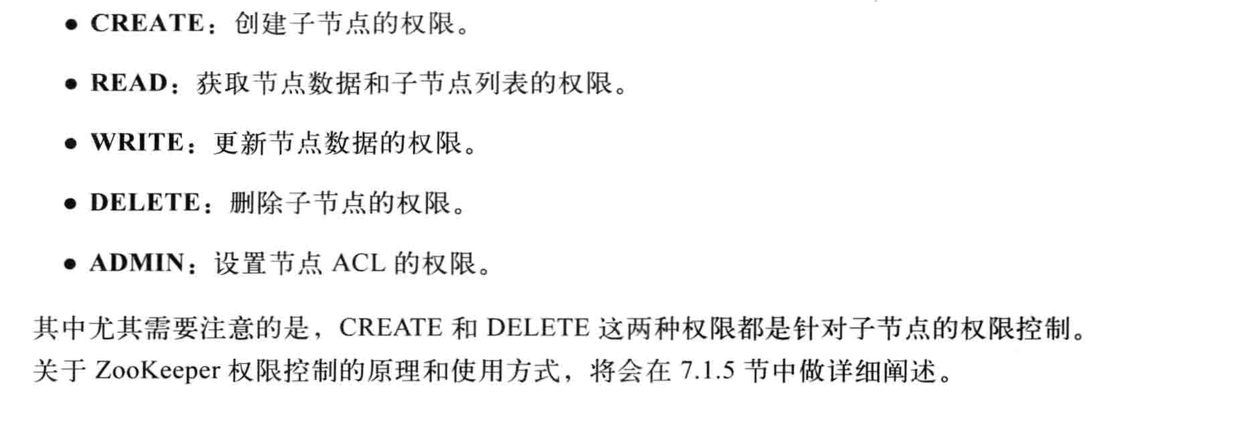
临时节点：生命周期和客户端会话绑定。会话失效，客户端创建的所有临时节点都会被移除。

版本：

Zookeeper为每个ZNode维护一个叫做Stat的数据结构，Stat中记录了这个ZNode的是三个数据版本，分别是version(当前ZNode版本)、cversion（当前ZNode子结点的版本）和aversion(当前ZNode的ACL版本)。

Watcher（事件监听器）：

ACL（Access Control Lists）策略来进行权限控制。



### 4.1.2 Zookeeper介绍

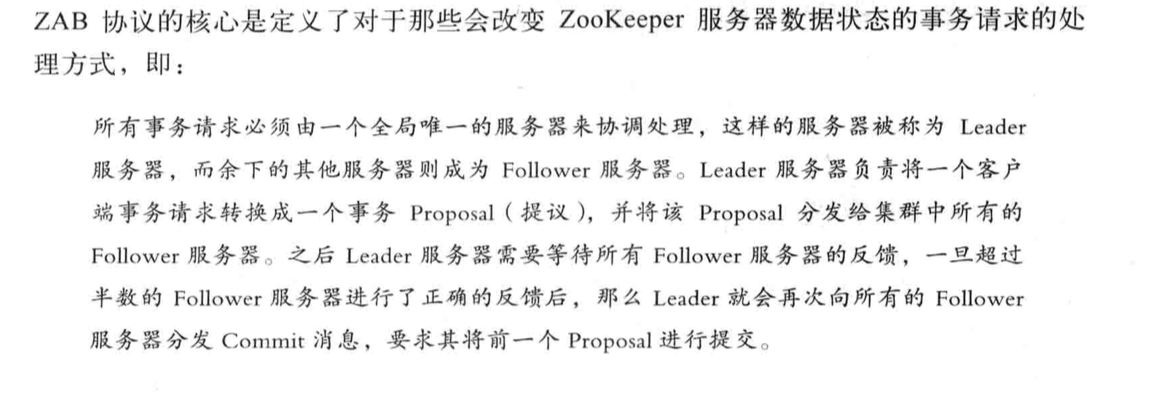
### 4.1.3 Zookeeper介绍

### 4.1.4 Zookeeper介绍

## 4.2 Zookeeper的ZAB协议

Zookeeper Atomic Broadcast(ZAB，Zookeeper原子消息广播协议)的协议作为其数据的一致性核心算法。支持崩溃恢复的原子广播协议。

Zookeeper实现了会中准备模式的系统架构来保持集群中各副本之间数据的一致性。



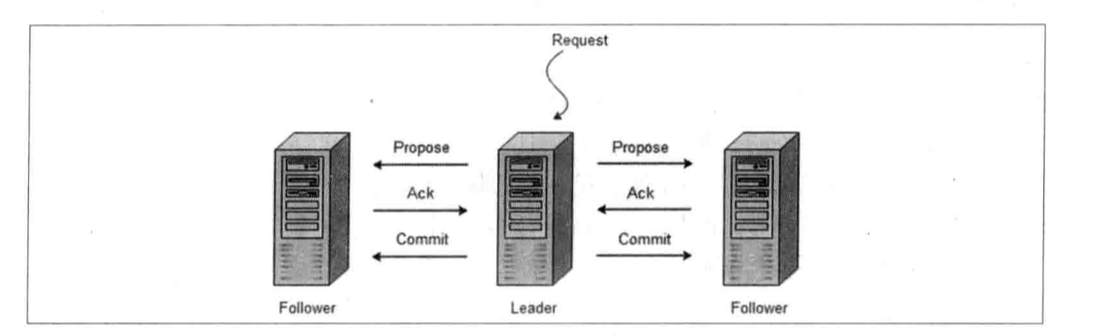
ZAB协议包括两种模式：崩溃恢复和消息广播。

当整个服务器在启动过程中，或是当Leader服务器出现网络中断、崩溃退出与重启等异常情况时，ZAB协议就会进入恢复模式并选举产生新的Leader服务器。当选举产生了新的Leader服务器，同时集群中有过半的服务器与该Leader服务器完成了状态同步之后，ZAB协议就会退出恢复模式。状态同步是指数据同步，保证集群中过半的机器能够和Leader服务器的数据状态保持一致。

集群中过半Follower服务器完成了和Leader服务器的状态同步，那么整个服务框架就可以进入消息广播模式了。当新加入一个台ZAB协议的服务器，且已经存在Leader服务器，那么新加入的服务器自动进入数据恢复模式：找到Leader所在服务器，并与其进行数据同步，然后一起参与到消息广播流程中取。

消息广播：

ZAB协议的消息广播过程使用了一个院子广播协议，



在整个消息广播过程中，Leader服务器会为每个拾取请求生成对应的Proposal来进行广播，

并且在广播拾取Proposal之前，Leader服务器会首先为这个事务Proposal分配一个全局单调递增的唯一ID，我们称之为事务ID(即ZXID)。由于ZAB协议需要保证每一个消息的因果关系，因此必须将每一个事务Proposal按照其ZXID的先后顺序来进行排序与处理。

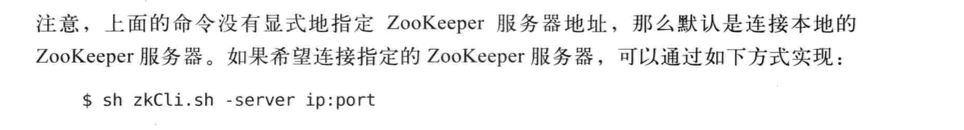
崩溃恢复：

数据同步：

分别是发现(Discovery)、同步(Synchronization)和广播(Boradcast).

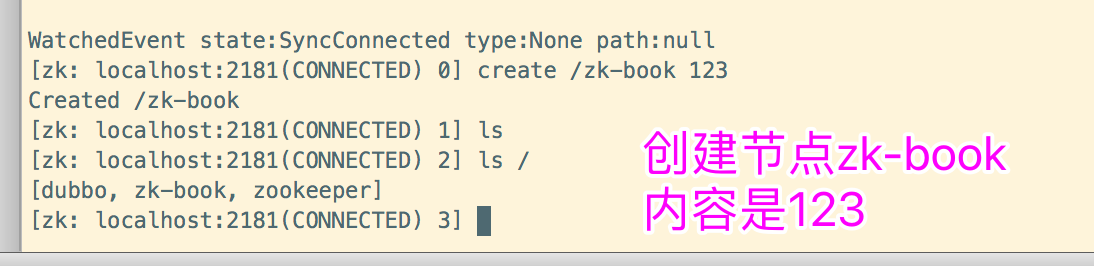
# 使用Zookeeper

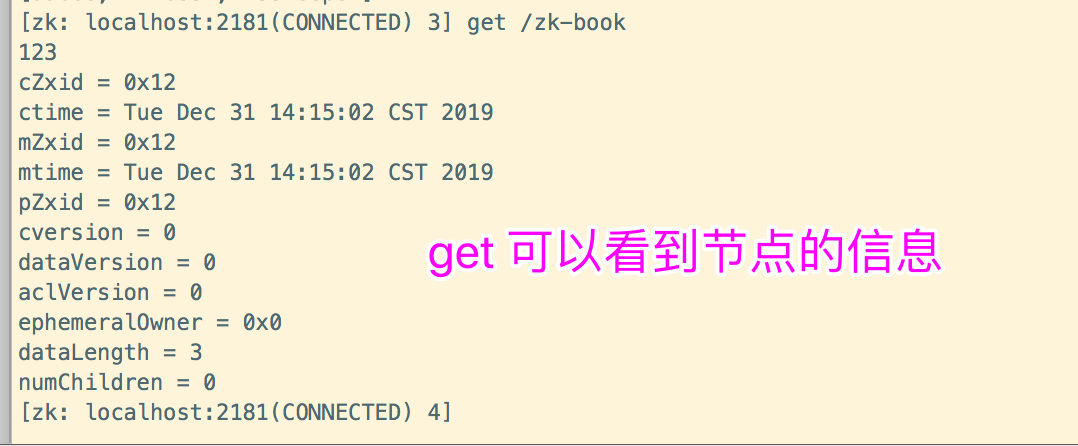
## 5.2 客户端脚本

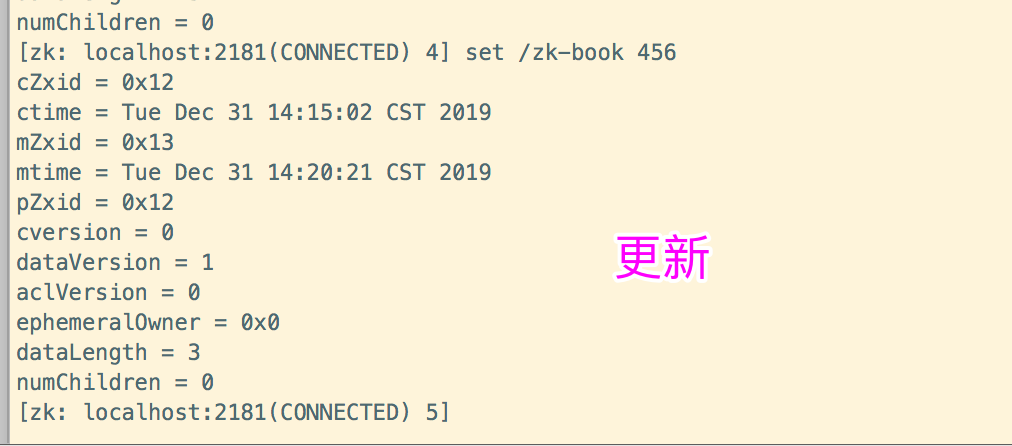


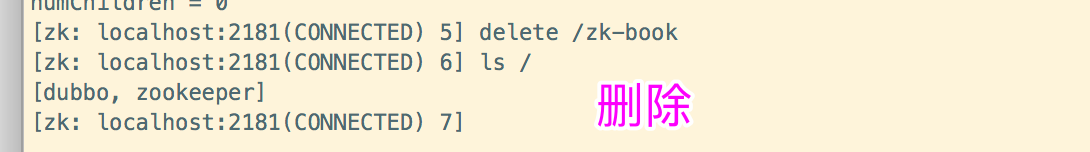
创建：create [-s] [-e] path data acl

-s或-e分别制定节点特性：顺序或临时节点。默认情况不加-s或-e参数的，创建的是持久节点。







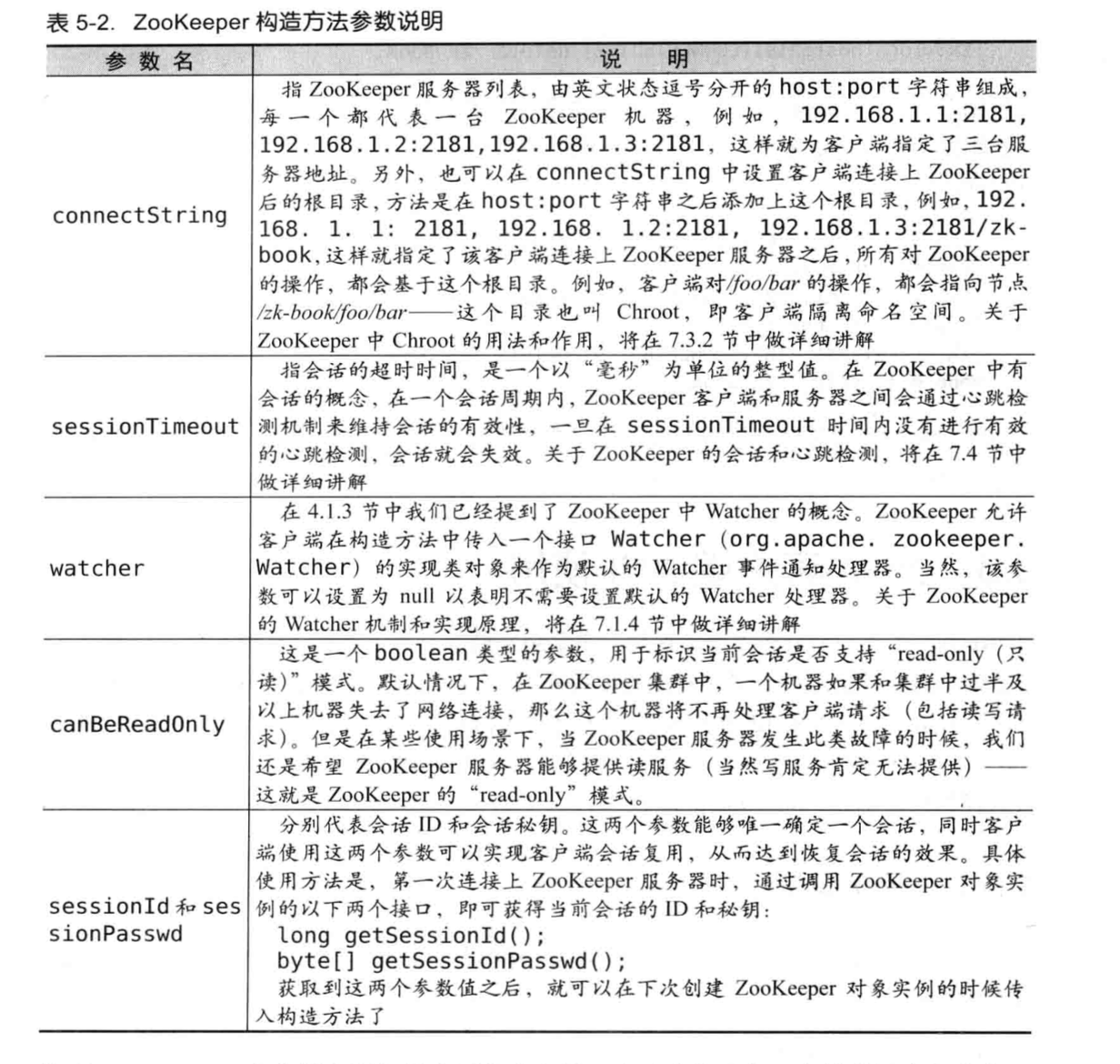


## 5.3 java客户端api

### 5.3.1 创建会话

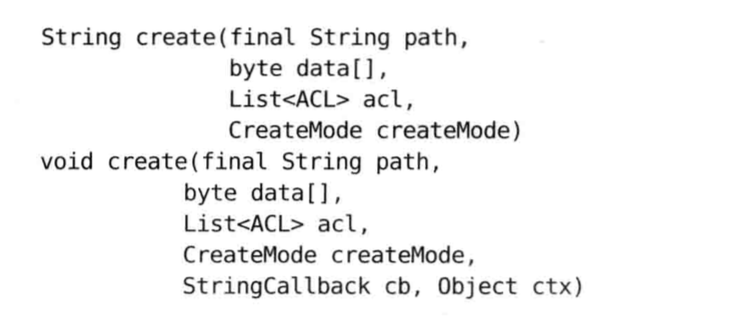
zookeeper构造方法

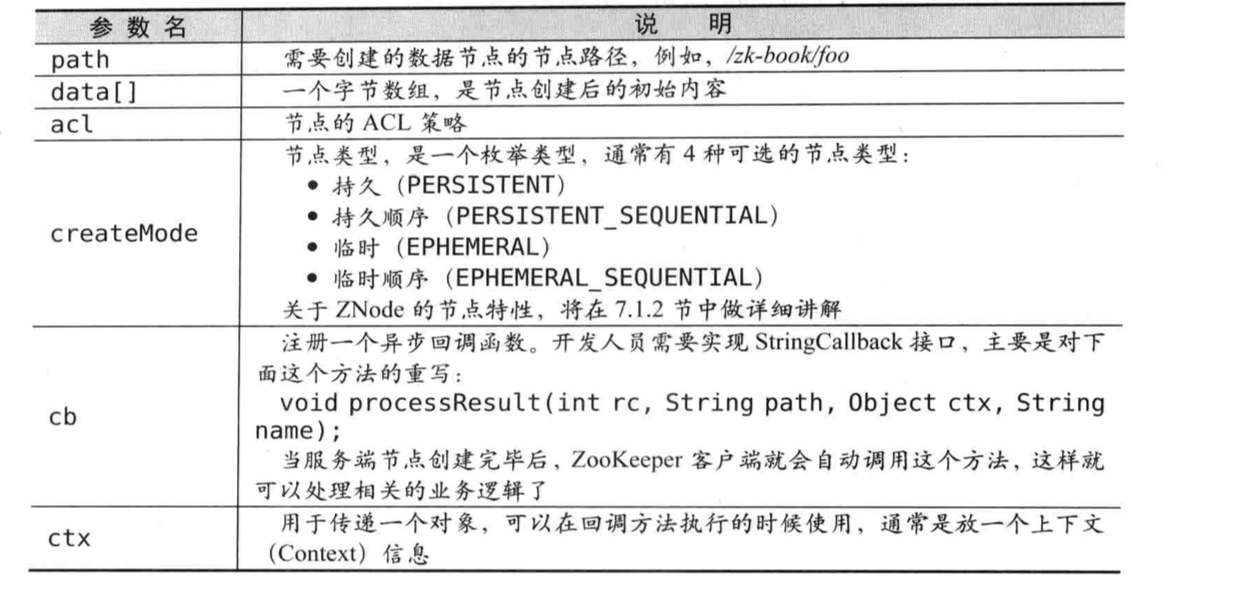
|  |
| --- |
| public ZooKeeper(String connectString, int sessionTimeout, Watcher watcher) throws IOException {  this(connectString, sessionTimeout, watcher, false); }  public ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, boolean canBeReadOnly) throws IOException {  this.watchManager = new ZooKeeper.ZKWatchManager();  LOG.info("Initiating client connection, connectString=" + connectString + " sessionTimeout=" + sessionTimeout + " watcher=" + watcher);  this.watchManager.defaultWatcher = watcher;  ConnectStringParser connectStringParser = new ConnectStringParser(connectString);  HostProvider hostProvider = new StaticHostProvider(connectStringParser.getServerAddresses());  this.cnxn = new ClientCnxn(connectStringParser.getChrootPath(), hostProvider, sessionTimeout, this, this.watchManager, getClientCnxnSocket(), canBeReadOnly);  this.cnxn.start(); }  public ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd) throws IOException {  this(connectString, sessionTimeout, watcher, sessionId, sessionPasswd, false); }  public ZooKeeper(String connectString, int sessionTimeout, Watcher watcher, long sessionId, byte[] sessionPasswd, boolean canBeReadOnly) throws IOException {  this.watchManager = new ZooKeeper.ZKWatchManager();  LOG.info("Initiating client connection, connectString=" + connectString + " sessionTimeout=" + sessionTimeout + " watcher=" + watcher + " sessionId=" + Long.toHexString(sessionId) + " sessionPasswd=" + (sessionPasswd == null ? "<null>" : "<hidden>"));  this.watchManager.defaultWatcher = watcher;  ConnectStringParser connectStringParser = new ConnectStringParser(connectString);  HostProvider hostProvider = new StaticHostProvider(connectStringParser.getServerAddresses());  this.cnxn = new ClientCnxn(connectStringParser.getChrootPath(), hostProvider, sessionTimeout, this, this.watchManager, getClientCnxnSocket(), sessionId, sessionPasswd, canBeReadOnly);  this.cnxn.seenRwServerBefore = true;  this.cnxn.start(); } |



|  |
| --- |
| package com.zookeeper.study;  import org.apache.zookeeper.WatchedEvent; import org.apache.zookeeper.Watcher; import org.apache.zookeeper.ZooKeeper;  import java.io.IOException; import java.util.concurrent.CountDownLatch;  public class ZookeeperConstructorUsageSimple implements Watcher {    private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);    @Override  public void process(WatchedEvent watchedEvent) {  System.*out*.println("Receive watched event: "+watchedEvent);  // 接收服务端发来的SyncConnected事件，解除主程序在CountDownLatch上等待阻塞。  if(Event.KeeperState.*SyncConnected* == watchedEvent.getState()){  System.*out*.println("123");  *connectedSemaphore*.countDown();  }  }   public static void main(String[] args) throws IOException {   ZooKeeper zooKeeper = new ZooKeeper("172.16.144.145:2181", 5000, new ZookeeperConstructorUsageSimple());   System.*out*.println(zooKeeper.getState());   try {  *connectedSemaphore*.await();  }catch (InterruptedException e){  System.*out*.println("Zookeeper session established. ");  }    } } |

### 5.3.2 创建节点



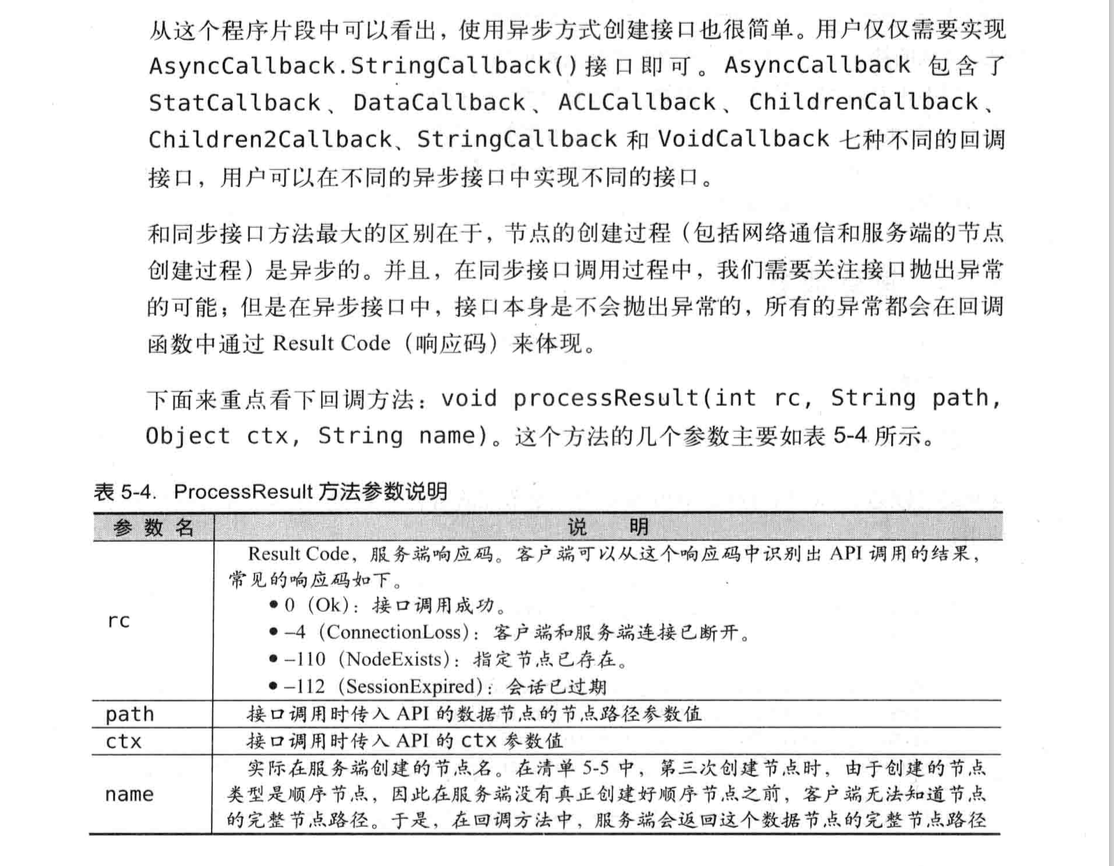


创建临时节点

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_2;  import com.zookeeper.study.chapter05.$5\_3\_1.ZookeeperConstructorUsageWithSIDPASSWD; import org.apache.zookeeper.\*;  import java.io.IOException; import java.util.concurrent.CountDownLatch;  public class ZookeeperCreateAPISyncUsage implements Watcher {   private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);   @Override  public void process(WatchedEvent event) {  if (Event.KeeperState.*SyncConnected* == event.getState()) {  *connectedSemaphore*.countDown();  }  }   public static void main(String[] args) throws Exception {    ZooKeeper zooKeeper = new ZooKeeper("172.16.144.145:2181", 5000, new ZookeeperCreateAPISyncUsage());   *connectedSemaphore*.await();    // 临时节点  String path1 = zooKeeper.create("/zk-test-ephemeral-", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);  System.*out*.println("Success create znode: " + path1);   // 临时顺序节点  String path2 = zooKeeper.create("/zk-test-ephemeral-", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL\_SEQUENTIAL*);  System.*out*.println("Success create znode: " + path2);   } } |

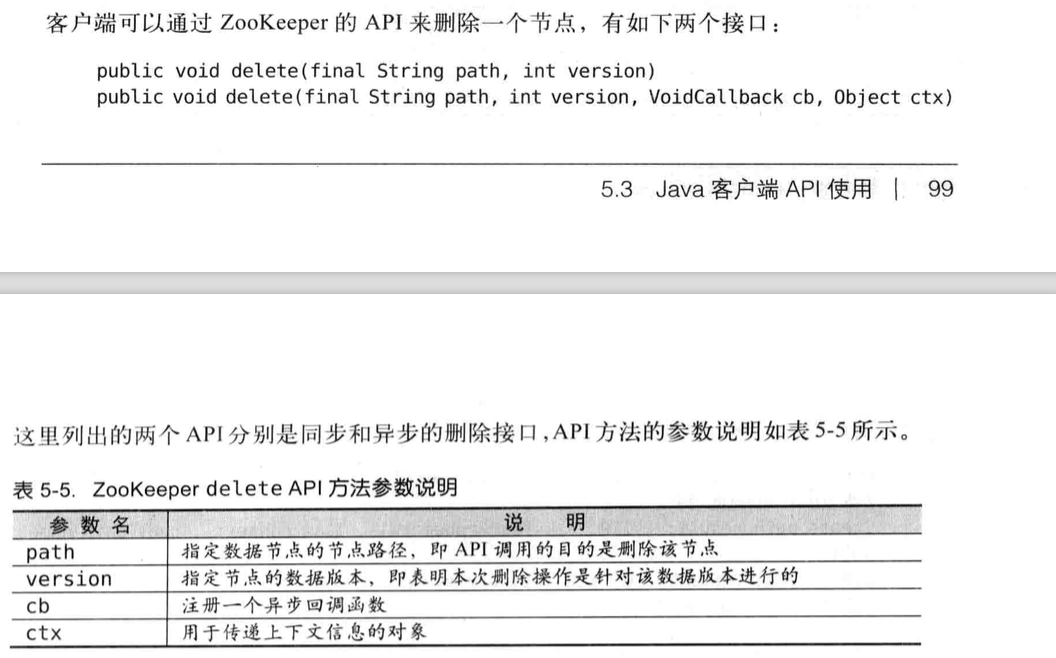
异步创建临时节点

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_2;  import org.apache.zookeeper.\*;  import java.util.concurrent.CountDownLatch;  public class ZookeeperCreateAPIASyncUsage implements Watcher {   private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);   @Override  public void process(WatchedEvent event) {  if (Watcher.Event.KeeperState.*SyncConnected* == event.getState()) {  *connectedSemaphore*.countDown();  }  }   public static void main(String[] args) throws Exception {    ZooKeeper zooKeeper = new ZooKeeper("172.16.144.145:2181", 5000, new ZookeeperCreateAPIASyncUsage());   *connectedSemaphore*.await();    // 临时节点  zooKeeper.create("/zk-test-ephemeral-", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*, new IStringCallback(),"I am context.");  zooKeeper.create("/zk-test-ephemeral-", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*, new IStringCallback(),"I am context.");  zooKeeper.create("/zk-test-ephemeral-", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL\_SEQUENTIAL*, new IStringCallback(),"I am context.");   Thread.*sleep*(Integer.*MAX\_VALUE*);  }    static class IStringCallback implements AsyncCallback.StringCallback {   @Override  public void processResult(int rc, String path, Object ctx, String name) {  System.*out*.println("Create path result: [" + rc + ", " + path + ", "  + ctx + ", real path name: " + name);  }  } } |

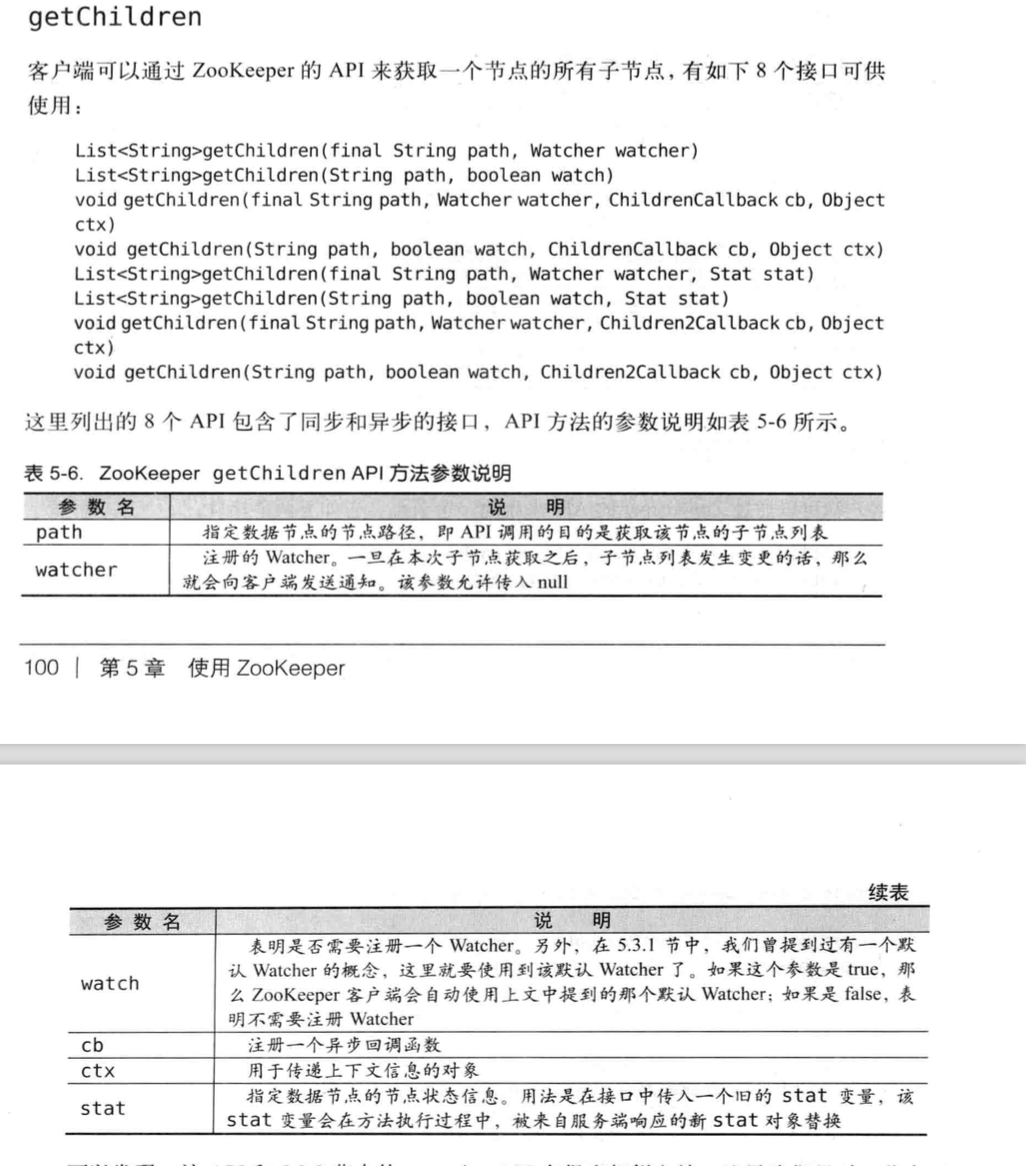


### 5.3.3 删除节点

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_3;  import com.zookeeper.study.chapter05.$5\_3\_2.ZookeeperCreateAPISyncUsage; import org.apache.zookeeper.\*;  import java.io.IOException; import java.util.concurrent.CountDownLatch;  public class DeleteAPISyncUsage implements Watcher {   private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk*;   @Override  public void process(WatchedEvent event) {  if (Event.KeeperState.*SyncConnected* == event.getState()) {  *connectedSemaphore*.countDown();  }  }   public static void main(String[] args) throws Exception {   String path = "/zk-book";   *zk* = new ZooKeeper("172.16.144.145:2181", 5000, new DeleteAPISyncUsage());   *connectedSemaphore*.await();    *zk*.create(path, "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);  *zk*.delete(path, -1);   Thread.*sleep*(Integer.*MAX\_VALUE*);    }   } |



### 5.3.4 读取数据

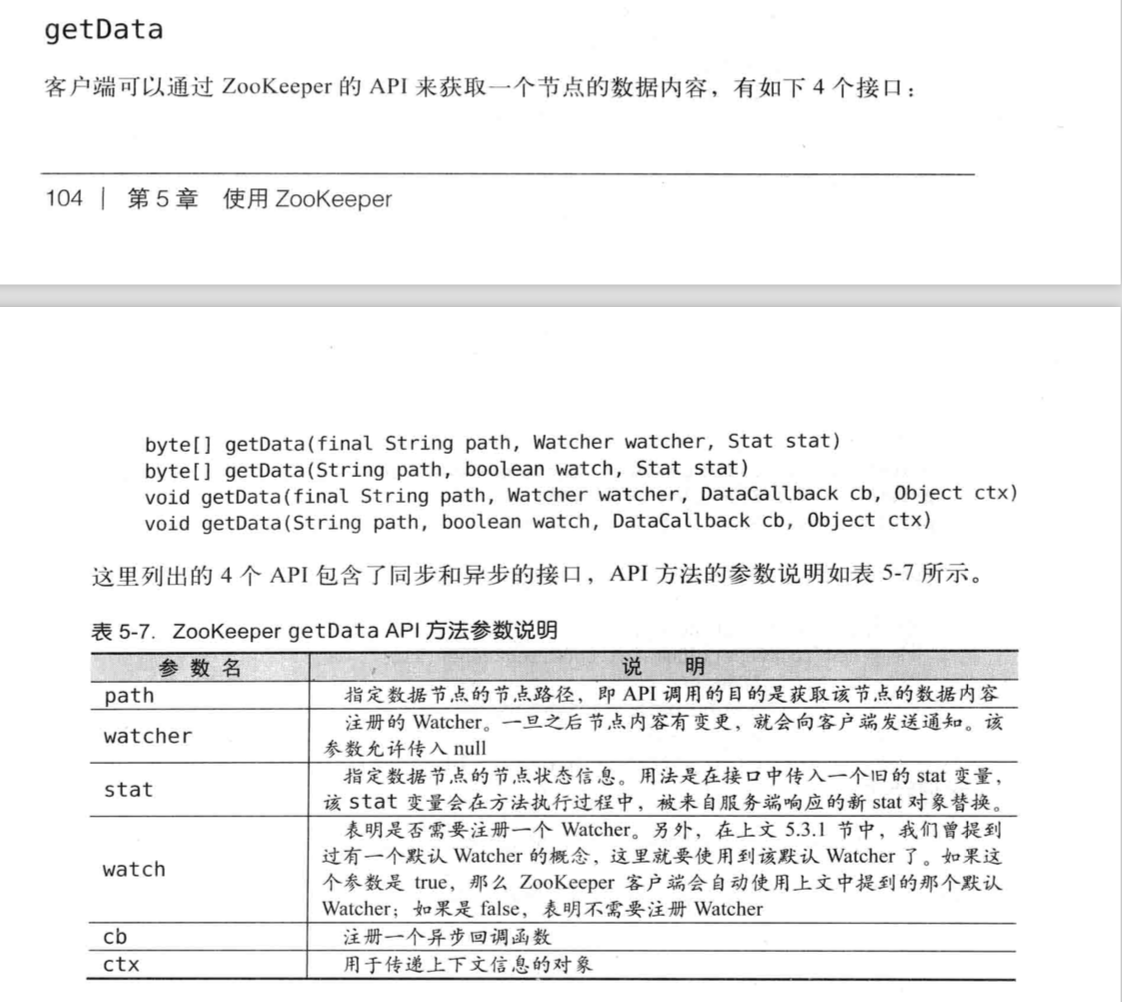


获取子结点列表，并且更新后，发丝不过通知

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_4;  import com.zookeeper.study.chapter05.$5\_3\_3.DeleteAPISyncUsage; import org.apache.zookeeper.\*;  import java.io.IOException; import java.util.List; import java.util.concurrent.CountDownLatch;  public class ZookeeperGetChildrenAPISyncUsage implements Watcher {  private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk* = null;   @Override  public void process(WatchedEvent event) {  if (Event.KeeperState.*SyncConnected* == event.getState()) {  if (Event.EventType.*None* == event.getType() && null == event.getPath()) {  *connectedSemaphore*.countDown();  } else if (event.getType() == Event.EventType.*NodeChildrenChanged*) {  try {  System.*out*.println("ReGet Child: " + *zk*.getChildren(event.getPath(), true));  } catch (Exception e) {  }  }  }  }    public static void main(String[] args) throws Exception {  String path = "/zk-book";   *zk* = new ZooKeeper("172.16.144.145:2181", 5000, new ZookeeperGetChildrenAPISyncUsage());   *connectedSemaphore*.await();    *zk*.create(path, "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT*);  *zk*.create(path + "/c1", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);   List<String> children = *zk*.getChildren(path, true);  System.*out*.println(children);   *zk*.create(path + "/c2", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);   Thread.*sleep*(Integer.*MAX\_VALUE*);    } } |

异步回调的方式获得

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_4;  import org.apache.zookeeper.\*; import org.apache.zookeeper.data.Stat;  import java.io.IOException; import java.util.List; import java.util.concurrent.CountDownLatch;  public class ZookeeperGetChildrenAPIASyncUsage implements Watcher {   private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk* = null;    public static void main(String[] args) throws Exception {   String path = "/zk-book";   *zk* = new ZooKeeper("172.16.144.145:2181", 5000, new ZookeeperGetChildrenAPIASyncUsage());   *connectedSemaphore*.await();    *zk*.create(path, "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT*);  *zk*.create(path + "/c1", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);   *zk*.getChildren(path, true, new IChildren2Callback(), null);  *zk*.create(path + "/c2", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);   Thread.*sleep*(Integer.*MAX\_VALUE*);    }    @Override  public void process(WatchedEvent event) {  if (Event.KeeperState.*SyncConnected* == event.getState()) {  if (Event.EventType.*None* == event.getType() && null == event.getPath()) {  *connectedSemaphore*.countDown();  } else if (event.getType() == Event.EventType.*NodeChildrenChanged*) {  try {  System.*out*.println("ReGet Child: " + *zk*.getChildren(event.getPath(), true));  } catch (Exception e) {  }  }  }  }   static class IChildren2Callback implements AsyncCallback.Children2Callback {   @Override  public void processResult(int rc, String path, Object ctx, List<String> children, Stat stat) {  System.*out*.println("Get Children znode result: [response code: " + rc + ", param path: " + path  + ", ctx: " + ctx + ", children list: " + children + ", stat: " + stat);  }   } } |



同步方式获得data

节点的数据内容和版本任何一个变化都是变化的

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_4;  import org.apache.zookeeper.\*; import org.apache.zookeeper.data.Stat;  import java.io.IOException; import java.util.concurrent.CountDownLatch;  public class GetDataAPISyncUsage implements Watcher {    private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk* = null;  private static Stat *stat* = new Stat();    @Override  public void process(WatchedEvent event) {   if (Event.KeeperState.*SyncConnected* == event.getState()) {  if (Event.EventType.*None* == event.getType() && null == event.getPath()) {  *connectedSemaphore*.countDown();  } else if (event.getType() == Event.EventType.*NodeDataChanged*) {  try {  byte[] data = *zk*.getData(event.getPath(), true, *stat*);  System.*out*.println(new String(data));   System.*out*.println(*stat*.getCzxid() + ", " + *stat*.getMzxid() + ", " + *stat*.getVersion());  } catch (Exception e) {  }  }  }  }    public static void main(String[] args) throws Exception {  String path = "/zk-book";  *zk* = new ZooKeeper("172.16.144.145:2181", 5000, new GetDataAPISyncUsage());  *connectedSemaphore*.await();   *zk*.create(path, "123".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);   System.*out*.println(new String(*zk*.getData(path, true, *stat*)));  System.*out*.println(*stat*.getCzxid() + " ," + *stat*.getMzxid() + ", " + *stat*.getVersion());   *zk*.setData(path, "123".getBytes(), -1);   Thread.*sleep*(Integer.*MAX\_VALUE*);  } } |

异步获取节点内容

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_4;  import org.apache.zookeeper.\*; import org.apache.zookeeper.data.Stat;  import java.util.concurrent.CountDownLatch;  public class GetDataAPIASyncUsage implements Watcher {  private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk* = null;  private static Stat *stat* = new Stat();    @Override  public void process(WatchedEvent event) {   if (Event.KeeperState.*SyncConnected* == event.getState()) {  if (Event.EventType.*None* == event.getType() && null == event.getPath()) {  *connectedSemaphore*.countDown();  } else if (event.getType() == Event.EventType.*NodeDataChanged*) {  try {  *zk*.getData( event.getPath(), true, new IDataCallback(), null );  } catch (Exception e) {  }  }  }  }    public static void main(String[] args) throws Exception {  String path = "/zk-book";  *zk* = new ZooKeeper("172.16.144.145:2181", 5000, new GetDataAPIASyncUsage());  *connectedSemaphore*.await();   *zk*.create(path, "123".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL*);   *zk*.getData(path, true, new IDataCallback(), null);  *zk*.setData(path, "123".getBytes(), -1);   Thread.*sleep*(Integer.*MAX\_VALUE*);  }   static class IDataCallback implements AsyncCallback.DataCallback {   @Override  public void processResult(int rc, String path, Object ctx, byte[] data, Stat stat) {  System.*out*.println(rc + ", " + path + ", " + new String(data));  System.*out*.println(stat.getCzxid()+","+  stat.getMzxid()+","+  stat.getVersion());  }  } } |

### 5.3.5 更新数据



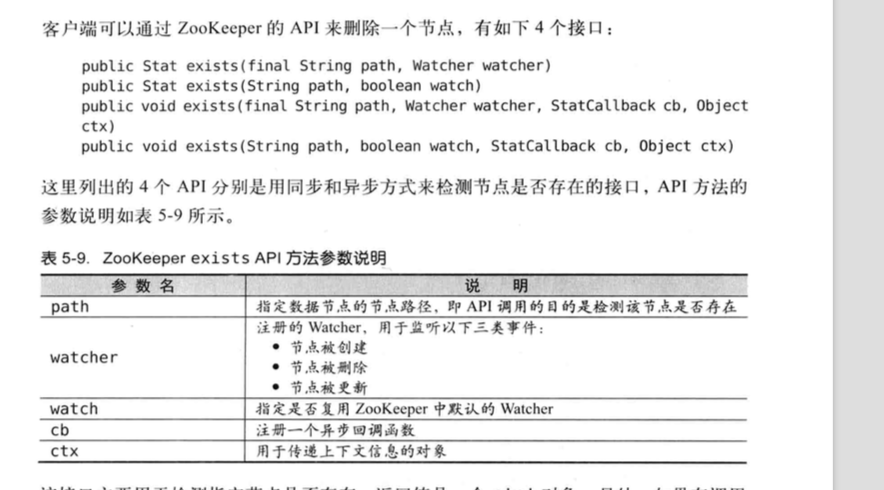
同步方式

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_5;  import org.apache.zookeeper.\*; import org.apache.zookeeper.data.Stat;  import java.util.concurrent.CountDownLatch;  public class SetDataAPISyncUsage implements Watcher {   private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk*;   @Override  public void process(WatchedEvent event) {  if (Event.KeeperState.*SyncConnected* == event.getState()) {  if (Event.EventType.*None* == event.getType() && null == event.getPath()) {  *connectedSemaphore*.countDown();  }  }  }    public static void main(String[] args) throws Exception {   String path = "/zk-book";  *zk* = new ZooKeeper("172.16.144.145:2181",  5000, //  new SetDataAPISyncUsage());  *connectedSemaphore*.await();   *zk*.create( path, "123".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL* );  *zk*.getData( path, true, null );   // -1 表示以最新版本进行更新操作  Stat stat = *zk*.setData( path, "456".getBytes(), -1 );  System.*out*.println(stat.getCzxid()+","+  stat.getMzxid()+","+  stat.getVersion());  Stat stat2 = *zk*.setData( path, "456".getBytes(), stat.getVersion() );  System.*out*.println(stat2.getCzxid()+","+  stat2.getMzxid()+","+  stat2.getVersion());  try {  *zk*.setData( path, "456".getBytes(), stat.getVersion() );  } catch ( KeeperException e ) {  System.*out*.println("Error: " + e.code() + "," + e.getMessage());  }  Thread.*sleep*( Integer.*MAX\_VALUE* );  }       } |

异步方式

|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_5;  import org.apache.zookeeper.\*; import org.apache.zookeeper.data.Stat;  import java.util.concurrent.CountDownLatch;  public class SetDataAPIASyncUsage implements Watcher {   private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk*;   @Override  public void process(WatchedEvent event) {  if (Event.KeeperState.*SyncConnected* == event.getState()) {  if (Event.EventType.*None* == event.getType() && null == event.getPath()) {  *connectedSemaphore*.countDown();  }  }  }    public static void main(String[] args) throws Exception {   String path = "/zk-book";  *zk* = new ZooKeeper("172.16.144.145:2181",  5000, //  new SetDataAPIASyncUsage());  *connectedSemaphore*.await();   *zk*.create( path, "123".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*EPHEMERAL* );  *zk*.setData( path, "345".getBytes(), -1, new IStatCallback(),null);   Thread.*sleep*( Integer.*MAX\_VALUE* );  }    private static class IStatCallback implements AsyncCallback.StatCallback {  public void processResult(int rc, String path, Object ctx, Stat stat) {  if (rc == 0) {  System.*out*.println("SUCCESS");  }  }  } } |

### 5.3.6 检测节点是否存在



|  |
| --- |
| package com.zookeeper.study.chapter05.$5\_3\_6;  import org.apache.zookeeper.\*;  import java.util.concurrent.CountDownLatch;  public class ExistAPISyncUsage implements Watcher {   private static CountDownLatch *connectedSemaphore* = new CountDownLatch(1);  private static ZooKeeper *zk*;  public static void main(String[] args) throws Exception {   String path = "/zk-book";  *zk* = new ZooKeeper("172.16.144.145:2181",  5000, //  new ExistAPISyncUsage());  *connectedSemaphore*.await();   *zk*.exists( path, true );  *zk*.create( path, "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT* );  *zk*.setData( path, "123".getBytes(), -1 );  *zk*.create( path+"/c1", "".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT* );  *zk*.delete( path+"/c1", -1 );  *zk*.delete( path, -1 );  Thread.*sleep*( Integer.*MAX\_VALUE* );  }   @Override  public void process(WatchedEvent event) {  try {  if (Event.KeeperState.*SyncConnected* == event.getState()) {  if (Event.EventType.*None* == event.getType() && null == event.getPath()) {  *connectedSemaphore*.countDown();  } else if (Event.EventType.*NodeCreated* == event.getType()) {  System.*out*.println("Node(" + event.getPath() + ")Created");  *zk*.exists( event.getPath(), true );  } else if (Event.EventType.*NodeDeleted* == event.getType()) {  System.*out*.println("Node(" + event.getPath() + ")Deleted");  *zk*.exists( event.getPath(), true );  } else if (Event.EventType.*NodeDataChanged* == event.getType()) {  System.*out*.println("Node(" + event.getPath() + ")DataChanged");  *zk*.exists( event.getPath(), true );  }  }  } catch (Exception e) {}  } } |

### 5.3.7 权限控制

zookeeper提供了多种权限控制模式：world，auth，dugest,，ip和supper

# Zookeeper的典型应用场景

## 6.1 典型用用场景及实现

### 6.1.1 数据发布/订阅

数据发布/订阅(Publish/Subscribe)系统。即所谓的配置中心。就是发布者将数据发布到Zookeeper的一个或一系列节点上，供订阅者进行数据订阅。进而达到动态获取数据的目的，实现配置信息的集中式管理和数据的动态更新。

发布订阅一般两种设计模式，推(push)模式和拉(pull)。

### 6.1.2 负载均衡

基于zookeeper实现的动态DNS方法(简称“DDNS”，Dynamic DNS)

### 6.1.3 命名服务

zookeeper提供的命名功能与JNDI技术有相似的地方，帮助应用系统通过资源引用的方式来实现对资源的定位与使用。

在分布式系统中广泛使用的用于唯一标识元素的标准，最典型的实现是GUID（Globally Unique Identifier，全局唯一标识符）。

### 6.1.4 分布式协调/通知

Zookeeper 中特有的Watcher注册与异步通知机制，能够很好地实现分布式环境下不同机器，甚至是不同系统之间的协调与通知。

**MySQL数据复制总线：Mysql\_Replicator**

**任务注册**

**任务热备份**

**热备切换**

**记录执行状态**

**控制台协调**

**冷备切换**

**冷热备份对比**

**心跳检测**

**工作进度汇报**

**系统调度**

### 6.1.5 集群管理

### 6.1.6 Master选举

### 6.1.7 分布式锁

### 6.1.8 分布式对列