Zookeeper分布式系统开发实战 第4课

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

第四讲 开源客户端---原生api的不足

- 连接的创建是异步的，需要开发人员自行编码实现等待
- 连接没有自动的超时重连机制
- Zk本身不提供序列化机制，需要开发人员自行指定，从而实现数据的序列化和反序列化
- Watcher注册一次只会生效一次，需要不断的重复注册
- Watcher的使用方式不符合java本身的术语，如果采用监听器方式，更容易理解
- 不支持递归创建树形节点

第四讲 开源客户端---ZkClient介绍

- Github上一个开源的zk客户端，由datameer的工程师Stefan Groschupf和Peter Voss一起开发
  - 解决session会话超时重连
  - Watcher反复注册
  - 简化开发api
  - 还有。。。。。
  - https://github.com/sgroschupf/zkclient

```
<dependency>
    <groupId>com.101tec</groupId>
    <artifactId>zkclient</artifactId>
    <version>0.4</version>
</dependency>
```

第四讲 开源客户端---ZkClient介绍

- 特点
  - 简单
  - 社区不活跃，连api文档都不完善

```
public class ZkclientTest {
    private ZkClient zkclient = null;
    public ZkclientTest(){
        this.zkclient = new ZkClient("localhost:2181,localhost:2182,localhost:2183",5000);
    }

    public void createPersistentNode(String path,Object data){
        //zkclient.createPersistent(path, data);
        zkclient.create(path, data, CreateMode.PERSISTENT);
    }
    public static void main(String[] args) {
        ZkclientTest zt = new ZkclientTest();
        zt.createPersistentNode("/zkclient/node1", "zkclientnode1");

    }
```

第四讲 开源客户端---Curator介绍

- Apache基金会的顶级项目之一
  - 解决session会话超时重连
  - Watcher反复注册
  - 简化开发api
  - 遵循Fluent风格Api规范
  - NodeExistsException异常处理
  - 大招：共享锁服务 master选举 分布式计数器等
  - 还有。。。。。
  - http://curator.apache.org/

第四讲 开源客户端---Curator介绍

- 创建会话
  1. 使用CuratorFrameworkFactory工厂的两个静态方法创建客户端
     a) static CuratorFramework newClient(String connectString, int sessionTimeoutMs, int connectionTimeoutMs, RetryPolicy retryPolicy)
     b) static CuratorFramework newClient(String connectString, RetryPolicy retryPolicy)
  2. Start()方法启动

参数名	说明
connectString	逗号分开的ip：port对
retryPolicy	重试策略，默认四种：Exponential BackoffRetry 、RetryNTimes 、RetryOneTime、RetryUntilElapsed
sessionTimeoutMs	会话超时时间，单位为毫秒，默认60000ms
connectionTimeoutMs	连接创建超时时间，单位为毫秒，默认是15000ms

第四讲 开源客户端--- Curator介绍

- 重试策略
  - 实现接口RetryPolicy可以自定义重试策略
    - boolean allowRetry(int retryCount, long elapsedTimeMs, RetrySleeper sleeper)

参数名	说明
retryCount	已经重试的次数，如果第一次重试，此值为0
elapsedTimeMs	重试花费的时间，单位为毫秒
sleeper	类似于Thread.sleep,用于sleep指定时间
返回值	如果还会继续重试，则返回true

  - 四种默认重试策略

第四讲 开源客户端--- Curator介绍

- 默认重试策略
  - ExponentialBackoffRetry
    - ExponentialBackoffRetry(int baseSleepTimeMs, int maxRetries)
    - ExponentialBackoffRetry(int baseSleepTimeMs, int maxRetries, int maxSleepMs)
    - 当前应该sleep的时间： baseSleepTimeMs * Math.max(1, random.nextInt(1 << (retryCount + 1)))

参数名	说明
baseSleepTimeMs	初始sleep时间
maxRetries	最大重试次数
maxSleepMs	最大重试时间
返回值	如果还会继续重试，则返回true

第四讲 开源客户端--- Curator介绍

- 默认重试策略
  - RetryNTimes
    - RetryNTimes(**int n, int sleepMsBetweenRetries)**
    - 当前应该sleep的时间：

参数名	说明
n	最大重试次数
sleepMsBetweenRetries	每次重试的间隔时间

  - RetryOneTime
    - 只重试一次
    - **RetryOneTime**(int sleepMsBetweenRetry)， sleepMsBetweenRetry为重试间隔的时间

第四讲 开源客户端--- Curator介绍

- 默认重试策略
  - RetryUntilElapsed
    - RetryUntilElapsed(int maxElapsedTimeMs, int sleepMsBetweenRetries)
    - 重试的时间超过最大时间后，就不再重试

参数名	说明
maxElapsedTimeMs	最大重试时间
sleepMsBetweenRetries	每次重试的间隔时间

第四讲 开源客户端--- Curator介绍

- Fluent风格的API
  - 定义：一种面向对象的开发方式，目的是提高代码的可读性
  - 实现方式：通过方法的级联或者方法链的方式实现
  - 举例：

```
CuratorFramework client = CuratorFrameworkFactory.builder()
        .connectString("localhost:2181,localhost:2182")
        .sessionTimeoutMs(10000).retryPolicy(retryPolicy)
        .namespace("base").build();
```

第四讲 开源客户端--- Curator介绍

- 创建节点
  - 构建操作包装类（Builder）： CreateBuilder　create()---- CuratorFramework
  - CreateBuilder
    - creatingParentsIfNeeded() //递归创建父目录
    - withMode（ CreateMode mode）//设置节点属性，比如： CreateMode.*PERSISTENT*，如果是递归创建，创建模式为临时节点，则只有叶子节点是临时节点，非叶子节点都为持久节点
    - withACL(List aclList) //设置acl
    - forPath(String path) //指定路径

第四讲 开源客户端--- Curator介绍

```java
public class CuratorClientTest {
    private CuratorFramework client = null;
    public CuratorClientTest() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
                .connectString("localhost:2181,localhost:2182")
                .sessionTimeoutMs(10000).retryPolicy(retryPolicy)
                .namespace("base").build();
        client.start();
    }
    public void closeClient() {
        if(client!=null)
            this.client.close();
    }
    public void createNode(String path, byte[] data) throws Exception {
        client.create().creatingParentsIfNeeded()
                .withMode(CreateMode.PERSISTENT).withACL(Ids.OPEN_ACL_UNSAFE)
                .forPath(path, data);
    }
    public static void main(String[] args) {
        CuratorClientTest ct = null;
        try {
            ct = new CuratorClientTest();
            ct.createNode("/curator/test/node1", "test-node1".getBytes());
            Thread.sleep(30000);
        } catch (Exception e) {
        } finally {
            ct.closeClient();
        }
```

第四讲 开源客户端--- Curator介绍

- 删除节点
  - 构建操作包装类（Builder）： DeleteBuilder　delete() -----CuratorFramework
  - DeleteBuilder
    - withVersion(int version) //特定版本号
    - guaranteed() //确保节点被删除
    - forPath(String path) //指定路径
    - deletingChildrenIfNeeded() //递归删除所有子节点

关于guaranteed：

Solves edge cases where an operation may succeed on the server but connection failure occurs before a response can be successfully returned to the client

意思是：解决当某个删除操作在服务器端可能成功，但是此时客户端与服务器端的连接中断，而删除的响应没有成功返回到客户端

底层的本质：重试

第四讲 开源客户端--- Curator介绍

```
public Void forPath(String path) throws Exception
{
    final String unfixedPath = path;
    path = client.fixForNamespace(path);

    if ( backgrounding.inBackground() )
    {
        OperationAndData.ErrorCallback<String> errorCallback = null;
        if ( guaranteed )
        {
            errorCallback = new OperationAndData.ErrorCallback<String>()
            {
                @Override
                public void retriesExhausted(OperationAndData<String> operationAndData)
                {
                    client.getFailedDeleteManager().addFailedDelete(unfixedPath);
                }
            };
        }
        client.processBackgroundOperation(new OperationAndData<String>(this, path, backgrounding.getCallb
    }
```

删除失败的集合

第四讲 开源客户端--- Curator介绍

```
void addFailedDelete(String path)
{
    if ( debugListener != null )
    {
        debugListener.pathAddedForDelete(path);
    }

    if ( client.getState() == CuratorFrameworkState.STARTED )
    {
        log.debug("Path being added to guaranteed delete set: " + path);
        try
        {
            client.delete().guaranteed().inBackground().forPath(path);
        }
        catch ( Exception e )
        {
            addFailedDelete(path);
        }
    }
}
```

客户端状态属于启动状态

再次执行删除

第四讲 开源客户端--- Curator介绍

- 关于异步操作
  - inBackground()
  - inBackground(Object context)
  - inBackground(BackgroundCallback callback)
  - inBackground(BackgroundCallback callback, Object context)
  - inBackground(BackgroundCallback callback, Executor executor)
  - inBackground(BackgroundCallback callback, Object context, Executor executor)

从参数看跟zk的原生异步api相同，多了一个线程池，用于执行回调

第四讲 开源客户端--- Curator介绍

- 异步操作回调

```
client.delete().guaranteed().withVersion(version).inBackground(new DeleteCallBack()).forPath(path);
```

```
public class DeleteCallBack implements BackgroundCallback {

    public void processResult(CuratorFramework client, CuratorEvent event)
            throws Exception {
        // TODO Auto-generated method stub
        System.out.println(event.getPath()+",data="+event.getData());
    }
```

第四讲 开源客户端--- Curator介绍

- 异步操作事件状态：event.getType()

```
public enum CuratorEventType
{
    /**
     * Corresponds to {@link CuratorFramework#create()}
     */
    CREATE,

    /**
     * Corresponds to {@link CuratorFramework#delete()}
     */
    DELETE,

    /**
     * Corresponds to {@link CuratorFramework#checkExists()}
     */
    EXISTS,

    /**
     * Corresponds to {@link CuratorFramework#getData()}
     */
    GET_DATA,

    /**
     * Corresponds to {@link CuratorFramework#setData()}
     */
    SET_DATA,

    /**
     * Corresponds to {@link CuratorFramework#getChildren()}
     */
    CHILDREN,

    /**
     * Corresponds to {@link CuratorFramework#sync(String, Object)}
     */
    SYNC,

    /**
     * Corresponds to {@link CuratorFramework#getACL()}
     */
    GET_ACL,

    /**
     * Corresponds to {@link CuratorFramework#setACL()}
     */
    SET_ACL,

    /**
     * Corresponds to {@link Watchable#usingWatcher(Watcher)} or {@link Watchable#watched()}
     */
    WATCHED,

    /**
     * Event sent when client is being closed
     */
    CLOSING
```

第四讲 开源客户端--- Curator介绍

- 异步操作事件状态码：event.getResultCode()
  - org.apache.zookeeper. KeeperException. Code

```
public static enum Code implements CodeDeprecated {
    /** Everything is OK */
    OK (Ok),

    /** System and server-side errors.
     * This is never thrown by the server, it shouldn't be used other than
     * to indicate a range. Specifically error codes greater than this
     * value, but lesser than {@link #APIERROR}, are system errors.
     */
    SYSTEMERROR (SystemError),

    /** A runtime inconsistency was found */
    RUNTIMEINCONSISTENCY (RuntimeInconsistency),
    /** A data inconsistency was found */
    DATAINCONSISTENCY (DataInconsistency),
    /** Connection to the server has been lost */
    CONNECTIONLOSS (ConnectionLoss),
    /** Error while marshalling or unmarshalling data */
    MARSHALLINGERROR (MarshallingError),
    /** Operation is unimplemented */
    UNIMPLEMENTED (Unimplemented),
    /** Operation timeout */
    OPERATIONTIMEOUT (OperationTimeout),
    /** Invalid arguments */
    BADARGUMENTS (BadArguments),

    /** API errors.
     * This is never thrown by the server, it shouldn't be used other than
     * to indicate a range. Specifically error codes greater than this
     * value are API errors (while values less than this indicate a
     * {@link #SYSTEMERROR}).
     */
    APIERROR (APIError),

    /** Node does not exist */
    NONODE (NoNode),
    /** Not authenticated */
    NOAUTH (NoAuth),
    /** Version conflict */
    BADVERSION (BadVersion),
    /** Ephemeral nodes may not have children */
    NOCHILDRENFOREPHEMERALS (NoChildrenForEphemerals),
    /** The node already exists */
    NODEEXISTS (NodeExists),
    /** The node has children */
    NOTEMPTY (NotEmpty),
    /** The session has been expired by the server */
    SESSIONEXPIRED (SessionExpired),
    /** Invalid callback specified */
    INVALIDCALLBACK (InvalidCallback),
    /** Invalid ACL specified */
    INVALIDACL (InvalidACL),
    /** Client authentication failed */
    AUTHFAILED (AuthFailed),
    /** Session moved to another server, so operation is ignored */
    SESSIONMOVED (-118),
```

第四讲 开源客户端--- Curator介绍

- 异步操作事件状态码：event.getResultCode()
  - org.apache.zookeeper. KeeperException. Code

```
public void processResult(CuratorFramework client, CuratorEvent event)
        throws Exception {
    // TODO Auto-generated method stub
    System.out.println(event.getPath()+",data="+event.getData());
    System.out.println("event type="+event.getType());
    System.out.println("event code="+event.getResultCode());

}
```

```
/curator,data=null
event type=DELETE
event code=-111
```

-111表示有子节点，所以删除失败，只有为0（ok）时表示删除成功

```
client.delete().guaranteed().withVersion(version).inBackground(new DeleteCallBack()).forPath(path);
```

第四讲 开源客户端--- Curator介绍

- 读取数据
  - 构建操作包装类（Builder）： GetDataBuilder getData() -----CuratorFramework
  - GetDataBuilder
    - storingStatIn(org.apache.zookeeper.data.Stat stat) //把服务器端获取的状态数据存储到stat对象
    - Byte[] forPath（String path）//节点路径

```
public void readNode(String path) throws Exception{
    Stat stat = new Stat();
    byte[] data = client.getData().storingStatIn(stat).forPath(path);
    System.out.println("读取节点"+path+"的数据:"+new String(data));
    System.out.println(stat.toString());
}
```

第四讲 开源客户端--- Curator介绍

- 更新数据
  - 构建操作包装类（Builder） SetDataBuilder　　setData() -----CuratorFramework
  - SetDataBuilder
    - withVersion(int version) //特定版本号
    - forPath（String path，byte[] data）//节点路径
    - forPath（String path）//节点路径

```
public void updateNode(String path,byte[] data,int version) throws Exception{
    client.setData().withVersion(version).forPath(path, data);
}
```

第四讲 开源客户端--- Curator介绍

- 读取子节点
  - 构建操作包装类（Builder）： GetChildrenBuilder　　getChildren() -----CuratorFramework
  - GetChildrenBuilder
    - storingStatIn(org.apache.zookeeper.data.Stat stat) //把服务器端获取的状态数据存储到stat对象
    - Byte[] forPath（String path）//节点路径
    - usingWatcher(org.apache.zookeeper.Watcher watcher) //设置watcher，类似于zk本身的api，也只能使用一次
    - usingWatcher(CuratorWatcher watcher) //设置watcher ，类似于zk本身的api，也只能使用一次

```
public void getChildren(String path) throws Exception{
    List<String> children = client.getChildren().forPath("/curator");

    for(String pth : children){
        System.out.println("child="+pth);
    }
}
```

第四讲 开源客户端--- Curator介绍

- 设置watcher
  - NodeCache
    - 监听数据节点的内容变更
    - 监听节点的创建，即如果指定的节点不存在，则节点创建后，会触发这个监听
  - PathChildrenCache
    - 监听指定节点的子节点变化情况
    - 包括：新增子节点 子节点数据变更 和子节点删除

第四讲 开源客户端--- Curator介绍

- NodeCache
  - 构造函数
    - NodeCache(CuratorFramework client, String path)
    - NodeCache(CuratorFramework client, String path, boolean dataIsCompressed)

参数名	说明
client	客户端实例
path	数据节点路径
dataIsCompressed	是否进行数据压缩

  - 回调接口
    - public interface NodeCacheListener

      void nodeChanged() //没有参数，怎么获取事件信息以及节点数据？

第四讲 开源客户端--- Curator介绍

Modifier and Type	Method and Description
void	close()
ChildData	getCurrentData() Return the current data.
ListenerContainer<NodeCacheListener>	getListenable() Return the cache listenable
protected void	handleException(Throwable e) Default behavior is just to log the exception
void	rebuild() NOTE: this is a BLOCKING method.
void	start() Start the cache.
void	start(boolean buildInitial) Same as start() but gives the option of doing an initial build

Modifier and Type	Method and Description
int	compareTo(ChildData rhs) Note: this class has a natural ordering that is inconsistent with equals.
boolean	equals(Object o)
byte[]	getData() Returns the node data for this child when the cache mode is set to cache data.
String	getPath() Returns the full path of the this child
org.apache.zookeeper.data.Stat	getStat() Returns the stat data for this child
int	hashCode()
String	toString()

第四讲 开源客户端--- Curator介绍

```
public void addWatcher(String path) throws Exception{
    final NodeCache nodeC = new NodeCache(client,path);
    nodeC.start(true);
    nodeC.getListenable().addListener(new NodeCacheListener(){
        public void nodeChanged() throws Exception {
            String data = new String(nodeC.getCurrentData().getData());
            System.out.println("path="+nodeC.getCurrentData().getPath()+":data="+data);
        }
    });
```

```
public static void main(String[] args) {
    CuratorClientTest ct = null;
    try {
        ct = new CuratorClientTest();
        //ct.createNode("/curator/test/node1", "test-node1".getBytes());
        ct.readNode("/curator/test/node1");
        ct.getChildren("/curator");
        //ct.updateNode("/curator/test/node1", "test-node1-new".getBytes(), 0);
        //ct.readNode("/curator/test/node1");
        //ct.deleteNode("/curator", 0);

        ct.addWatcher("/curator");
        Thread.sleep(300000);
    } catch (Exception e) {
    } finally {
        ct.closeClient();
    }
```

第四讲 开源客户端--- Curator介绍

- PathChildrenCache
  - 构造函数
    - PathChildrenCache(CuratorFramework client, String path, boolean cacheData)
    - PathChildrenCache(CuratorFramework client, String path, boolean cacheData, boolean dataIsCompressed, CloseableExecutorService executorService)
    - PathChildrenCache(CuratorFramework client, String path, boolean cacheData, boolean dataIsCompressed, ExecutorService executorService)
    - PathChildrenCache(CuratorFramework client, String path, boolean cacheData, boolean dataIsCompressed, ThreadFactory threadFactory)
    - PathChildrenCache(CuratorFramework client, String path, boolean cacheData, ThreadFactory threadFactory)
  - 回调接口
    - interface PathChildrenCacheListener

      void childEvent(CuratorFramework client, PathChildrenCacheEvent event)

第四讲 开源客户端--- Curator介绍

- PathChildrenCache
  - 构造函数参数

参数名	说明
client	客户端实例
path	数据节点路径
dataIsCompressed	是否进行数据压缩
cacheData	用于配置是否把节点内容缓存起来，如果配置为true，那么客户端在接收到节点列表变更的同时，也能够获取到节点的数据内容；如果为false则无法取到数据内容
threadFactory	通过这两个参数构造专门的线程池来处理事件通知
executorService	

第四讲 开源客户端--- Curator介绍

- PathChildrenCache
  - 监听接口
    - 时间类型包括：新增子节点（CHILD_ADDED）,子节点数据变更（CHILD_UPDATED）,子节点删除（CHILD_REMOVED）
  - PathChildrenCache.StartMode
    - BUILD_INITIAL_CACHE //同步初始化客户端的cache，及创建cache后，就从服务器端拉入对应的数据
    - NORMAL //异步初始化cache
    - POST_INITIALIZED_EVENT //异步初始化，初始化完成触发事件PathChildrenCacheEvent.Type.*INITIALIZED*

第四讲 开源客户端--- Curator介绍

```
    public void addChildWatcher(String path) throws Exception {
        final PathChildrenCache cache = new PathChildrenCache(this.client,
                path, true);
        cache.start(StartMode.POST_INITIALIZED_EVENT);//ppt中需要讲StartMode
        System.out.println(cache.getCurrentData().size());
        //byte childone[] = cache.getCurrentData().get(0).getData();
//      System.out.println("childone:"
//              + cache.getCurrentData().get(0).getPath() + ";data="
//              + new String(childone));
        cache.getListenable().addListener(new PathChildrenCacheListener() {
            public void childEvent(CuratorFramework client,
                    PathChildrenCacheEvent event) throws Exception {
                if(event.getType().equals(PathChildrenCacheEvent.Type.INITIALIZED)){
                    System.out.println("客户端子节点cache初始化数据完成");
                }else if(event.getType().equals(PathChildrenCacheEvent.Type.CHILD_ADDED)){
                    System.out.println("添加子节点:"+event.getData().getPath());
                }else if(event.getType().equals(PathChildrenCacheEvent.Type.CHILD_REMOVED)){
                    System.out.println("删除子节点:"+event.getData().getPath());
                }else if(event.getType().equals(PathChildrenCacheEvent.Type.CHILD_UPDATED)){
                    System.out.println("修改子节点数据:"+event.getData().getPath());
                }
            }
        });
    }
```

第四讲 开源客户端--- Curator介绍

```
public static void main(String[] args) {
    CuratorClientTest ct = null;
    try {
        ct = new CuratorClientTest();
        // ct.createNode("/curator/test/node1", "test-node1".getBytes());
        ct.readNode("/curator/test/node1");
        ct.getChildren("/curator");
        // ct.updateNode("/curator/test/node1", "test-node1-new".getBytes(),
        // 0);
        // ct.readNode("/curator/test/node1");
        // ct.deleteNode("/curator", 0);

        ct.addNodeDataWatcher("/curator");
        ct.addChildWatcher("/curator");
        Thread.sleep(300000);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        ct.closeClient();
    }
```

```
添加子节点:/curator/test1
添加子节点:/curator/test
添加子节点:/curator/test6
添加子节点:/curator/test4
添加子节点:/curator/test5
添加子节点:/curator/test2
添加子节点:/curator/test3
客户端子节点cache初始化数据完成
添加子节点:/curator/test7
删除子节点:/curator/test6
修改子节点数据:/curator/test7
```

启动后初始化

异步初始化事件通知

添加节点

Thanks

FAQ时间