跟着实例学习ZooKeeper的用法： Curator框架应用

前面的几篇文章介绍了一些ZooKeeper的应用方法， 本文将介绍Curator访问ZooKeeper的一些基本方法， 而不仅仅限于指定的Recipes， 你可以使用Curator API任意的访问ZooKeeper。

**CuratorFramework**

Curator框架提供了一套高级的API， 简化了ZooKeeper的操作。 它增加了很多使用ZooKeeper开发的特性，可以处理ZooKeeper集群复杂的连接管理和重试机制。 这些特性包括：

* 自动化的连接管理: 重新建立到ZooKeeper的连接和重试机制存在一些潜在的错误case。 Curator帮助你处理这些事情，对你来说是透明的。
* 清理API:
  + 简化了原生的ZooKeeper的方法，事件等
  + 提供了一个现代的流式接口
* 提供了Recipes实现： 如前面的文章介绍的那样，基于这些Recipes可以创建很多复杂的分布式应用

Curator框架通过CuratorFrameworkFactory以工厂模式和builder模式创建CuratorFramework实 例。 CuratorFramework实例都是线程安全的，你应该在你的应用中共享同一个CuratorFramework实例.

工厂方法newClient()提供了一个简单方式创建实例。 而Builder提供了更多的参数控制。一旦你创建了一个CuratorFramework实例，你必须调用它的start()启动，在应用退出时调用close()方法关闭.

下面的例子演示了两种创建Curator的方法：

package com.colobu.zkrecipe.framework;

import org.apache.curator.RetryPolicy;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.retry.ExponentialBackoffRetry;

import org.apache.curator.test.TestingServer;

import org.apache.curator.utils.CloseableUtils;

public class CreateClientExample {

private static final String PATH = "/example/basic";

public static void main(String[] args) throws Exception {

TestingServer server = new TestingServer();

CuratorFramework client = null;

try {

client = createSimple(server.getConnectString());

client.start();

client.create().creatingParentsIfNeeded().forPath(PATH, "test".getBytes());

CloseableUtils.closeQuietly(client);

client = createWithOptions(server.getConnectString(), new ExponentialBackoffRetry(1000, 3), 1000, 1000);

client.start();

System.out.println(new String(client.getData().forPath(PATH)));

} catch (Exception ex) {

ex.printStackTrace();

} finally {

CloseableUtils.closeQuietly(client);

CloseableUtils.closeQuietly(server);

}

}

public static CuratorFramework createSimple(String connectionString) {

// these are reasonable arguments for the ExponentialBackoffRetry.

// The first retry will wait 1 second - the second will wait up to 2 seconds - the

// third will wait up to 4 seconds.

ExponentialBackoffRetry retryPolicy = new ExponentialBackoffRetry(1000, 3);

// The simplest way to get a CuratorFramework instance. This will use default values.

// The only required arguments are the connection string and the retry policy

return CuratorFrameworkFactory.newClient(connectionString, retryPolicy);

}

public static CuratorFramework createWithOptions(String connectionString, RetryPolicy retryPolicy, int connectionTimeoutMs, int sessionTimeoutMs) {

// using the CuratorFrameworkFactory.builder() gives fine grained control

// over creation options. See the CuratorFrameworkFactory.Builder javadoc details

return CuratorFrameworkFactory.builder().connectString(connectionString)

.retryPolicy(retryPolicy)

.connectionTimeoutMs(connectionTimeoutMs)

.sessionTimeoutMs(sessionTimeoutMs)

// etc. etc.

.build();

}

}

Curator框架提供了一种流式接口。 操作通过builder串联起来， 这样方法调用类似语句一样。

client.create().forPath("/head", new byte[0]);

client.delete().inBackground().forPath("/head");

client.create().withMode(CreateMode.EPHEMERAL\_SEQUENTIAL).forPath("/head/child", new byte[0]);

client.getData().watched().inBackground().forPath("/test");

CuratorFramework提供的方法：

| 方法名 | 描述 |
| --- | --- |
| create() | 开始创建操作， 可以调用额外的方法(比如方式mode 或者后台执行background) 并在最后调用forPath()指定要操作的ZNode |
| delete() | 开始删除操作. 可以调用额外的方法(版本或者后台处理version or background)并在最后调用forPath()指定要操作的ZNode |
| checkExists() | 开始检查ZNode是否存在的操作. 可以调用额外的方法(监控或者后台处理)并在最后调用forPath()指定要操作的ZNode |
| getData() | 开始获得ZNode节点数据的操作. 可以调用额外的方法(监控、后台处理或者获取状态watch, background or get stat) 并在最后调用forPath()指定要操作的ZNode |
| setData() | 开始设置ZNode节点数据的操作. 可以调用额外的方法(版本或者后台处理) 并在最后调用forPath()指定要操作的ZNode |
| getChildren() | 开始获得ZNode的子节点列表。 以调用额外的方法(监控、后台处理或者获取状态watch, background or get stat) 并在最后调用forPath()指定要操作的ZNode |
| inTransaction() | 开始是原子ZooKeeper事务. 可以复合create, setData, check, and/or delete 等操作然后调用commit()作为一个原子操作提交 |

后台操作的通知和监控可以通过ClientListener接口发布. 你可以在CuratorFramework实例上通过addListener()注册listener, Listener实现了下面的方法:

* eventReceived() 一个后台操作完成或者一个监控被触发

事件类型以及事件的方法如下：

| Event Type | Event Methods |
| --- | --- |
| CREATE | getResultCode() and getPath() |
| DELETE | getResultCode() and getPath() |
| EXISTS | getResultCode(), getPath() and getStat() |
| GETDATA | getResultCode(), getPath(), getStat() and getData() |
| SETDATA | getResultCode(), getPath() and getStat() |
| CHILDREN | getResultCode(), getPath(), getStat(), getChildren() |
| WATCHED | getWatchedEvent() |

还可以通过ConnectionStateListener接口监控连接的状态。 强烈推荐你增加这个监控器。

你可以使用命名空间Namespace避免多个应用的节点的名称冲突。 CuratorFramework提供了命名空间的概念，这样CuratorFramework会为它的API调用的path加上命名空间：

CuratorFramework client = CuratorFrameworkFactory.builder().namespace("MyApp") ... build();

...

client.create().forPath("/test", data);

// node was actually written to: "/MyApp/test"

Curator还提供了临时的CuratorFramework： CuratorTempFramework， 一定时间不活动后连接会被关闭。这hi基于Camille Fournier的一篇文章： <http://whilefalse.blogspot.com/2012/12/building-global-highly-available.html>.

创建builder时不是调用build()而是调用buildTemp()。 3分钟不活动连接就被关闭，你也可以指定不活动的时间。 它只提供了下面几个方法：

public void close();

public CuratorTransaction inTransaction() throws Exception;

public TempGetDataBuilder getData() throws Exception;

**操作方法**

上面的表格列出了CuratorFramework可以用的操作。 下面就是一个例子：

package com.colobu.zkrecipe.framework;

import java.util.List;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.api.BackgroundCallback;

import org.apache.curator.framework.api.CuratorEvent;

import org.apache.curator.framework.api.CuratorListener;

import org.apache.zookeeper.CreateMode;

import org.apache.zookeeper.Watcher;

public class CrudExample {

public static void main(String[] args) {

}

public static void create(CuratorFramework client, String path, byte[] payload) throws Exception {

// this will create the given ZNode with the given data

client.create().forPath(path, payload);

}

public static void createEphemeral(CuratorFramework client, String path, byte[] payload) throws Exception {

// this will create the given EPHEMERAL ZNode with the given data

client.create().withMode(CreateMode.EPHEMERAL).forPath(path, payload);

}

public static String createEphemeralSequential(CuratorFramework client, String path, byte[] payload) throws Exception {

// this will create the given EPHEMERAL-SEQUENTIAL ZNode with the given

// data using Curator protection.

return client.create().withProtection().withMode(CreateMode.EPHEMERAL\_SEQUENTIAL).forPath(path, payload);

}

public static void setData(CuratorFramework client, String path, byte[] payload) throws Exception {

// set data for the given node

client.setData().forPath(path, payload);

}

public static void setDataAsync(CuratorFramework client, String path, byte[] payload) throws Exception {

// this is one method of getting event/async notifications

CuratorListener listener = new CuratorListener() {

@Override

public void eventReceived(CuratorFramework client, CuratorEvent event) throws Exception {

// examine event for details

}

};

client.getCuratorListenable().addListener(listener);

// set data for the given node asynchronously. The completion

// notification

// is done via the CuratorListener.

client.setData().inBackground().forPath(path, payload);

}

public static void setDataAsyncWithCallback(CuratorFramework client, BackgroundCallback callback, String path, byte[] payload) throws Exception {

// this is another method of getting notification of an async completion

client.setData().inBackground(callback).forPath(path, payload);

}

public static void delete(CuratorFramework client, String path) throws Exception {

// delete the given node

client.delete().forPath(path);

}

public static void guaranteedDelete(CuratorFramework client, String path) throws Exception {

// delete the given node and guarantee that it completes

client.delete().guaranteed().forPath(path);

}

public static List<String> watchedGetChildren(CuratorFramework client, String path) throws Exception {

/\*\*

\* Get children and set a watcher on the node. The watcher notification

\* will come through the CuratorListener (see setDataAsync() above).

\*/

return client.getChildren().watched().forPath(path);

}

public static List<String> watchedGetChildren(CuratorFramework client, String path, Watcher watcher) throws Exception {

/\*\*

\* Get children and set the given watcher on the node.

\*/

return client.getChildren().usingWatcher(watcher).forPath(path);

}

}

**事务**

上面也提到， CuratorFramework提供了事务的概念，可以将一组操作放在一个原子事务中。 什么叫事务？ 事务是原子的， 一组操作要么都成功，要么都失败。

下面的例子演示了事务的操作：

package com.colobu.zkrecipe.framework;

import java.util.Collection;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.api.transaction.CuratorTransaction;

import org.apache.curator.framework.api.transaction.CuratorTransactionFinal;

import org.apache.curator.framework.api.transaction.CuratorTransactionResult;

public class TransactionExample {

public static void main(String[] args) {

}

public static Collection<CuratorTransactionResult> transaction(CuratorFramework client) throws Exception {

// this example shows how to use ZooKeeper's new transactions

Collection<CuratorTransactionResult> results = client.inTransaction().create().forPath("/a/path", "some data".getBytes())

.and().setData().forPath("/another/path", "other data".getBytes())

.and().delete().forPath("/yet/another/path")

.and().commit(); // IMPORTANT!

// called

for (CuratorTransactionResult result : results) {

System.out.println(result.getForPath() + " - " + result.getType());

}

return results;

}

/\*

\* These next four methods show how to use Curator's transaction APIs in a

\* more traditional - one-at-a-time - manner

\*/

public static CuratorTransaction startTransaction(CuratorFramework client) {

// start the transaction builder

return client.inTransaction();

}

public static CuratorTransactionFinal addCreateToTransaction(CuratorTransaction transaction) throws Exception {

// add a create operation

return transaction.create().forPath("/a/path", "some data".getBytes()).and();

}

public static CuratorTransactionFinal addDeleteToTransaction(CuratorTransaction transaction) throws Exception {

// add a delete operation

return transaction.delete().forPath("/another/path").and();

}

public static void commitTransaction(CuratorTransactionFinal transaction) throws Exception {

// commit the transaction

transaction.commit();

}

}