# 介绍

Curator是Netflix公司开源的一个Zookeeper客户端，2013年2月提出与2013年3月被正式录取进入孵化器，2013年9月毕业于顶级项目。与Zookeeper提供的原生客户端相比，Curator的抽象层次更高，简化了Zookeeper客户端的开发量。

|  |  |
| --- | --- |
| 模块 | 简介 |
| Framework | 简化Zookeeper的使用,提供了一系列高阶API, 在Zookeeper集群连接的管理, 以及重连操作上化繁为简. |
| Receipts | 实现了一些ZooKeeper的基本"模式", 该实现基于Curator Framework. |
| Utilities | 一系列工具 |
| Client | ZooKeeper类的替代 |
| Errors | 定义了curator如何处理错误,连接问题,可恢复的异常等. |
| Extensions | 实现了一些Zookeeper文档中提及的通用模式. |

版本支持

Curator 2.x.x - ZooKeeper 3.4.x, ZooKeeper 3.5.x

Curator 3.x.x - 只兼容ZooKeeper 3.5.x; 支持dynamic reconfiguration等新feature.

# Zookeeper安装部署

Zookeeper的部署很简单，如果已经有Java运行环境的话，下载tarball解压后即可运行。

[root@vm Temp]$ wget http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.4.6/zookeeper-3.4.6.tar.gz

[root@vm Temp]$ tar zxvf zookeeper-3.4.6.tar.gz

[root@vm Temp]$ cd zookeeper-3.4.6

[root@vm zookeeper-3.4.6]$ cp conf/zoo\_sample.cfg conf/zoo.cfg

[root@vm zookeeper-3.4.6]$ export ZOOKEEPER\_HOME=/usr/local/src/zookeeper-3.4.5

[root@vm zookeeper-3.4.6]$ export PATH=$ZOOKEEPER\_HOME/bin:$PATH

[root@vm zookeeper-3.4.6]$ bin/zkServer.sh start

[root@vm zookeeper-3.4.6]$ bin/zkCli.sh -server 127.0.0.1:2181

# 客户端常用操作

用zkCli.sh连接上Zookeeper服务后，用help能列出所有命令：

[root@BC-VM-edce4ac67d304079868c0bb265337bd4 zookeeper-3.4.6]# bin/zkCli.sh -127.0.0.1:2181

Connecting to localhost:2181

2015-06-11 10:55:14,387 [myid:] - INFO [main:Environment@100] - Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09 GMT

...

[zk: localhost:2181(CONNECTED) 5] help

ZooKeeper -server host:port cmd args

connect host:port

get path [watch]

ls path [watch]

set path data [version]

rmr path

delquota [-n|-b] path

quit

printwatches on|off

create [-s] [-e] path data acl

stat path [watch]

close

ls2 path [watch]

history

listquota path

setAcl path acl

getAcl path

sync path

redo cmdno

addauth scheme auth

delete path [version]

setquota -n|-b val path

下面就试验一下常用的命令：

create：创建路径结点。

ls：查看路径下的所有结点。

get：获得结点上的值。

set：修改结点上的值。

delete：删除结点。

[zk: localhost:2181(CONNECTED) 6] create /zktest mydata

Created /zktest

[zk: localhost:2181(CONNECTED) 12] ls /

[zktest, zookeeper]

[zk: localhost:2181(CONNECTED) 7] ls /zktest

[]

[zk: localhost:2181(CONNECTED) 13] get /zktest

mydata

cZxid = 0x1c

ctime = Thu Jun 11 10:58:06 CST 2015

mZxid = 0x1c

mtime = Thu Jun 11 10:58:06 CST 2015

pZxid = 0x1c

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 6

numChildren = 0

[zk: localhost:2181(CONNECTED) 14] set /zktest junk

cZxid = 0x1c

ctime = Thu Jun 11 10:58:06 CST 2015

mZxid = 0x1f

mtime = Thu Jun 11 10:59:08 CST 2015

pZxid = 0x1c

cversion = 0

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 4

numChildren = 0

[zk: localhost:2181(CONNECTED) 15] delete /zktest

[zk: localhost:2181(CONNECTED) 16] ls /

[zookeeper]

# 用Curator管理Zookeeper

## 介绍

首先, 获取一个连接实例(CuratorFramework). 对于每个集群, 只需要创建一个实例.

CuratorFrameworkFactory.newClient(zookeeperConnectionString, retryPolicy)

这里我们需要指定一个retryPolicy, 这里用一个指数补偿策略:

RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3)

CuratorFramework client = CuratorFrameworkFactory.newClient(zookeeperConnectionString, retryPolicy);

client.start();

然后我们就可以调用Zookeeper的一系列操作了, 比如创建:

client.create().forPath("/my/path", myData)

这里如果出现了连接问题,curator也会帮你自动重连

Curator的Maven依赖如下，一般直接使用curator-recipes就行了，如果需要自己封装一些底层些的功能的话，例如增加连接管理重试机制等，则可以引入curator-framework包。

## Pom依赖

<dependency>

<groupId>org.apache.curator</groupId>

<artifactId>curator-recipes</artifactId>

<version>2.7.0</version>

</dependency>

## Client操作

利用Curator提供的客户端API，可以完全实现上面原生客户端的功能。值得注意的是，Curator采用流式风格API。

package com.cdai.codebase.bigdata.hadoop.zookeeper.curator;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.retry.RetryNTimes;

/\*\*

\* Curator framework's client test.

\* Output:

\* $ create /zktest hello

\* $ ls /

\* [zktest, zookeeper]

\* $ get /zktest

\* hello

\* $ set /zktest world

\* $ get /zktest

\* world

\* $ delete /zktest

\* $ ls /

\* [zookeeper]

\*/

public class CuratorClientTest {

/\*\* Zookeeper info \*/

private static final String ZK\_ADDRESS = "192.168.1.100:2181";

private static final String ZK\_PATH = "/zktest";

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

// 1.Connect to zk

CuratorFramework client = CuratorFrameworkFactory.newClient(

ZK\_ADDRESS,

new RetryNTimes(10, 5000)

);

client.start();

System.out.println("zk client start successfully!");

// 2.Client API test

// 2.1 Create node

String data1 = "hello";

print("create", ZK\_PATH, data1);

client.create().

creatingParentsIfNeeded().

forPath(ZK\_PATH, data1.getBytes());

// 2.2 Get node and data

print("ls", "/");

print(client.getChildren().forPath("/"));

print("get", ZK\_PATH);

print(client.getData().forPath(ZK\_PATH));

// 2.3 Modify data

String data2 = "world";

print("set", ZK\_PATH, data2);

client.setData().forPath(ZK\_PATH, data2.getBytes());

print("get", ZK\_PATH);

print(client.getData().forPath(ZK\_PATH));

// 2.4 Remove node

print("delete", ZK\_PATH);

client.delete().forPath(ZK\_PATH);

print("ls", "/");

print(client.getChildren().forPath("/"));

}

private static void print(String... cmds) {

StringBuilder text = new StringBuilder("$ ");

for (String cmd : cmds) {

text.append(cmd).append(" ");

}

System.out.println(text.toString());

}

private static void print(Object result) {

System.out.println(

result instanceof byte[]

? new String((byte[]) result)

: result);

}

}

## 监听器

Curator提供了三种Watcher(Cache)来监听结点的变化：

Path Cache：监视一个路径下1）孩子结点的创建、2）删除，3）以及结点数据的更新。产生的事件会传递给注册的PathChildrenCacheListener。

Node Cache：监视一个结点的创建、更新、删除，并将结点的数据缓存在本地。

Tree Cache：Path Cache和Node Cache的“合体”，监视路径下的创建、更新、删除事件，并缓存路径下所有孩子结点的数据。

下面就测试一下最简单的Path Watcher：

package com.cdai.codebase.bigdata.hadoop.zookeeper.curator;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.framework.recipes.cache.ChildData;

import org.apache.curator.framework.recipes.cache.PathChildrenCache;

import org.apache.curator.framework.recipes.cache.PathChildrenCache.StartMode;

import org.apache.curator.retry.RetryNTimes;

/\*\*

\* Curator framework watch test.

\*/

public class CuratorWatcherTest {

/\*\* Zookeeper info \*/

private static final String ZK\_ADDRESS = "192.168.1.100:2181";

private static final String ZK\_PATH = "/zktest";

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

// 1.Connect to zk

CuratorFramework client = CuratorFrameworkFactory.newClient(

ZK\_ADDRESS,

new RetryNTimes(10, 5000)

);

client.start();

System.out.println("zk client start successfully!");

// 2.Register watcher

PathChildrenCache watcher = new PathChildrenCache(

client,

ZK\_PATH,

true // if cache data

);

watcher.getListenable().addListener((client1, event) -> {

ChildData data = event.getData();

if (data == null) {

System.out.println("No data in event[" + event + "]");

} else {

System.out.println("Receive event: "

+ "type=[" + event.getType() + "]"

+ ", path=[" + data.getPath() + "]"

+ ", data=[" + new String(data.getData()) + "]"

+ ", stat=[" + data.getStat() + "]");

}

});

watcher.start(StartMode.BUILD\_INITIAL\_CACHE);

System.out.println("Register zk watcher successfully!");

Thread.sleep(Integer.MAX\_VALUE);

}

}

下面是在zkCli.sh中操作时Java程序的输出：

Java: zk client start successfully!

Java: Register zk watcher successfully!

zkCli: [zk: localhost:2181(CONNECTED) 11] create /zktest/hello mydata

Java: Receive event: type=[CHILD\_ADDED], path=[/zktest/hello], data=[mydata], stat=[121,121,1434001221097,1434001221097,0,0,0,0,6,0,121]

zkCli: [zk: localhost:2181(CONNECTED) 12] set /zktest/hello otherdata

Java: Receive event: type=[CHILD\_UPDATED], path=[/zktest/hello], data=[otherdata], stat=[121,122,1434001221097,1434001228467,1,0,0,0,9,0,121]

zkCli: [zk: localhost:2181(CONNECTED) 13] delete /zktest/hello

Java: Receive event: type=[CHILD\_REMOVED], path=[/zktest/hello], data=[otherdata], stat=[121,122,1434001221097,1434001228467,1,0,0,0,9,0,121]

# 功能

既然Maven包叫做curator-recipes，那说明Curator有它独特的“菜谱”：

* 锁：包括共享锁、共享可重入锁、读写锁等。
* 选举：Leader选举算法。
* Barrier：阻止分布式计算直至某个条件被满足的“栅栏”，可以看做JDK Concurrent包中Barrier的分布式实现。
* 缓存：前面提到过的三种Cache及监听机制。
* 持久化结点：连接或Session终止后仍然在Zookeeper中存在的结点。
* 队列：分布式队列、分布式优先级队列等。

## 分布式锁

分布式编程时，比如最容易碰到的情况就是应用程序在线上多机部署，于是当多个应用同时访问某一资源时，就需要某种机制去协调它们。例如，现在一台应用正在rebuild缓存内容，要临时锁住某个区域暂时不让访问；又比如调度程序每次只想一个任务被一台应用执行等等。

下面的程序会启动两个线程t1和t2去争夺锁，拿到锁的线程会占用5秒。运行多次可以观察到，有时是t1先拿到锁而t2等待，有时又会反过来。Curator会用我们提供的lock路径的结点作为全局锁，这个结点的数据类似这种格式：[\_c\_64e0811f-9475-44ca-aa36-c1db65ae5350-lock-0000000005]，每次获得锁时会生成这种串，释放锁时清空数据。

package com.cdai.codebase.bigdata.hadoop.zookeeper.curator;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.framework.recipes.locks.InterProcessMutex;

import org.apache.curator.retry.RetryNTimes;

import java.util.concurrent.TimeUnit;

/\*\*

\* Curator framework's distributed lock test.

\*/

public class CuratorDistrLockTest {

/\*\* Zookeeper info \*/

private static final String ZK\_ADDRESS = "192.168.1.100:2181";

private static final String ZK\_LOCK\_PATH = "/zktest";

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

// 1.Connect to zk

CuratorFramework client = CuratorFrameworkFactory.newClient(

ZK\_ADDRESS,

new RetryNTimes(10, 5000)

);

client.start();

System.out.println("zk client start successfully!");

Thread t1 = new Thread(() -> {

doWithLock(client);

}, "t1");

Thread t2 = new Thread(() -> {

doWithLock(client);

}, "t2");

t1.start();

t2.start();

}

private static void doWithLock(CuratorFramework client) {

InterProcessMutex lock = new InterProcessMutex(client, ZK\_LOCK\_PATH);

try {

if (lock.acquire(10 \* 1000, TimeUnit.SECONDS)) {

System.out.println(Thread.currentThread().getName() + " hold lock");

Thread.sleep(5000L);

System.out.println(Thread.currentThread().getName() + " release lock");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

lock.release();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

## Leader选举

当集群里的某个服务down机时，我们可能要从slave结点里选出一个作为新的master，这时就需要一套能在分布式环境中自动协调的Leader选举方法。Curator提供了LeaderSelector监听器实现Leader选举功能。同一时刻，只有一个Listener会进入takeLeadership()方法，说明它是当前的Leader。注意：当Listener从takeLeadership()退出时就说明它放弃了“Leader身份”，这时Curator会利用Zookeeper再从剩余的Listener中选出一个新的Leader。autoRequeue()方法使放弃Leadership的Listener有机会重新获得Leadership，如果不设置的话放弃了的Listener是不会再变成Leader的。

package com.cdai.codebase.bigdata.hadoop.zookeeper.curator;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.framework.recipes.leader.LeaderSelector;

import org.apache.curator.framework.recipes.leader.LeaderSelectorListener;

import org.apache.curator.framework.state.ConnectionState;

import org.apache.curator.retry.RetryNTimes;

import org.apache.curator.utils.EnsurePath;

/\*\*

\* Curator framework's leader election test.

\* Output:

\* LeaderSelector-2 take leadership!

\* LeaderSelector-2 relinquish leadership!

\* LeaderSelector-1 take leadership!

\* LeaderSelector-1 relinquish leadership!

\* LeaderSelector-0 take leadership!

\* LeaderSelector-0 relinquish leadership!

\* ...

\*/

public class CuratorLeaderTest {

/\*\* Zookeeper info \*/

private static final String ZK\_ADDRESS = "192.168.1.100:2181";

private static final String ZK\_PATH = "/zktest";

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

LeaderSelectorListener listener = new LeaderSelectorListener() {

@Override

public void takeLeadership(CuratorFramework client) throws Exception {

System.out.println(Thread.currentThread().getName() + " take leadership!");

// takeLeadership() method should only return when leadership is being relinquished.

Thread.sleep(5000L);

System.out.println(Thread.currentThread().getName() + " relinquish leadership!");

}

@Override

public void stateChanged(CuratorFramework client, ConnectionState state) {

}

};

new Thread(() -> {

registerListener(listener);

}).start();

new Thread(() -> {

registerListener(listener);

}).start();

new Thread(() -> {

registerListener(listener);

}).start();

Thread.sleep(Integer.MAX\_VALUE);

}

private static void registerListener(LeaderSelectorListener listener) {

// 1.Connect to zk

CuratorFramework client = CuratorFrameworkFactory.newClient(

ZK\_ADDRESS,

new RetryNTimes(10, 5000)

);

client.start();

// 2.Ensure path

try {

new EnsurePath(ZK\_PATH).ensure(client.getZookeeperClient());

} catch (Exception e) {

e.printStackTrace();

}

// 3.Register listener

LeaderSelector selector = new LeaderSelector(client, ZK\_PATH, listener);

selector.autoRequeue();

selector.start();

}

}

## 事务

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! The transaction is not submitted until commit() is called